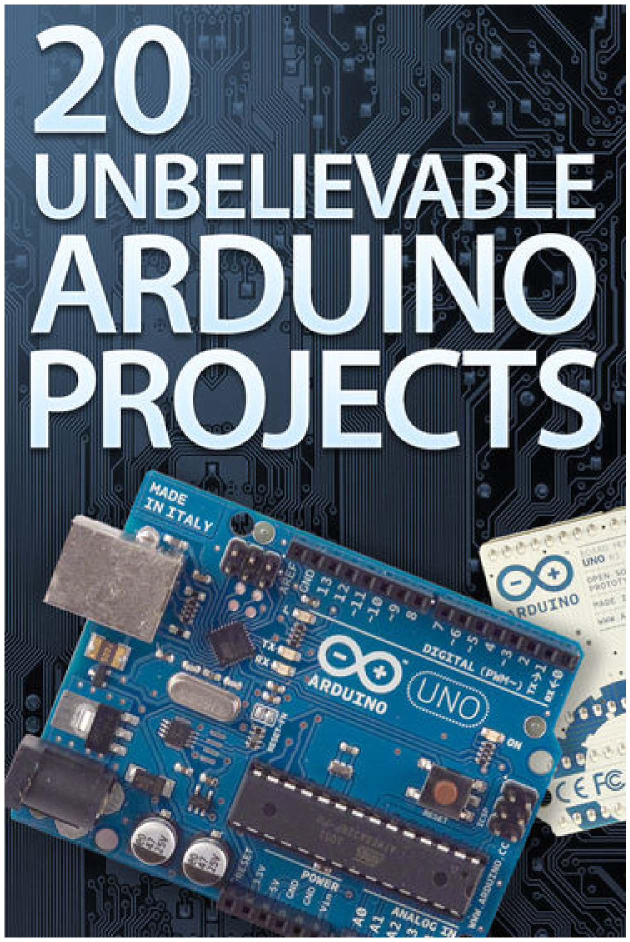


Introduction

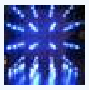
Here are twenty amazing Arduino projects that you almost wouldn't believe, if not for that they are the real deal. These authors have turned their wildest dreams into reality with the power of Arduino, an easy-to-use microcontroller development board. It is no wonder that Arduino literally translates to "Strong friend (masculine)" in Italian. Anything is possible with the mighty power of Arduino. It's compact, it's straightforward, and makes embedding electronics into the world-at-large fun and easy. Check out some of these amazing projects, and get inspired to build your own reality.

Table of Contents

Author and Copyright Notices

LED Cube 8x8x8

by **chr** on November 16, 2010



Author:chr
I like microcontrollers and LEDs :D

Intro: LED Cube 8x8x8

Create your own 8x8x8 LED Cube 3-dimensional display!

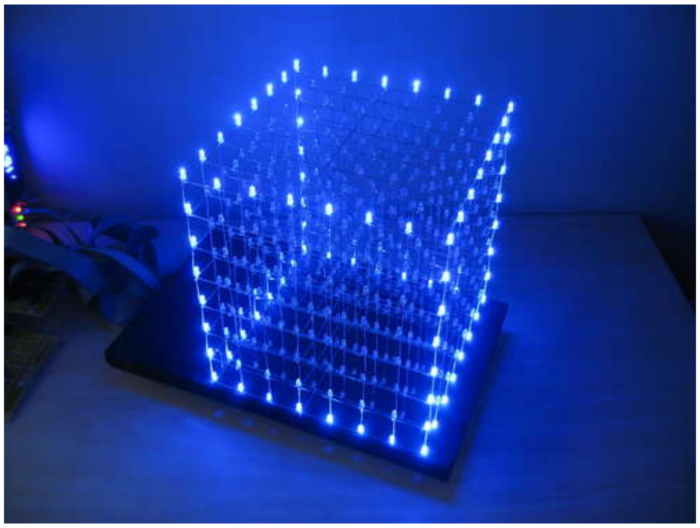
We believe this Instructable is the most comprehensive step-by-step guide to build an 8x8x8 LED Cube ever published on the intertubes. It will teach you everything from theory of operation, how to build the cube, to the inner workings of the software. We will take you through the software step by step, both the low level drivers/routines and how to create awesome animations. The software aspect of LED cubes is often overlooked, but a LED cube is only as awesome as the software it runs.

About halfway through the Instructable, you will actually have a fully functional LED cube. The remaining steps will show you how to create the software.

A video is worth a thousand words. I'll just leave it up to this video to convince you that this is the next project you will be building:

I made this LED cube together with my friend chiller. The build took about 4 days from small scale prototyping to completed cube. Then another couple of hours to debug some faulty transistors.

The software is probably another 4-5 days of work combined.



Step 1: Skills required

At first glance this project might seem like an overly complex and daunting task. However, we are dealing with digital electronics here, so everything is either on or off!

I've been doing electronics for a long time, and for years i struggled with analog circuits. The analog circuits failed over half the time even if i followed instructions. One resistor or capacitor with a slightly wrong value, and the circuit doesn't work.

About 4 years ago, I decided to give microcontrollers a try. This completely changed my relationship with electronics. I went from only being able to build simple analog circuits, to being able to build almost anything!

A digital circuit doesn't care if a resistor is 1k ohm or 2k ohm, as long as it can distinguish high from low. And believe me, this makes it A LOT easier to do electronics!

With that said, there are still some things you should know before venturing out and building this rather large project.

You should have an understanding of:

- Basic electronics. (We would recommend against building this as your very first electronics project. But please read the Instructable. You'll still learn a lot!)
- How to solder.
- How to use a multimeter etc.
- Writing code in C (optional. We provide a fully functional program, ready to go)

You should also have patience and a generous amount of free time.

Step 2: Component list

Here is what you need to make a LED cube:

- 512x LEDs (plus some extra for making mistakes!)
- 64x resistors. (see separate step for ohm value)
- 1x or 2x large prototype PCBs. The type with copper "eyes", see image.
- 1x ATmega32 microcontroller (you can also use the pin-compatible ATmega16)
- 3x status LEDs. You choose color and size.
- 3x resistors for the status LEDs.
- 8x 74HC574 ICs
- 16x PN2222 transistors
- 16x 1k resistors
- 1x 74HC138 IC
- 1x Maxim MAX232 IC
- 1x 14.7456 MHz crustal
- 2x 22pF ceramic capacitors
- 16x 0.1uF ceramic capacitors
- 3x 1000uF electrolytic capacitor
- 3x 10uF electrolytic capacitor
- 1x 100uF electrolytic capacitors
- 8x 20 pin IC sockets
- 1x 40 pin IC socket
- 2x 16 pin IC socket
- 1x 2-pin screw terminal
- 1x 2wire cable with plugs
- 9x 8-pin terminal pins
- 1x 4-pin terminal pins, right angle
- 2x 16-pin ribbon cable connector
- 1x 10-pin ribbon cable connector
- Ribbon cable
- 2x pushbuttons
- 2x ribbon cable plugs
- 9x 8-pin female header plugs
- Serial cable and 4pin female pin header
- Piece of wood for template and base
- 8x optional pull-up resistors for layers
- 5v power supply (see separate step for power supply)

Total estimated build cost: 67 USD. See attached price list.

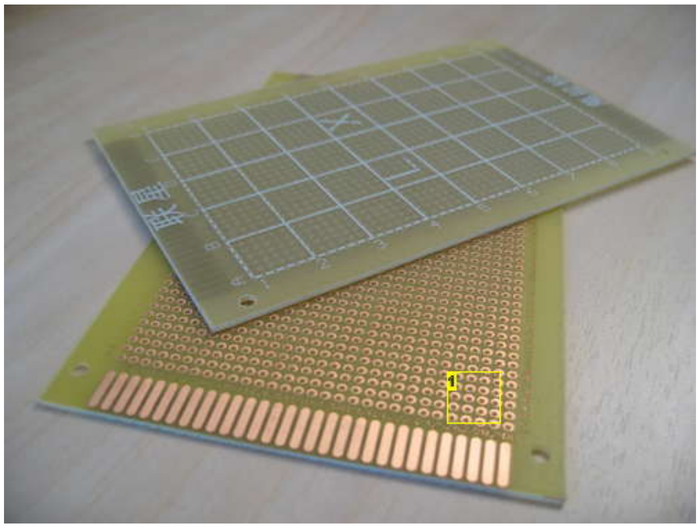


Image Notes
1. Make sure to get this type of prototyping PCB.

Image Notes
1. Lots and lots of LEDs!

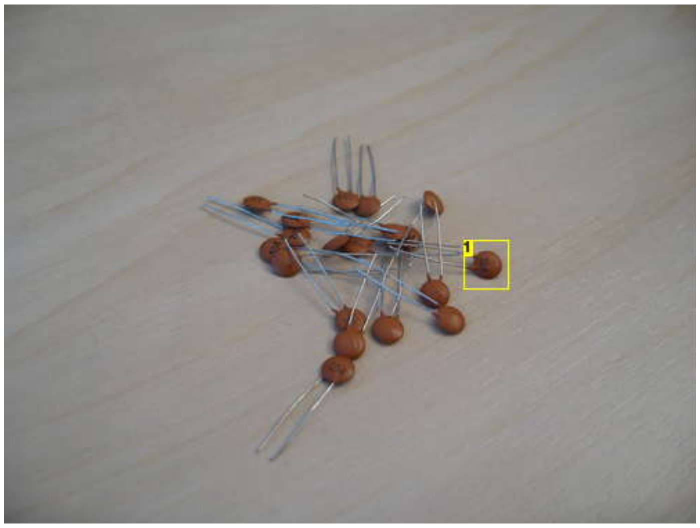


Image Notes
1. 100nF

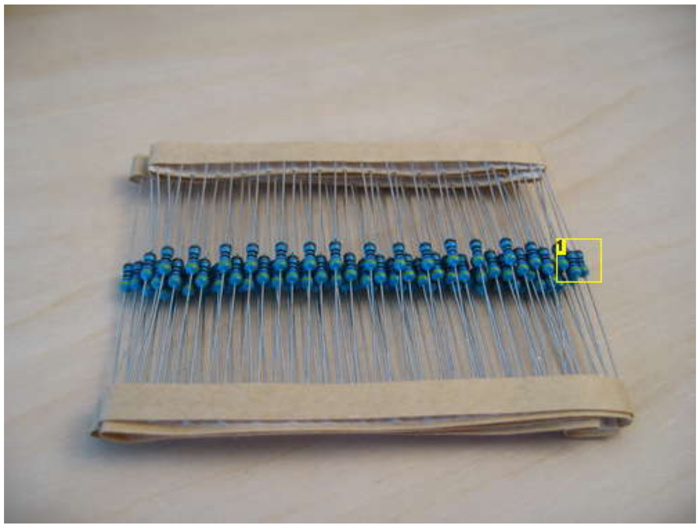


Image Notes
1. Don't look at the color codes. This is not 100ohms.

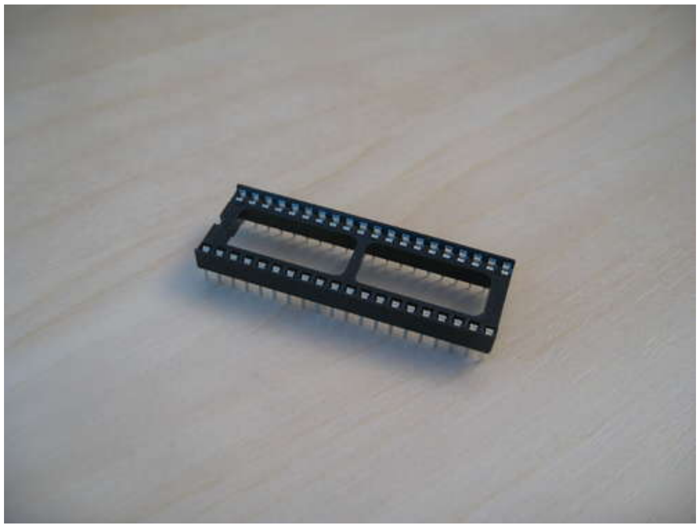


Image Notes
1. Kynar wrapping wire. 30 AWG.

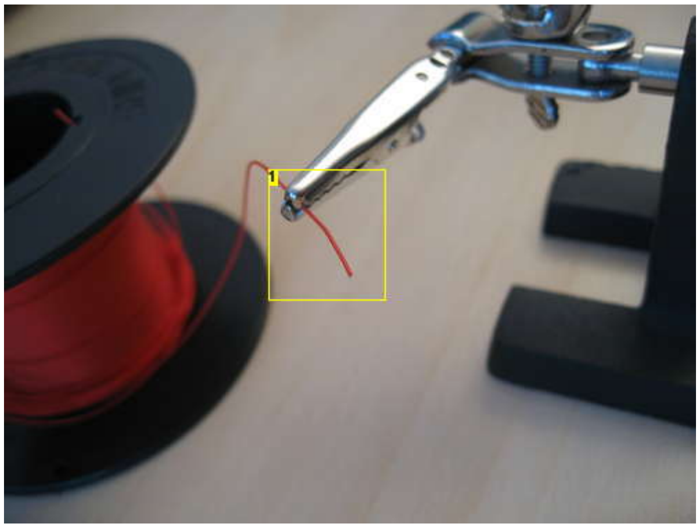


Image Notes
1. Very tiny wire. Perfect for working on prototyping PCBs.

Image Notes
1. Fan to blow away those soldering fumes.

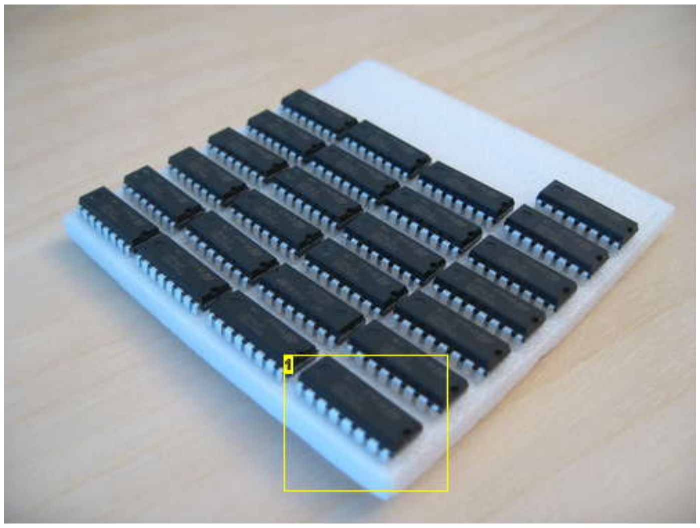
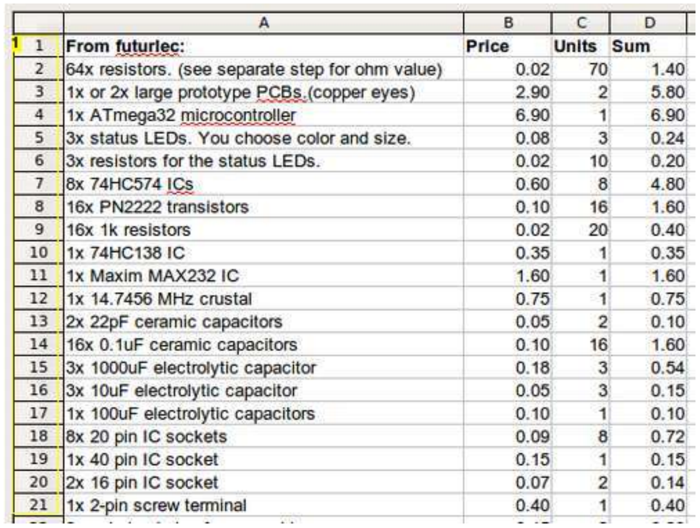


Image Notes
1. Lots of ICs



	A	B	C	D
1	**From futurlec:**	**Price**	**Units**	**Sum**
2	64x resistors. (see separate step for ohm value)	0.02	70	1.40
3	1x or 2x large prototype PCBs.(copper eyes)	2.90	2	5.80
4	1x ATmega32 microcontroller	6.90	1	6.90
5	3x status LEDs. You choose color and size.	0.08	3	0.24
6	3x resistors for the status LEDs.	0.02	10	0.20
7	8x 74HC574 ICs	0.60	8	4.80
8	16x PN2222 transistors	0.10	16	1.60
9	16x 1k resistors	0.02	20	0.40
10	1x 74HC138 IC	0.35	1	0.35
11	1x Maxim MAX232 IC	1.60	1	1.60
12	1x 14.7456 MHz crustal	0.75	1	0.75
13	2x 22pF ceramic capacitors	0.05	2	0.10
14	16x 0.1uF ceramic capacitors	0.10	16	1.60
15	3x 1000uF electrolytic capacitor	0.18	3	0.54
16	3x 10uF electrolytic capacitor	0.05	3	0.15
17	1x 100uF electrolytic capacitors	0.10	1	0.10
18	8x 20 pin IC sockets	0.09	8	0.72
19	1x 40 pin IC socket	0.15	1	0.15
20	2x 16 pin IC socket	0.07	2	0.14
21	1x 2-pin screw terminal	0.40	1	0.40

Image Notes
1. See attached excel file for full list.

File Downloads

pricelist.xls (12 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'pricelist.xls']

Step 3: Ordering components

We see a lot of people asking for part numbers for DigiKey, Mouser or other big electronics stores.

When you're working with hobby electronics, you don't necessarily need the most expensive components with the best quality.

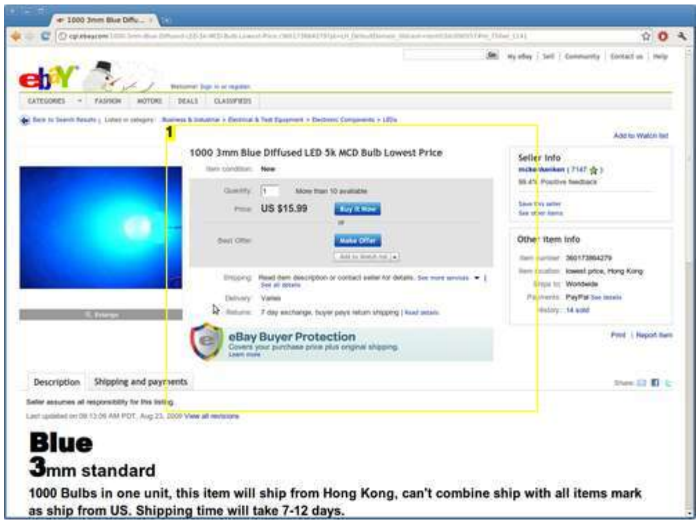
Most of the time, it is more important to actually have the component value at hand when you need it.

We are big fans of buying really cheap component lots on eBay. You can get assortments of resistor, capacitors, transistors and everything in between. If you buy these types of assortments, you will almost always have the parts you need in your part collection.

For 17 USD you can get 2000 resistors of 50 different values. Great value, and very convenient.

Try doing som eBay searches and buy some components for future projects!

Another one of our favorite stores is Futurlec (http://www.futurlec.com/). They have everything you need. The thing they don't have is 1000 different versions of that thing that you need, so browsing their inventory is a lot less confusing than buying from those bigger companies.


1000 3mm Blue Diffused LED 5k MCD Bulb Lowest Price

Item condition: New

Quantity: 1 More than 10 available

Price: US $15.99

Seller info

Other item info

Item location: lowest price, Hong Kong

Ships to: Worldwide

Payments: PayPal

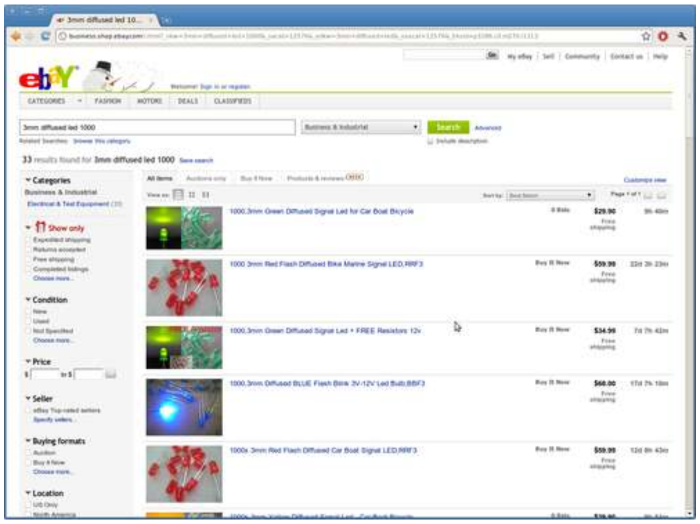
eBay Buyer Protection

Description | Shipping and payments

Blue

3mm standard

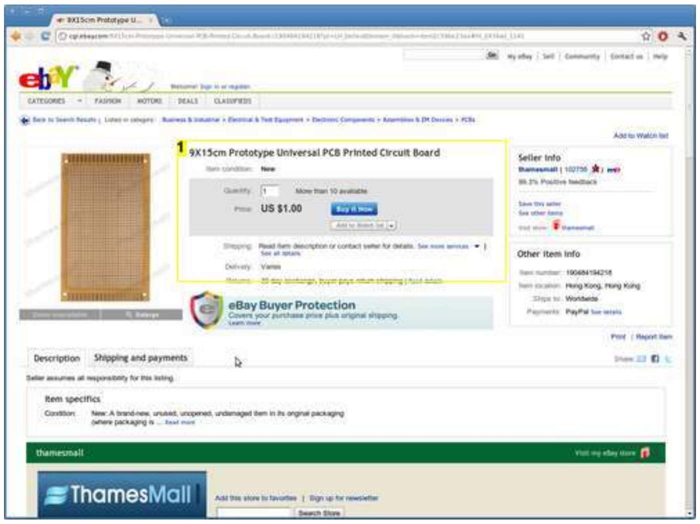
1000 Bulbs in one unit, this item will ship from Hong Kong, can't combine ship with all items mark as ship from US. Shipping time will take 7-12 days.


3mm diffused led 1000

33 results found for 3mm diffused led 1000

1000,3mm Green Diffused Signal Led for Car Boat Bicycle	$29.90	
1000 3mm Red Flash Diffused Bike Marine Signal LED,RRF3	$59.99	
1000,3mm Green Diffused Signal Led + FREE Resistors 12v	$34.99	
1000,3mm Diffused BLUE Flash Blink 3V-12V Led Bulb,BBF3	$68.00	
1000x 3mm Red Flash Diffused Car Boat Signal LED,RRF3	$59.99	

Image Notes
1. 1000 leds for 16 bucks. But beware! The descriptions aren't always that great. We ordered diffused leds and got clear ones :/


9X15cm Prototype Universal PCB Printed Circuit Board

Item condition: New

Quantity: 1 More than 10 available

Price: US $1.00

Other item info

Item location: Hong Kong, Hong Kong

Ships to: Worldwide

Payments: PayPal

eBay Buyer Protection

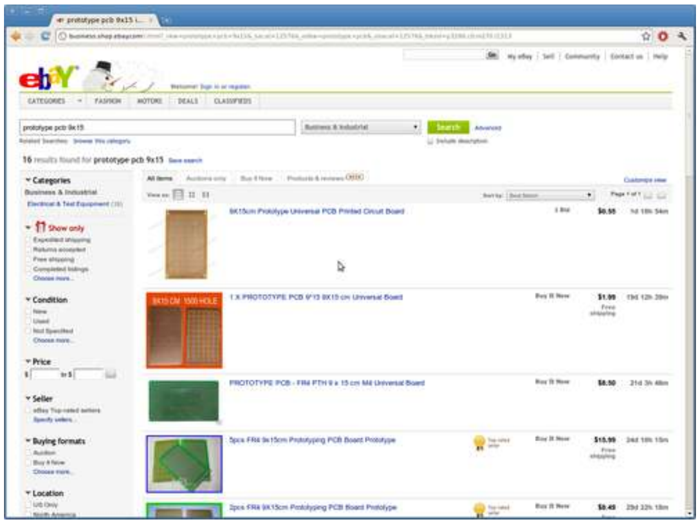
Description | Shipping and payments

Item specifics

Condition: New: A brand-new, unused, unopened, undamaged item in its original packaging (where packaging is ... Read more

thamesmall

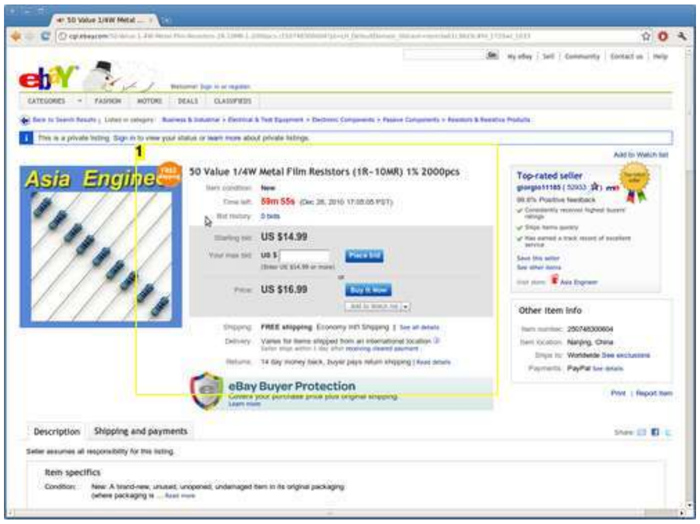
ThamesMall


prototype pcb 9x15

16 results found for prototype pcb 9x15

9X15cm Prototype Universal PCB Printed Circuit Board	$0.55
1 X PROTOTYPE PCB 9*15 9X15 cm Universal Board	$1.99
PROTOTYPE PCB - FR4 PTH 9 x 15 cm Mil Universal Board	$8.50
5pcs FR4 9x15cm Prototyping PCB Board Prototype	$15.99
2pcs FR4 9X15cm Prototyping PCB Board Prototype	$8.49

Image Notes
1. This is the type of prototype PCB we used. 1 dollar!



This is a private listing. Sign in to view your status or learn more about private listings.

1

50 Value 1/4W Metal Film Resistors (1R-10MR) 1% 2000pcs

Item condition: New
Time left: 59m 55s (Dec 28, 2010 17:08:05 PST)
Bid history: 0 bids
Starting bid: US $14.99
Your max bid: US $
Price: US $16.99
Shipping: FREE shipping Economy Int'l Shipping | See all details
Returns: 14 day money back, buyer pays return shipping | Read details

eBay Buyer Protection
Covers your purchase price plus original shipping.

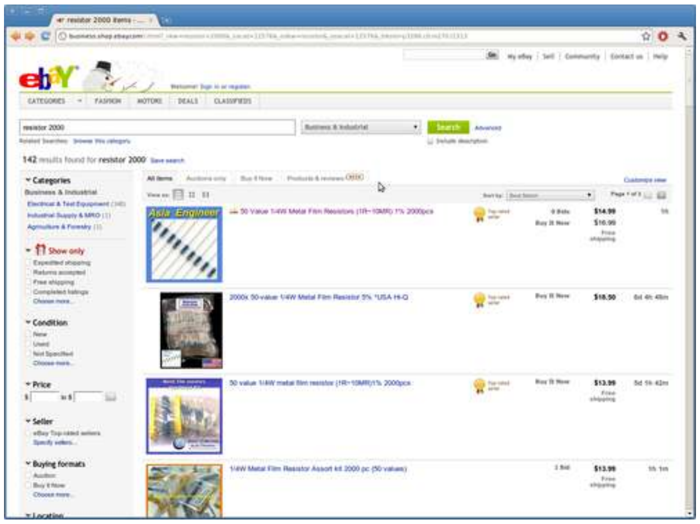
Top-rated seller
99.8% Positive feedback

Other Item Info
Item location: Nanjing, China
Ships to: Worldwide See exclusions
Payments: PayPal See details

Description | Shipping and payments

Seller assumes all responsibility for this listing.

Item specifics
Condition: New: A brand-new, unused, unopened, undamaged item in its original packaging (where packaging is ... Read more



resistor 2000 | Business & Industrial | Search

142 results found for resistor 2000 Save search

50 Value 1/4W Metal Film Resistors (1R-10MR) 1% 2000pcs	0 Bids / Buy It Now	$14.99 / $16.99 Free shipping	1h
2000x 50-value 1/4W Metal Film Resistor 5% *USA Hi-Q	Buy It Now	$18.50	6d 4h 48m
50 value 1/4W metal film resistor (1R~10MR)1% 2000pcs	Buy It Now	$13.99 Free shipping	5d 5h 42m
1/4W Metal Film Resistor Assort kit 2000 pc (50 values)	1 Bid	$13.99 Free shipping	1h 1m

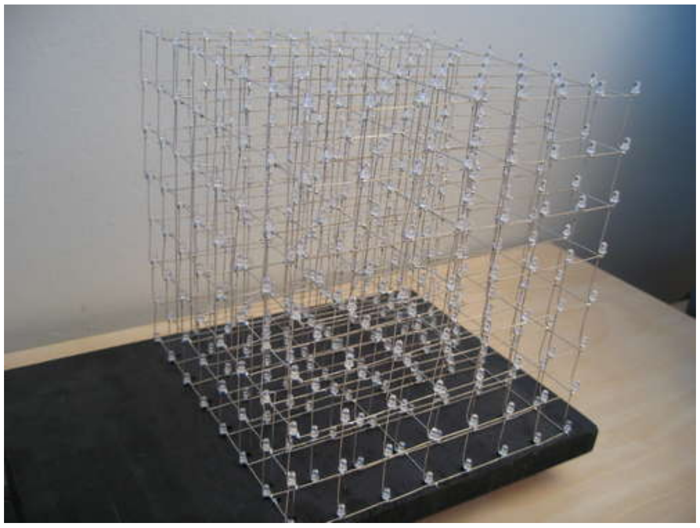
Image Notes
1. 2000 resistors for 17 USD

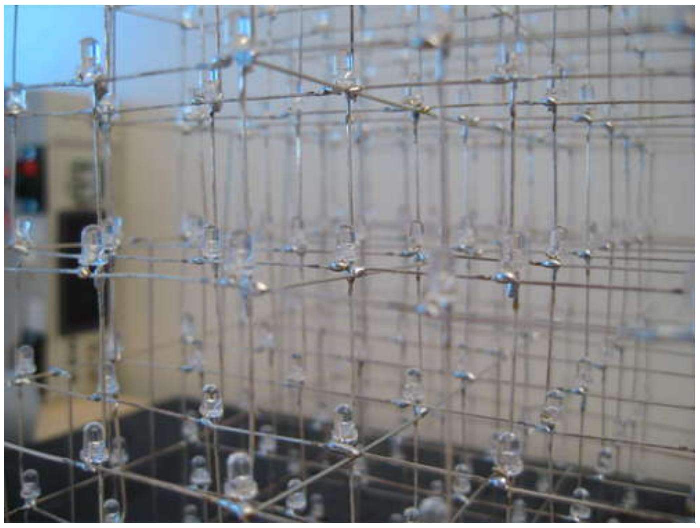
Step 4: What is a LED cube

A LED cube is like a LED screen, but it is special in that it has a third dimension, making it 3D. Think of it as many transparent low resolution displays. In normal displays it is normal to try to stack the pixels as close as possible in order to make it look better, but in a cube one must be able to see trough it, and more spacing between the pixels (actually it's voxels since it is in 3d) is needed. The spacing is a trade-off between how easy the layers behind it is seen, and voxel fidelity.

Since it is a lot more work making a LED cube than a LED display, they are usually low resolution. A LED display of 8x8 pixels is only 64 LEDs, but a LED cube in 8x8x8 is 512 LEDs, an order of magnitude harder to make! This is the reason LED cubes are only made in low resolution.

A LED cube does not have to be symetrical, it is possible to make a 7x8x9, or even oddly shaped ones.





Step 5: How does a LED cube work

This LED cube has 512 LEDs. Obviously, having a dedicated IO port for each LED would be very impractical. You would need a micro controller with 512 IO ports, and run 512 wires through the cube.

Instead, LED cubes rely on an optical phenomenon called persistence of vision (POV).

If you flash a led really fast, the image will stay on your retina for a little while after the led turns off.

By flashing each layer of the cube one after another really really fast, it gives the illusion of a 3d image, when int fact you are looking at a series of 2d images stacked ontop oneanother. This is also called multiplexing.

With this setup, we only need 64 (for the anodes) + 8 (for each layer) IO ports to control the LED cube.

In the video, the process is slowed down enough for you to see it, then it runs faster and faster until the refresh rate is fast enough for the camera to catch the POV effect.

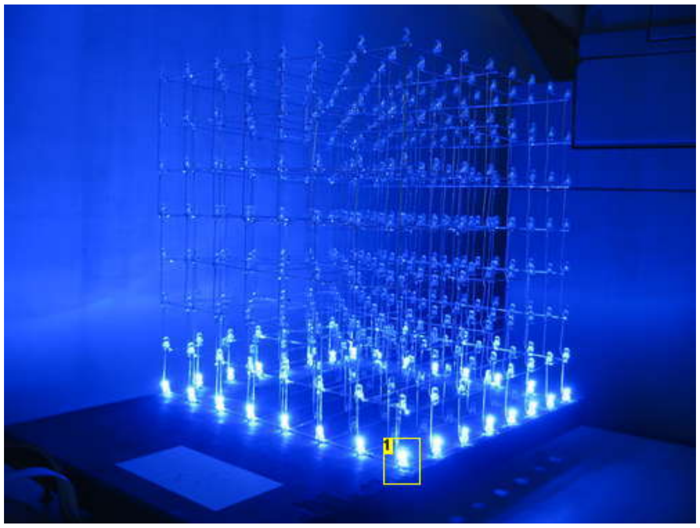


Image Notes
1. We start by flashing the bottom layer. Layer 0.

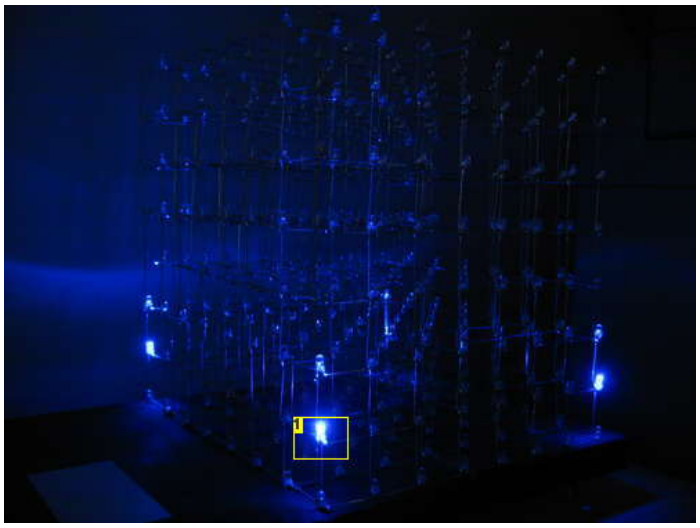


Image Notes
1. Then the second.

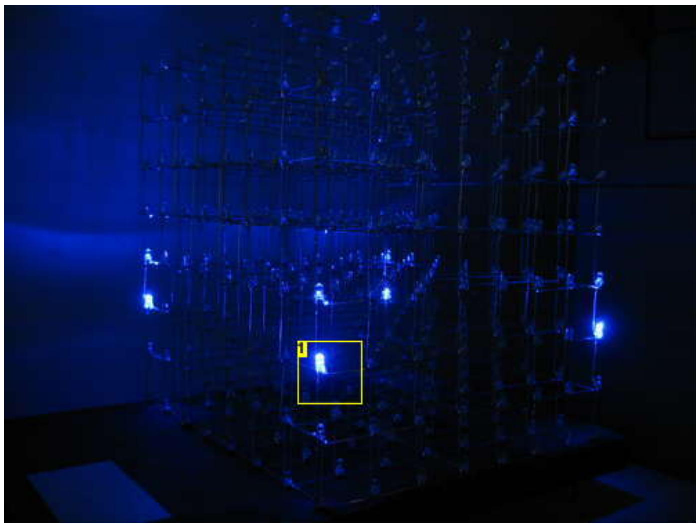


Image Notes
1. And so on..

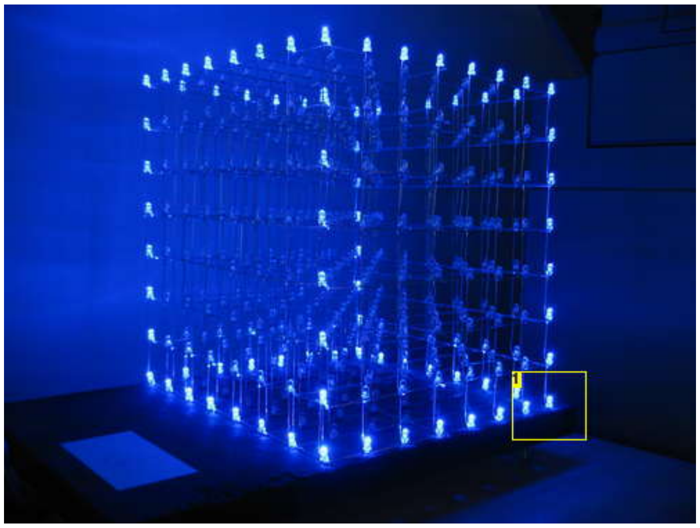


Image Notes
1. Do this fast enough, and your human eyes won't know the difference! Robots may be able to see past the illusion, though.

Step 6: The anatomy of a LED cube

We are going to be talking about anodes, cathodes, columns and layers, so lets take a moment to get familiar with the anatomy of a LED cube.

An LED has two legs. One positive (the anode) and one negative (cathode). In order to light up an LED, you have to run current from the positive to the negative leg. (If i remember correctly the actual flow of electrons is the other way around. But let's stick to the flow of current which is from positive to negative for now).

The LED cube is made up of columns and layers. The cathode legs of every LED in a layer are soldered together. All the anode legs in one column are soldered together.

Each of the 64 columns are connected to the controller board with a separate wire. Each column can be controlled individually. Each of the 8 layers also have a separate wire going to the controller board.

Each of the layers are connected to a transistor that enables the cube to turn on and off the flow of current through each layer.

By only turning on the transistor for one layer, current from the anode columns can only flow through that layer. The transistors for the other layers are off, and the image outputted on the 64 anode wires are only shown on the selected layer.

To display the next layer, simply turn off the transistor for the current layer, change the image on the 64 anode wires to the image for the next layer. Then turn on the transistor for the next layer. Rinse and repeat very very fast.

The layers will be referred to as layers, cathode layers or ground layers.
The columns will be referred to as columns, anode columns or anodes.

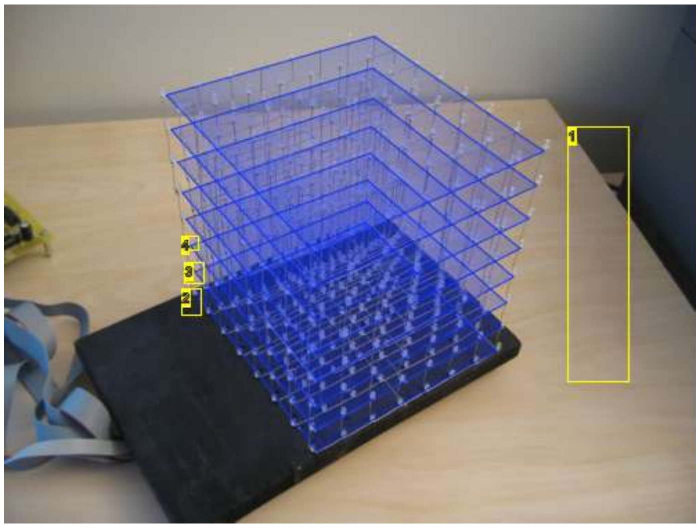


Image Notes
1. 8 layers
2. A 64x64 image is flashed first on layer 0
3. Then another image is flashed on layer 1
4. Wash rinse repeat

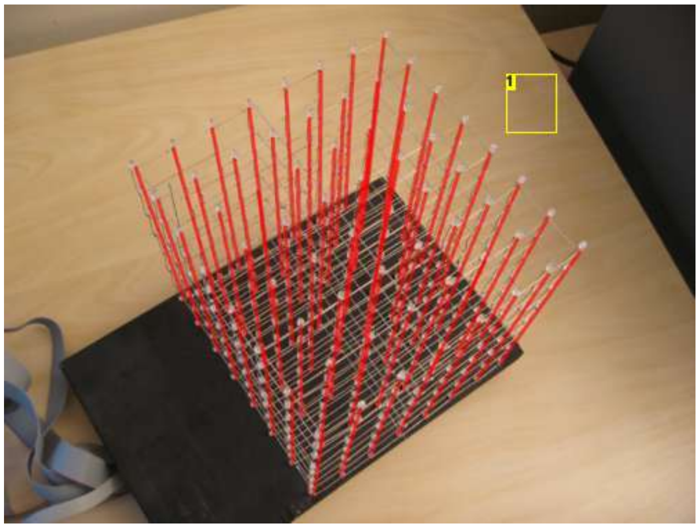


Image Notes
1. 64 columns

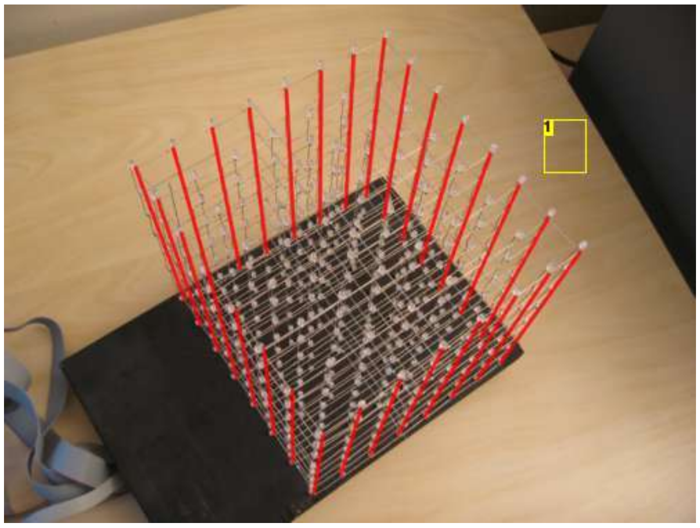


Image Notes
1. Was easier to see when I didn't draw all 64 lines

Step 7: Cube size and IO port requirements

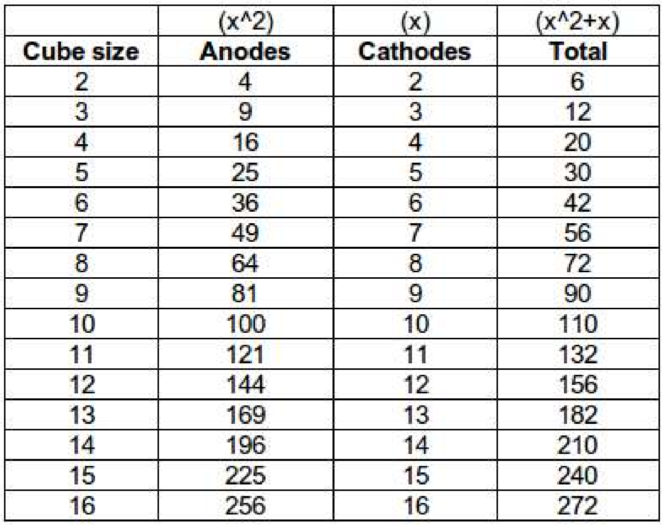
To drive a LED cube, you need two sets of IO ports. One to source all the LED anode columns, and one to sink all the cathode layers.

For the anode side of the cube, you'll need x^2 IO ports, where x^3 is the size of your LED cube. For an 8x8x8 (x=8), you need 64 IO ports to drive the LED anodes. (8x8). You also need 8 IO ports to drive the cathodes.

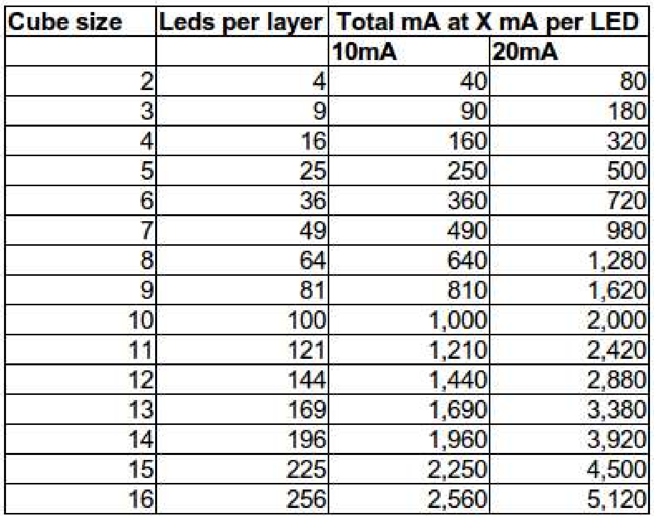
Keep in mind that the number of IO ports will increase exponentially. So will the number of LEDs. You can see a list of IO pin requirement for different cube sizes in table 1.

For a small LED cube, 3x3x3 or 4x4x4, you might get away with connecting the cathode layers directly to a micro controller IO pin. For a larger cube however, the current going through this pin will be too high. For an 8x8x8 LED cube with only 10mA per LED, you need to switch 0.64 Ampere. See table 2 for an overview of power requirements for a LED layer of different sizes. This table shows the current draw with all LEDs on.

If you are planning to build a larger cube than 8x8x8 or running each LED at more than 10-ish mA, remember to take into consideration that your layer transistors must be able to handle that load.



	(x^2)	(x)	(x^2+x)
Cube size	Anodes	Cathodes	Total
2	4	2	6
3	9	3	12
4	16	4	20
5	25	5	30
6	36	6	42
7	49	7	56
8	64	8	72
9	81	9	90
10	100	10	110
11	121	11	132
12	144	12	156
13	169	13	182
14	196	14	210
15	225	15	240
16	256	16	272



Cube size	Leds per layer	Total mA at X mA per LED	
		10mA	20mA
2	4	40	80
3	9	90	180
4	16	160	320
5	25	250	500
6	36	360	720
7	49	490	980
8	64	640	1,280
9	81	810	1,620
10	100	1,000	2,000
11	121	1,210	2,420
12	144	1,440	2,880
13	169	1,690	3,380
14	196	1,960	3,920
15	225	2,250	4,500
16	256	2,560	5,120

Step 8: IO port expansion, more multiplexing

We gathered from the last step that an 8x8x8 LED cube requires 64+8 IO lines to operate. No AVR micro controller with a DIP package (the kind of through hole chip you can easily solder or use in a breadboard, Dual Inline Package) have that many IO lines available.

To get get the required 64 output lines needed for the LED anodes, we will create a simple multiplexer circuit. This circuit will multiplex 11 IO lines into 64 output lines.

The multiplexer is built by using a component called a latch or a flip-flop. We will call them latches from here on.

This multiplexer uses an 8 bit latch IC called 74HC574. This chip has the following pins:

- 8 inputs (D0-7)
- 8 outputs (Q0-7)
- 1 "latch" pin (CP)
- 1 output enable pin (OE)

The job of the latch is to serve as a kind of simple memory. The latch can hold 8 bits of information, and these 8 bits are represented on the output pins. Consider a latch with an LED connected to output Q0. To turn this LED on, apply V+ (1) to input D0, then pull the CP pin low (GND), then high (V+).

When the CP pin changes from low to high, the state of the input D0 is "latched" onto the output Q0, and this output stays in that state regardless of future changes in the status of input D0, until new data is loaded by pulling the CP pin low and high again.
To make a latch array that can remember the on/off state of 64 LEDs we need 8 of these latches. The inputs D0-7 of all the latches are connected together in an 8 bit bus.

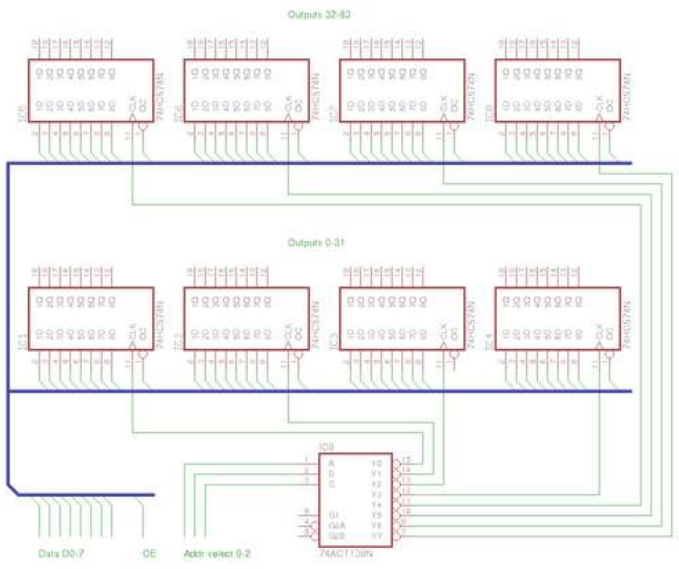
To load the on/off states of all the 64 LEDs we simply do this: Load the data of the first latch onto the bus. pull the CP pin of the first latch low then high. Load the data of the second latch onto the bus. pull the CP pin of the second latch low then high. Load the data of the third latch onto the bus. pull the CP pin of the third latch low then high. Rinse and repeat.

The only problem with this setup is that we need 8 IO lines to control the CP line for each latch. The solution is to use a 74HC138. This IC has 3 input lines and 8 outputs. The input lines are used to control which of the 8 output lines that will be pulled low at any time. The rest will be high. Each out the outputs on the 74HC138 is connected to the CP pin on one of the latches.

The following pseudo-code will load the contents of a buffer array onto the latch array:

```
// PORT A = data bus
// PORT B = address bus (74HC138)
// char buffer[8] holds 64 bits of data for the latch array

PORTB = 0x00; // This pulls CP on latch 1 low.
for (i=0; i < 8; i++)
{

PORTA = buffer[i];
PORTB = i+1;

}
```

The outputs of the 74HC138 are active LOW. That means that the output that is active is pulled LOW. The latch pin (CP) on the latch is a rising edge trigger, meaning that the data is latched when it changes from LOW to HIGH. To trigger the right latch, the 74HC138 needs to stay one step ahead of the counter i. If it had been an active HIGH chip, we could write PORTB = i; You are probably thinking, what happens when the counter reaches 7, that would mean that the output on PORTB is 8 (1000 binary)on the last iteration of the for() loop. Only the first 8 bits of PORT B are connected to the 74HC138. So when port B outputs 8 or 1000 in binary, the 74HC138 reads 000 in binary, thus completing its cycle. (it started at 0). The 74HC138 now outputs the following sequence: 1 2 3 4 5 6 7 0, thus giving a change from LOW to HIGH for the current latch according to counter i.



File Downloads

multiplex_theoretical.sch (21 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'multiplex_theoretical.sch']

Step 9: IO port expansion, alternative solution

There is another solution for providing more output lines. We went with the latch based multiplexer because we had 8 latches available when building the LED cube.

You can also use a serial-in-parallel out shift register to get 64 output lines. 74HC164 is an 8 bit shift register. This chip has two inputs (may also have an output enable pin, but we will ignore this in this example).

- data
- clock

Every time the clock input changes from low to high, the data in Q6 is moved into Q7, Q5 into Q6, Q4 into Q5 and so on. Everything is shifted one position to the right (assuming that Q0 is to the left). The state of the data input line is shifted into Q0.

The way you would normally load data into a chip like this, is to take a byte and bit-shift it into the chip one bit at a time. This uses a lot of CPU cycles. However, we have to use 8 of these chips to get our desired 64 output lines. We simply connect the data input of each shift register to each of the 8 bits on a port on the micro controller. All the clock inputs are connected together and connected to a pin on another IO port.

This setup will use 9 IO lines on the micro controller.

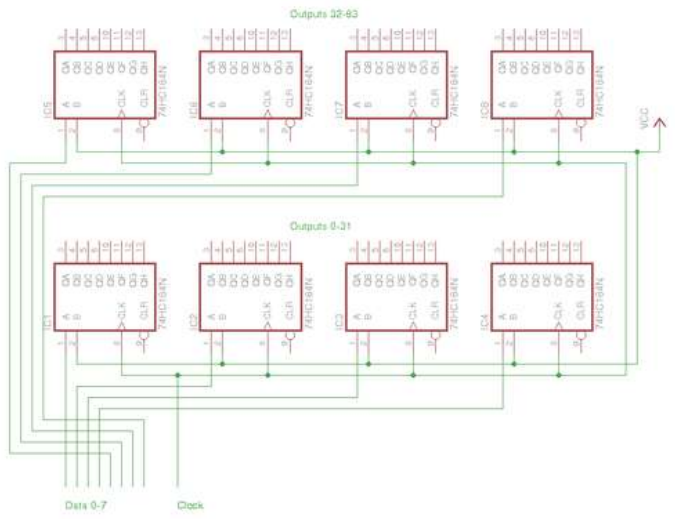
In the previous solution, each byte in our buffer array was placed in it's own latch IC. In this setup each byte will be distributed over all 8 shift registers, with one bit in each.

The following pseudo-code will transfer the contents of a 64 bit buffer array to the shift registers.

```
// PORT A: bit 0 connected to shift register 0's data input, bit 1 to shift register 1 and so on.
// PORT B: bit 0 connected to all the clock inputs
// char buffer[8] holds 64 bits of data

for (i=0; i < 8; i++)
{
PORTB = 0x00; // Pull the clock line low, so we can pull it high later to trigger the shift register
PORTA = buffer[i]; // Load a byte of data onto port A
PORTB = 0x01; // Pull the clock line high to shift data into the shift registers.

}
```

This is perhaps a better solution, but we had to use what we had available when building the cube. For the purposes of this instructable, we will be using a latch based multiplexer for IO port expansion. Feel free to use this solution instead if you understand how they both work.

With this setup, the contents of the buffer will be "rotated" 90 degrees compared to the latch based multiplexer. Wire up your cube accordingly, or simply just turn it 90 degrees to compensate ;)



File Downloads

multiplex_alternative.sch (10 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'multiplex_alternative.sch']

Step 10: Power supply considerations

This step is easy to overlook, as LEDs themselves don't draw that much current. But remember that this circuit will draw 64 times the mA of your LEDs if they are all on. In addition to that, the AVR and the latch ICs also draws current.

To calculate the current draw of your LEDs, connect a led to a 5V power supply with the resistor you intend to use, and measure the current in mA. Multiply this number by 64, and you have the power requirements for the cube itself. Add to that 15-20 mA for the AVR and a couple of mA for each latch IC.

Our first attempt at a power supply was to use a step-down voltage regulator, LM7805, with a 12V wall wart. At over 500mA and 12V input, this chip became extremely hot, and wasn't able to supply the desired current.

We later removed this chip, and soldered a wire from the input to the output pin where the chip used to be.

We now use a regulated computer power supply to get a stable high current 5V supply.

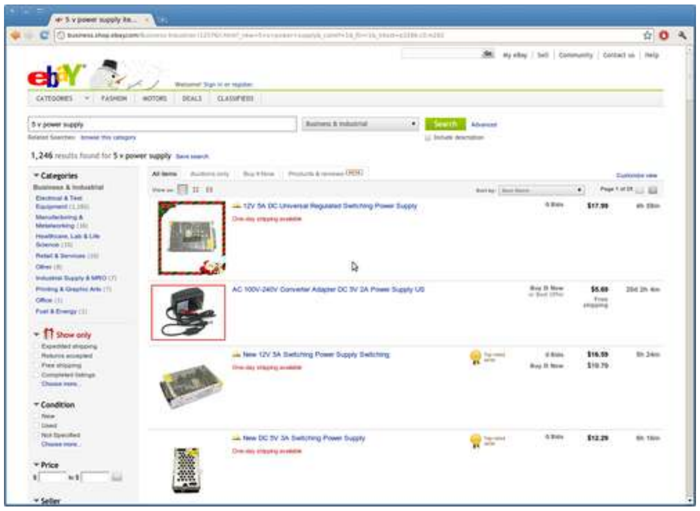
Image Notes
1. Cube drawing almost half an amp at 5 volts.

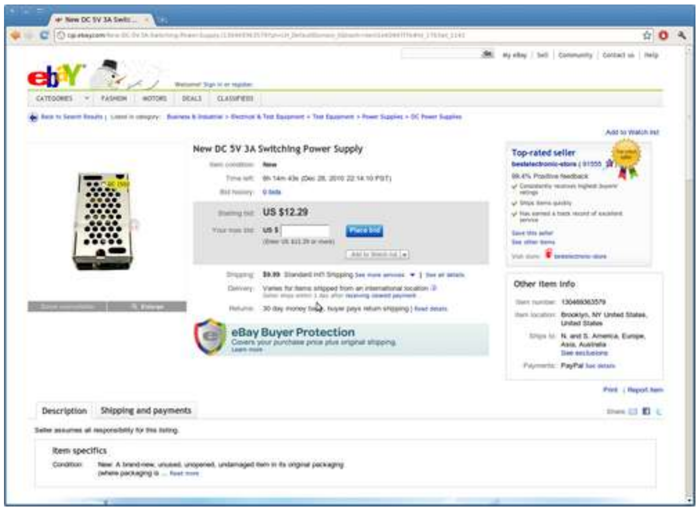
Step 11: Buy a power supply

If you don't have the parts necessary to build a 5V PSU, you can buy one.

eBay is a great place to buy these things.

Search for "5v power supply" and limit the search to "Business & Industrial", and you'll get a lot of suitable power supplies. About 15 bucks will get you a nice PSU.


5 v power supply
Business & Industrial
Search
1,246 results found for 5 v power supply
12V 5A DC Universal Regulated Switching Power Supply
AC 100V-240V Converter Adapter DC 5V 2A Power Supply US
New 12V 5A Switching Power Supply Switching
New DC 5V 3A Switching Power Supply



Step 12: Build a power supply

A couple of years before we built the LED cube, we made our self a nice little lab power supply from an old external SCSI drive. This is what we have been using to power the LED cube.

PC power supplies are nice, because they have regulated 12V and 5V rails with high Ampere ratings.

You can use either a regular AT or ATX power supply or and old external hard drive enclosure.

If you want to use an ATX power supply, you have to connect the green wire on the motherboard connector to ground (black). This will power it up.

External hard drive enclosures are especially nice to use as power supplies. They already have a convenient enclosure. The only thing you have to do is to add external power terminals.

Power supplies have a lot of wires, but the easiest place to get the power you need is through a molex connector. That is the kind of plug you find on hard drives (before the age of S-ATA).

Black is GND Yellow is +12V Red is +5V

Here is an image of our lab PSU. We have 12V output, 5V output with an ampere meter and 5V output without an ampere meter. We use the second 5V output to power an 80mm PC fan to suck or blow fumes away when we solder.

We won't get into any more details of how to make a power supply here. I'm sure you can find another instructable on how to do that.

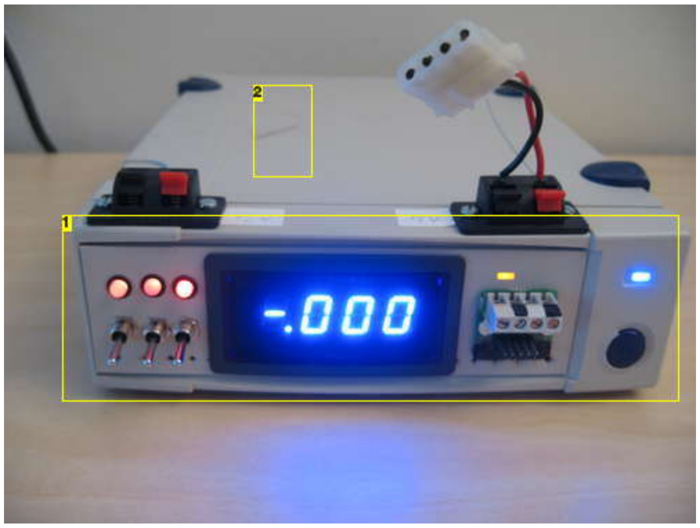


Image Notes
1. Old SCSI disk
2. Inside here is a small powersupply that used to supply the SCSI hard drive that was inside.

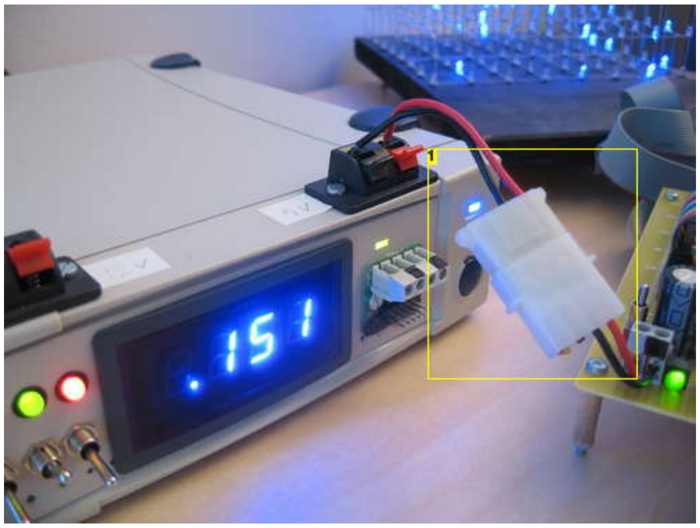
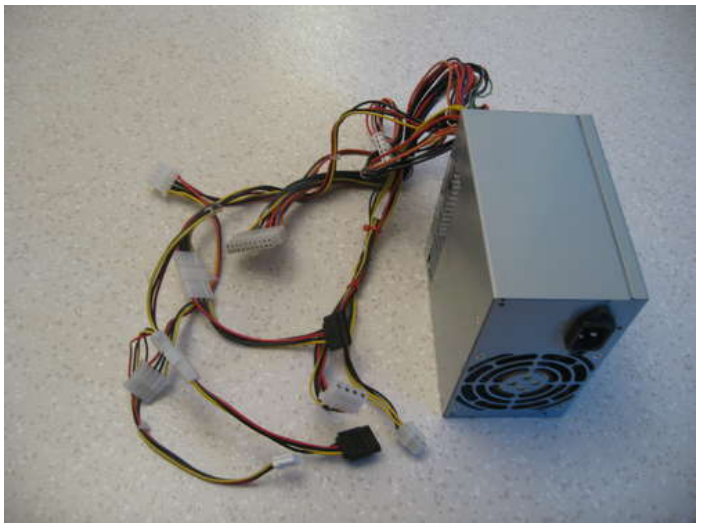


Image Notes
1. Used a Molex connector so we could disconnect the cube easily.



Step 13: Choose your LEDs

There are many things to consider when choosing LEDs.

1)
You want the LED cube to be equally visible from all sides. Therefore we strongly recommend using diffused LEDs. A clear LED will shoot the majority of it's light out the top of the LED. A diffused LED will be more or less equally bright from all sides. Clear LEDs also create another problem. If your cube is made up of clear LEDs. The LEDs will also partially illuminate the LEDs above them, since most of the light is directed upwards. This creates some unwanted ghosting effects.

We actually ordered diffused LEDs from eBay, but got 1000 clear LEDs instead. Shipping them back to China to receive a replacement would have taken too much time, so we decided to used the clear LEDs instead. It works fine, but the cube is a lot brighter when viewed from the top as opposed to the sides.

The LEDs we ordered from eBay were actually described as "Defused LEDs". Maybe we should have taken the hint ;) Defusing is something you do to a bomb when you want to prevent it from blowing up, hehe.

2)
Larger LEDs gives you a bigger and brighter pixel, but since since the cube is 8 layers deep, you want enough room to see all the way through to the furthest level. We went with 3mm LEDs because we wanted the cube to be as "transparent" as possible. Our recommendation is to use 3mm diffused LEDs.

3)
You can buy very cheap lots of 1000 LEDs on eBay. But keep in mind that the quality of the product may be reflected in it's price. We think that there is less chance of LED malfunction if you buy better quality/more expensive LEDs.

4)
Square LEDs would probably look cool to, but then you need to make a soldering template that can accommodate square LEDs. With 3mm round LEDs, all you need is a 3mm drill bit.

5)
Since the cube relies on multiplexing and persistence of vision to create images, each layer is only turned on for 1/8 of the time. This is called a 1/8 duty cycle. To compensate for this, the LEDs have to be bright enough to produce the wanted brightness level at 1/8 duty cycle.

6)
Leg length. The cube design in this instructable uses the legs of the LEDs themselves as the skeleton for the cube. The leg length of the LEDs must be equal to or greater than the distance you want between each LED.

Image Notes
1. So many choices..

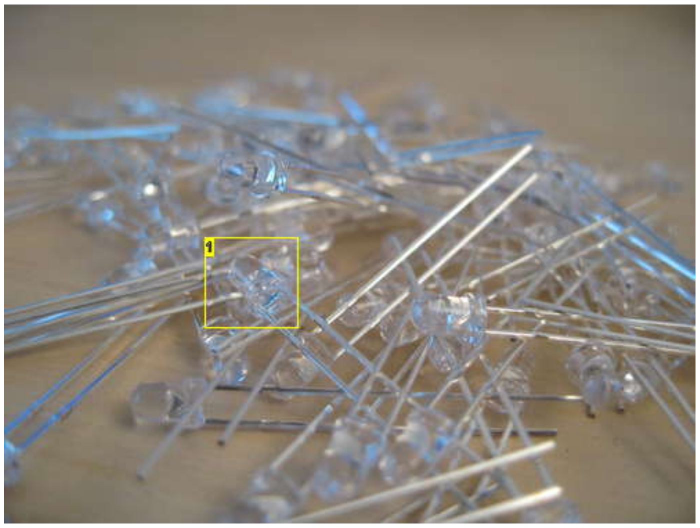


Image Notes
1. These are the ones we ended up using

Image Notes
1. BAD This is not what we ordered! Damn you ebay!

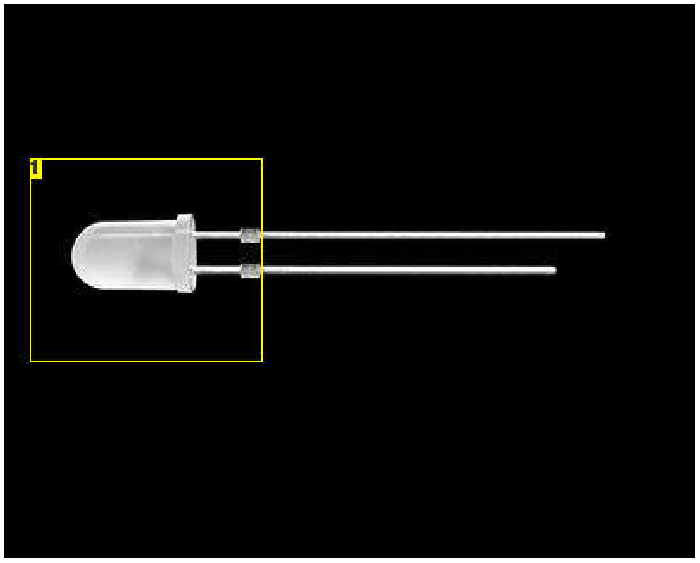


Image Notes
1. GOOD This is what we expected to receive. Diffused LED.

Step 14: Choose your resistors

There are three things to consider when choosing the value of your resistors, the LEDs, the 74HC574 that drive the LEDs, and the transistors used to switch the layers on and off.

1)
If your LEDs came with a data sheet, there should be some ampere ratings in there. Usually, there are two ratings, one mA for continuous load, and mA for burst loads. The LEDs will be running at 1/8 duty cycle, so you can refer to the burst rating.

2)
The 74HC574 also has some maximum ratings. If all the LEDs on one anode column are on, this chip will supply current 8/8 of the time. You have to keep within the specified maximum mA rating for the output pins. If you look in the data sheet, You will find this line: DC Output Source or Sink Current per Output Pin, IO: 25 mA. Also there is a VCC or GND current maximum rating of 50mA. In order not to exceed this, your LEDs can only run at 50/8 mA since the 74HC574 has 8 outputs. This gives you 6.25 mA to work with.

3)
The transistors have to switch on and off 64 x the mA of your LEDs. If your LEDs draw 20mA each, that would mean that you have to switch on and off 1.28 Ampere. The only transistors we had available had a maximum rating of 400mA.

We ended up using resistors of 100 ohms.

While you are waiting for your LED cube parts to arrive in the mail, you can build the guy in the picture below: http://www.instructables.com/id/Resistor-man/

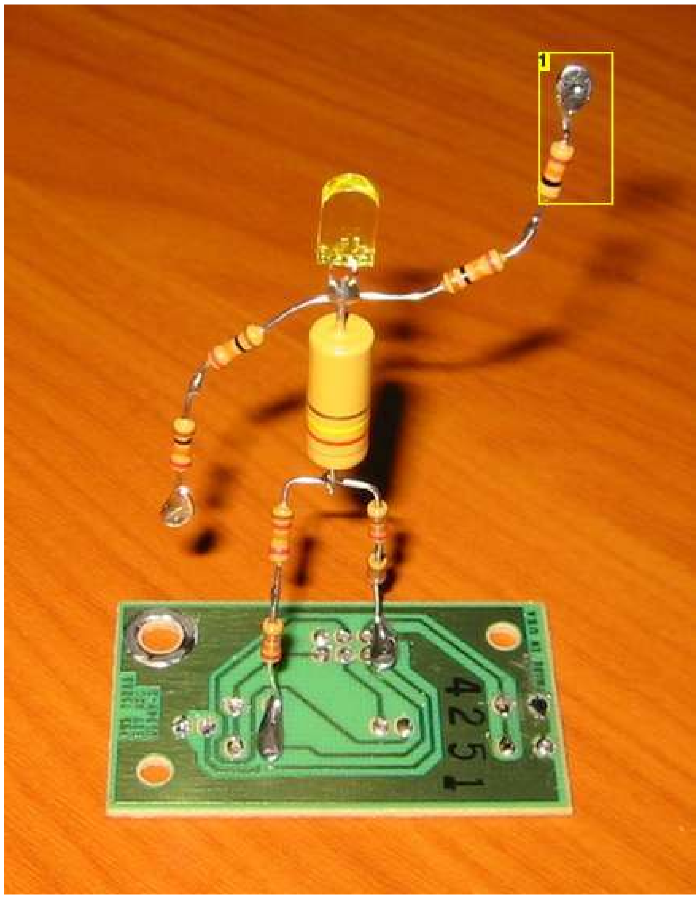


Image Notes
1. Viva la resistance!!

Step 15: Choose the size of your cube

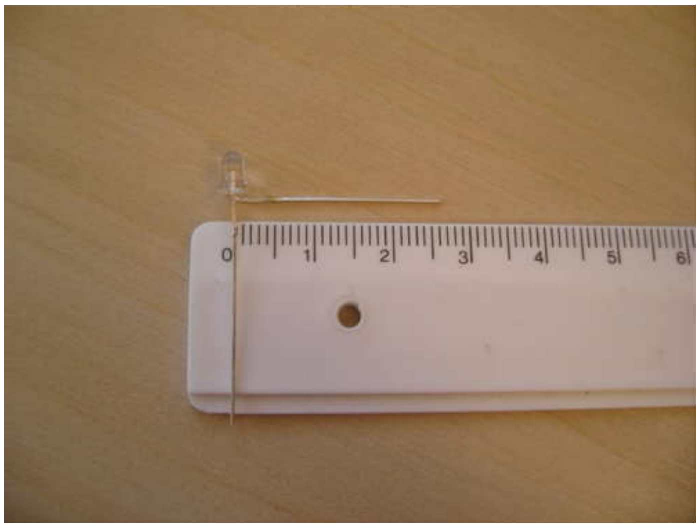
We wanted to make the LED cube using as few components as possible. We had seen some people using metal rods for their designs, but we didn't have any metal rods. Many of the metal rod designs also looked a little crooked.

We figured that the easiest way to build a led cube would be to bend the legs of the LEDs so that the legs become the scaffolding that holds the LEDs in place.

We bent the cathode leg on one of the LEDs and measured it to be 26 mm from the center of the LED. By choosing a LED spacing of 25mm, there would be a 1mm overlap for soldering. (1 inch = 25.4mm)

With a small 3mm LED 25mm between each led gave us plenty of open space inside the cube. Seeing all the way through to the furthest layer wouldn't be a problem. We could have made the cube smaller, but then we would have to cut every single leg, and visibility into the cube would be compromised.

Our recommendation is to use the maximum spacing that your LED can allow. Add 1mm margin for soldering.



Step 16: How to make straight wire

In order to make a nice looking LED Cube, you need some straight steel wire. The only wire we had was on spools, so it had to be straightened.

Our first attempt at this failed horribly. We tried to bend it into a straight wire, but no matter how much we bent, it just wasn't straight enough.

Then we remembered an episode of "How it's made" from the Discovery Channel. The episode was about how they make steel wire. They start out with a spool of really thick wire, then they pull it through smaller and smaller holes. We remembered that the wire was totally straight and symmetrical after being pulled like that.

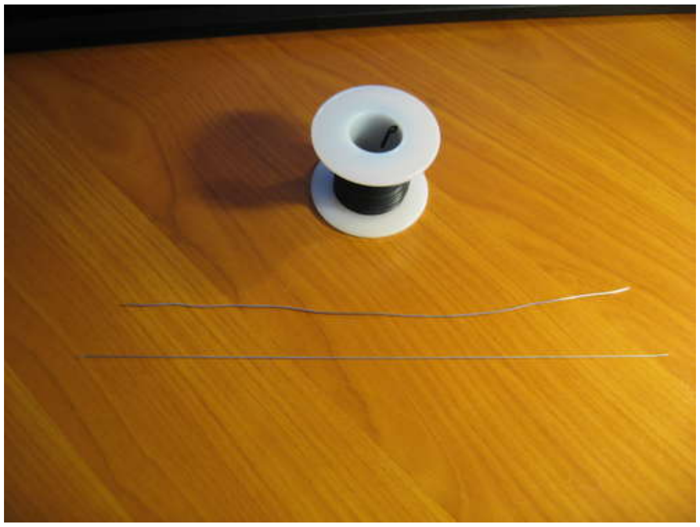
So we figured we should give pulling a try, and it worked! 100% straight metal wire from a spool!

Here is how you do it.

- cut of the length of wire you need from the spool, plus an inch or two.
- Remove the insulation, if any.
- Get a firm grip of each end of the wire with two pairs of pliers
- Pull hard!
- You will feel the wire stretch a little bit.

You only need to stretch it a couple of millimeters to make it nice and straight.

If you have a vice, you can secure one end in the vice and use one pair of pliers. This would probably be a lot easier, but we don't own a vice.



Step 17: Practice in small scale

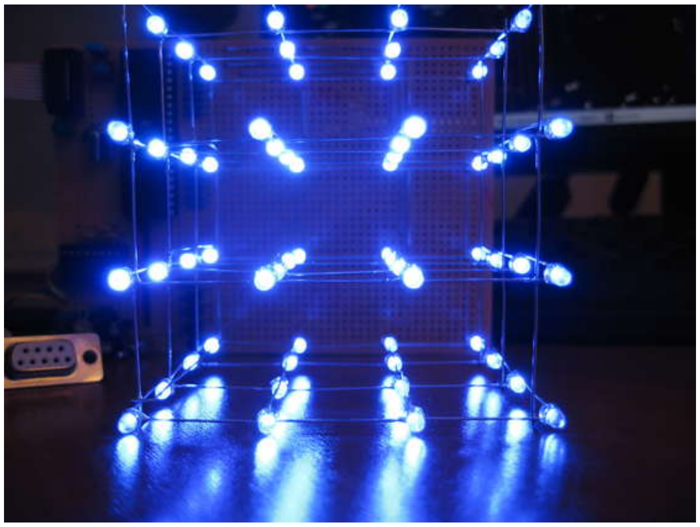
Whenever Myth Busters are testing a complex myth, they start by some small scale experiments.

We recommend that you do the same thing.

Before we built the 8x8x8 LED cube, we started by making a smaller version of it, 4x4x4. By making the 4x4x4 version first, you can perfect your cube soldering technique before starting on the big one.

Check out our 4x4x4 LED cube instructable for instructions on building a smaller "prototype".

http://www.instructables.com/id/LED-Cube-4x4x4/



Step 18: Build the cube: create a jig

In order to make a nice looking LED cube, it is important that it is completely symmetrical, that the space between each LED is identical, and that each LED points the same way. The easiest way to accomplish this is to create a temporary soldering jig/template.

1)
Find a piece of wood or plastic that is larger than the size of your cube.

2)
Find a drill bit that makes a hole that fits a LED snugly in place. You don't want it to be to tight, as that would make it difficult to remove the soldered layer from the jig without bending it. If the holes are too big, some of the LEDs might come out crooked.

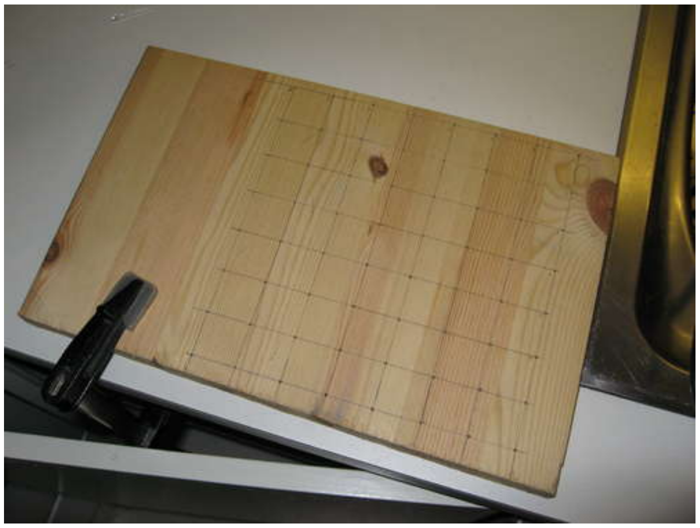
3)
Use a ruler and an angle iron to draw up a grid of 8 by 8 lines intersecting at 64 points, using the LED spacing determined in a previous step.

4)
Use a sharp pointy object to make indentions at each intersection. These indentions will prevent the drill from sliding sideways when you start drilling.

5)
Drill out all the holes.

6)
Take an LED and try every hole for size. If the hole is too snug, carefully drill it again until the LED fits snugly and can be pulled out without much resistance.

7)
Somewhere near the middle of one of the sides, draw a small mark or arrow. A steel wire will be soldered in here in every layer to give the cube some extra stiffening.



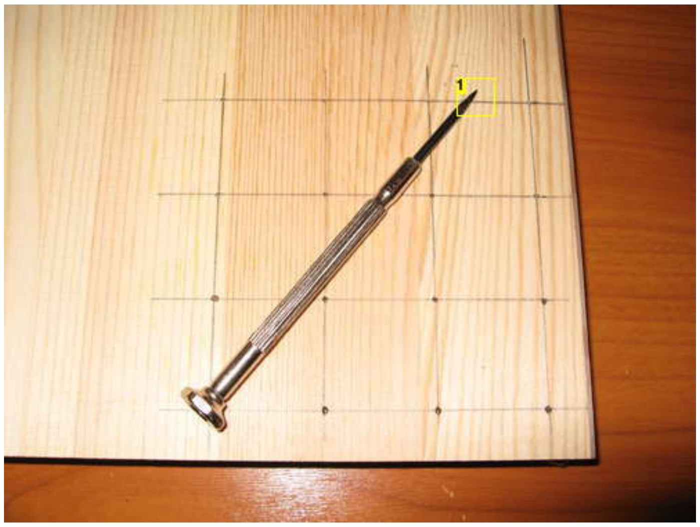


Image Notes
1. If you make a small indentation before drilling, the drill won't slide sideways.

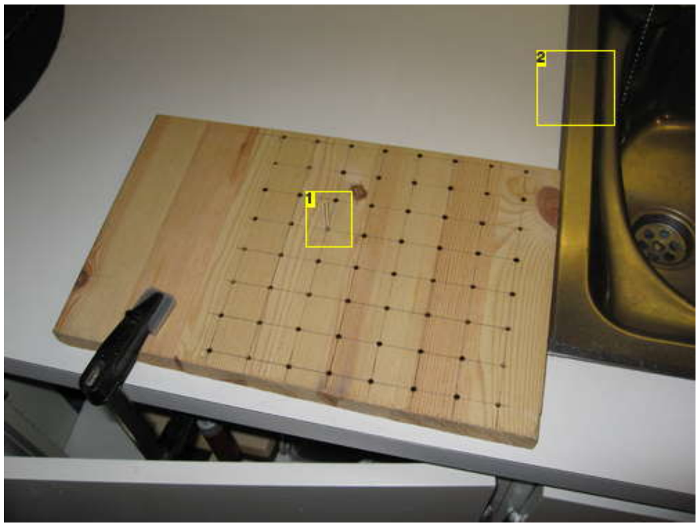


Image Notes
1. All done. We used this LED to test all the holes.
2. Everything but the kitchen sink? We sort of used the kitchen sink to hold the jig in place ;)

Step 19: Build the cube: soldering advice

You are going to be soldering VERY close to the LED body, and you are probably going to be using really cheap LEDs from eBay. LEDs don't like heat, cheap LEDs probably more so than others. This means that you have to take some precautions in order to avoid broken LEDs.

Soldering iron hygiene
First of all, you need to keep your soldering iron nice and clean. That means wiping it on the sponge every time you use it. The tip of your soldering iron should be clean and shiny. Whenever the you see the tip becoming dirty with flux or oxidizing, that means loosing it's shinyness, you should clean it. Even if you are in the middle of soldering. Having a clean soldering tip makes it A LOT easier to transfer heat to the soldering target.

Soldering speed
When soldering so close to the LED body, you need to get in and out quickly. Wipe your iron clean. Apply a tiny amount of solder to the tip. Touch the part you want to solder with the side of your iron where you just put a little solder. Let the target heat up for 0.5-1 seconds, then touch the other side of the target you are soldering with the solder. You only need to apply a little bit. Only the solder that is touching the metal of both wires will make a difference. A big blob of solder will not make the solder joint any stronger. Remove the soldering iron immediately after applying the solder.

Mistakes and cool down
If you make a mistake, for example if the wires move before the solder hardens or you don't apply enough solder. Do not try again right away. At this point the LED is already very hot, and applying more heat with the soldering iron will only make it hotter. Continue with the next LED and let it cool down for a minute, or blow on it to remove some heat.

Solder
We recommend using a thin solder for soldering the LEDs. This gives you a lot more control, and enable you to make nice looking solder joints without large blobs of solder. We used a 0.5 mm gauge solder. Don't use solder without flux. If your solder is very old and the flux isn't cleaning the target properly, get newer solder. We haven't experienced this, but we have heard that it can happen.

Are we paranoid?
When building the 8x8x8 LED Cube, we tested each and every LED before using it in the cube. We also tested every LED after we finished soldering a layer. Some of the LEDs didn't work after being soldered in place. We considered these things before making a single solder joint. Even with careful soldering, some LEDs were damaged. The last thing you want is a broken LED near the center of the cube when it is finished. The first and second layer from the outside can be fixed afterwards, but any further in than that, and you'll need endoscopic surgical tools ;)

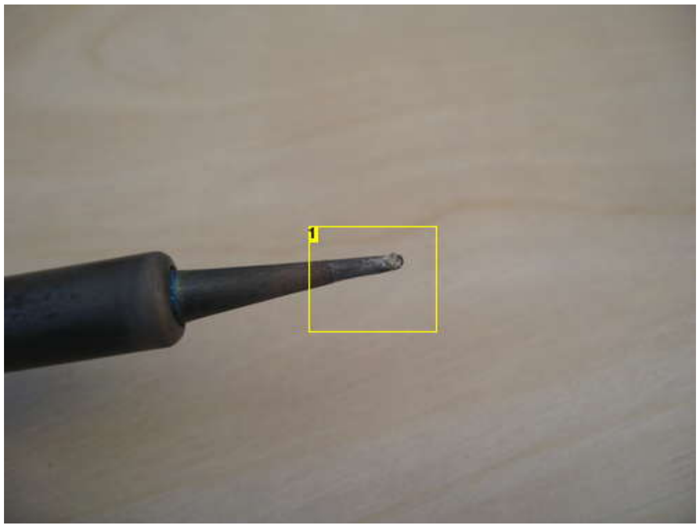


Image Notes
1. If the tip of your soldering iron looks like this, it is time to clean it!

Image Notes
1. This little gadget is great for cleaning your soldering iron

Step 20: Build the cube: test the LEDs

We got our LEDs from eBay, really cheap!

We tested some of the LED before we started soldering, and randomly stumbled on a LED that was a lot dimmer than the rest. So we decided to test every LED before using it. We found a couple of dead LEDs and some that were dimmer than the rest.

It would be very bad to have a dim LED inside your finished LED cube, so spend the time to test the LEDs before soldering! This might be less of a problem if you are using LEDs that are more expensive, but we found it worth while to test our LEDs.

Get out your breadboard, connect a power supply and a resistor, then pop the LEDs in one at a time. You might also want to have another LED with its own resistor permanently on the breadboard while testing. This makes it easier to spot differences in brightness.

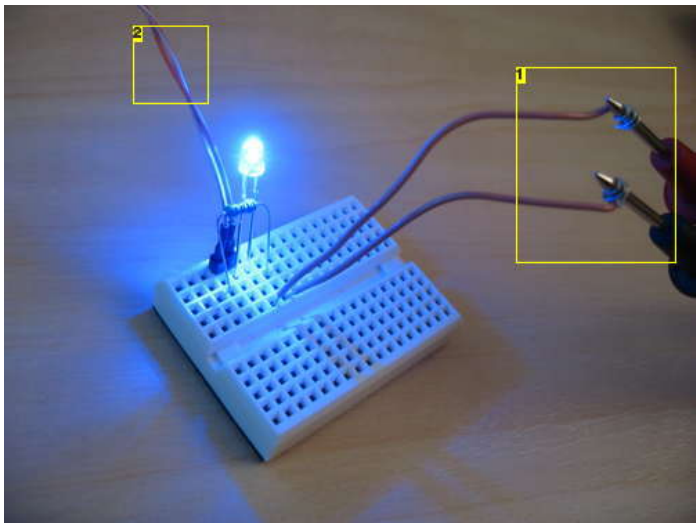


Image Notes
1. Multimeter connected in series to measure mA.
2. 5 volts from power supply

Step 21: Build the cube: solder a layer

Each layer is made up of 8 columns of LEDs held together by the legs of each LED. At the top of each layer each LED is rotated 90 degrees clockwise, so that the leg connects with the top LED of the next column. On the column to the right this leg will stick out of the side of the layer. We leave this in place and use it to connect ground when testing all the LEDs in a later step.

1) Prepare 64 LEDs
Bend the cathode leg of each LED 90 degrees. Make sure the legs are bent in the same direction on all the LEDs. Looking at the LED sitting in a hole in the template with the notch to the right, we bent the leg upwards.

2) Start with the row at the top
Start by placing the top right LED in the template. Then place the one to the left, positioning it so that it's cathode leg is touching the cathode leg of the previous LED. Rinse and repeat until you reach the left LED. Solder all the joints.

3) Solder all 8 columns
If you are right handed, we recommend you start with the column to the left. That way your hand can rest on the wooden template when you solder. You will need a steady hand when soldering freehand like this. Start by placing the LED second from the top, aligning it so it's leg touches the solder joint from the previous step. Then place the LED below that so that the cathode leg touches the LED above. Repeat until you reach the bottom. Solder all the joints.

4) Add braces
You now have a layer that looks like a comb. At this point the whole thing is very flimsy, and you will need to add some support. We used one bracing near the bottom and one near the middle. Take a straight peace of wire, roughly align it where you want it and solder one end to the layer. Fine tune the alignment and solder the other end in place. Now, make solder joints to the remaining 6 columns. Do this for both braces.

5) Test all the LEDs
This is covered in the next step. Just mentioning here so you don't remove the layer just yet.

6) Remove the layer
The first layer of your LED cube is all done, now all you have to do is remove it from the template. Depending on the size of your holes, some LEDs might have more resistance when you try to pull it out. Simply grabbing both ends of the layer and pulling would probably break the whole thing if a couple of the LEDs are stuck.

Start by lifting every single LED a couple of millimeters. Just enough to feel that there isn't any resistance. When all the LEDs are freed from their holes, try lifting it carefully. If it is still stuck, stop and pull the stuck LEDs out.

Repeat 8 times!

Note on images:
If you are having trouble seeing the detail in any of our pictures, you can views the full resolution by clicking on the little i icon in the top left corner of every image. All our close up pictures are taken with a mini tripod and should have excellent macro focus. On the image page, choose the original size from the "Available sizes" menu on the left hand side.

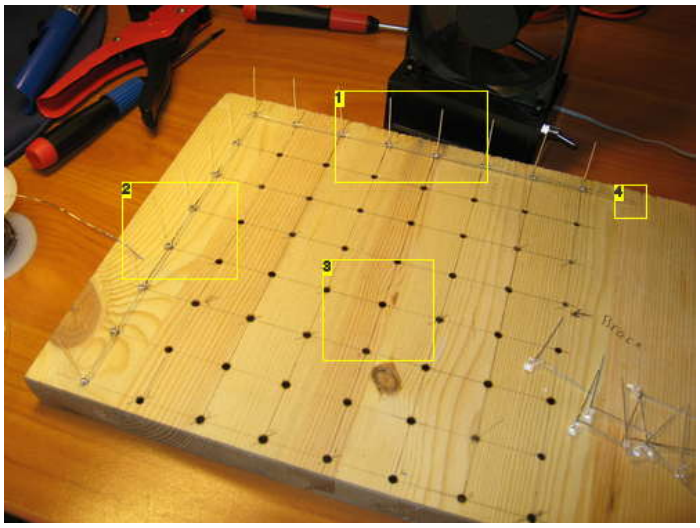


Image Notes
1. Start with this row
2. Then do this column
3. And then the rest..
4. Don't remove the leg that sticks out to the side. It is convenient to connect ground to it when testing the LEDs.

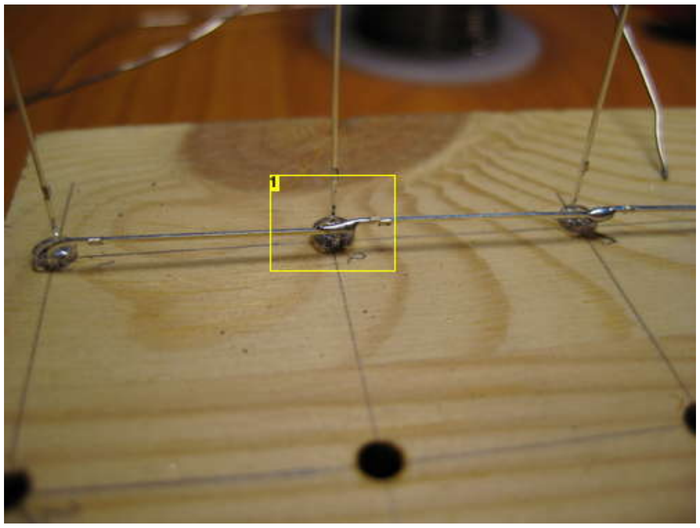


Image Notes
1. About 1mm overlap. Perfect!

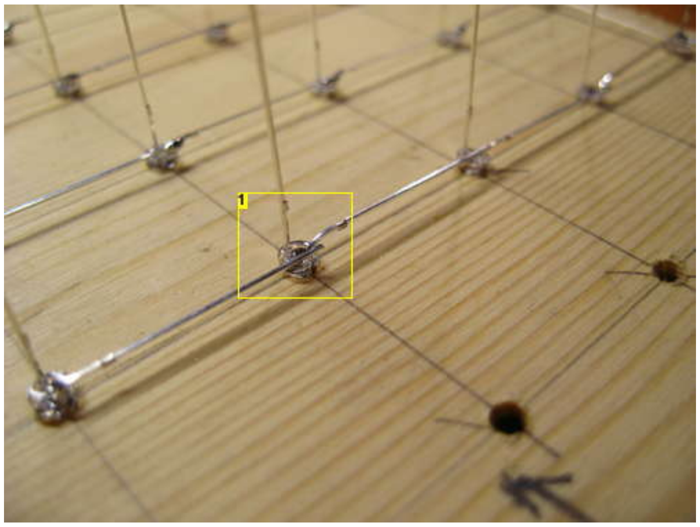


Image Notes
1. LED ready to be soldered. Look how nicely they line up.

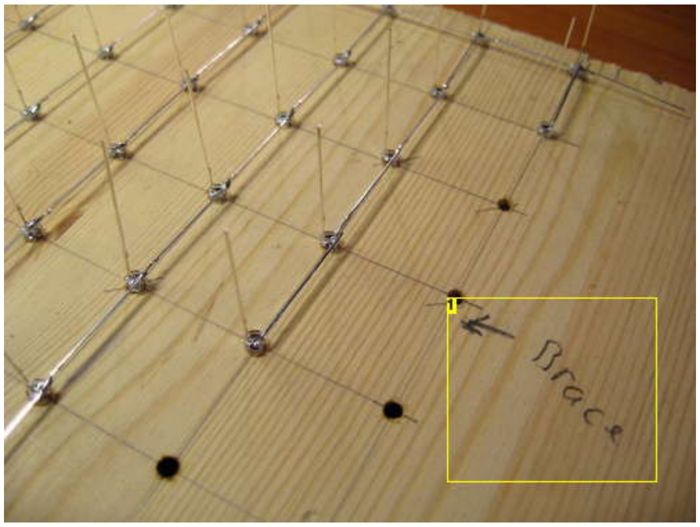
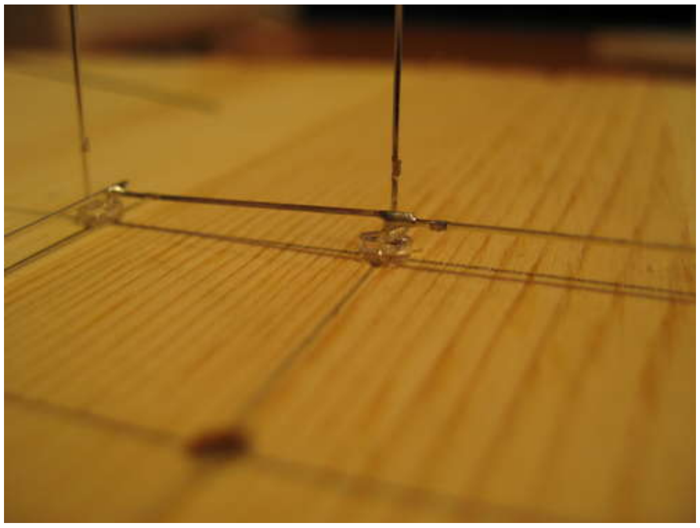
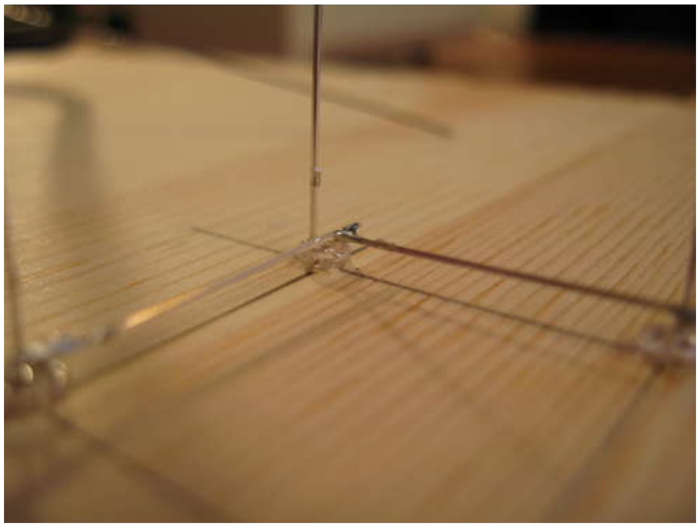


Image Notes
1. We marked off where we wanted to have the midway bracing, so we didn't accidentally put it in different locations in each layer :p





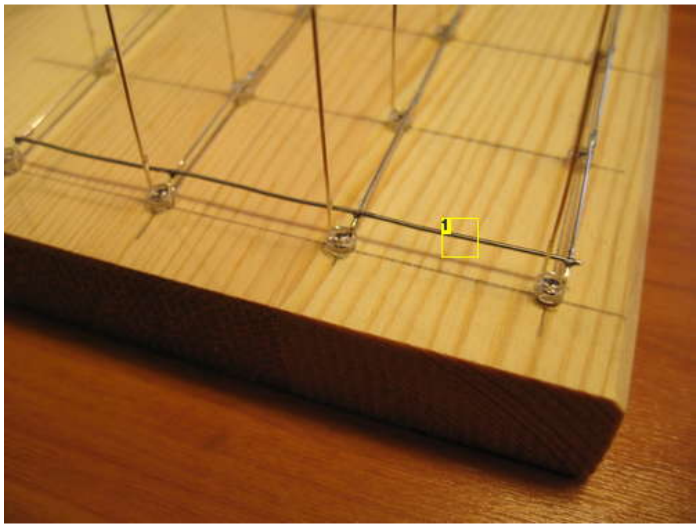


Image Notes
1. Brace

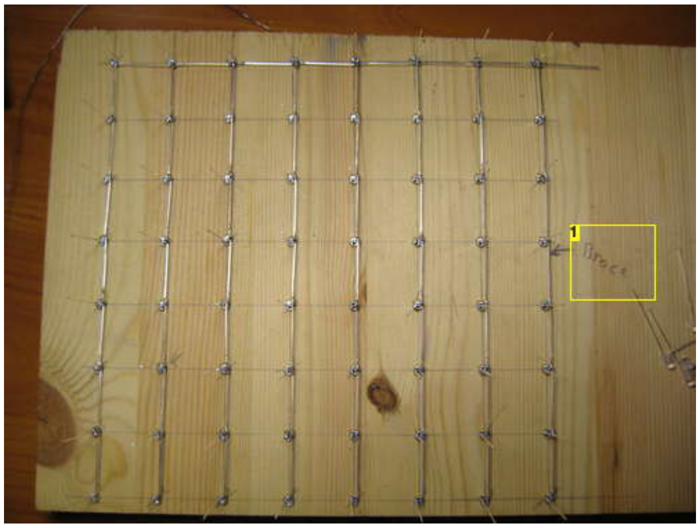


Image Notes
1. Almost done, just need the braces.

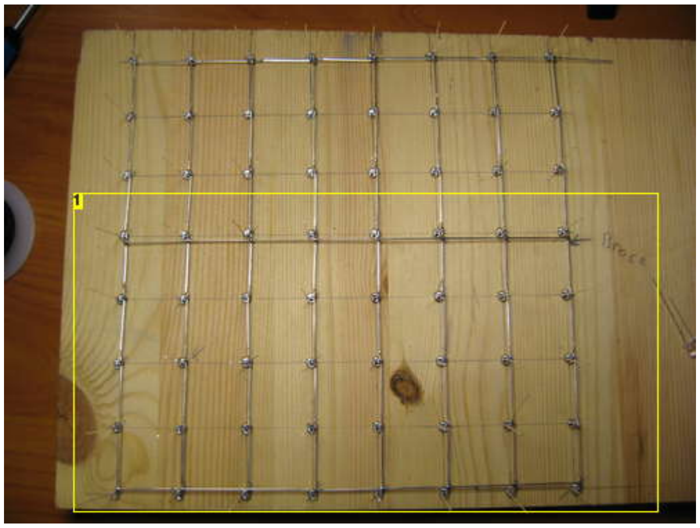
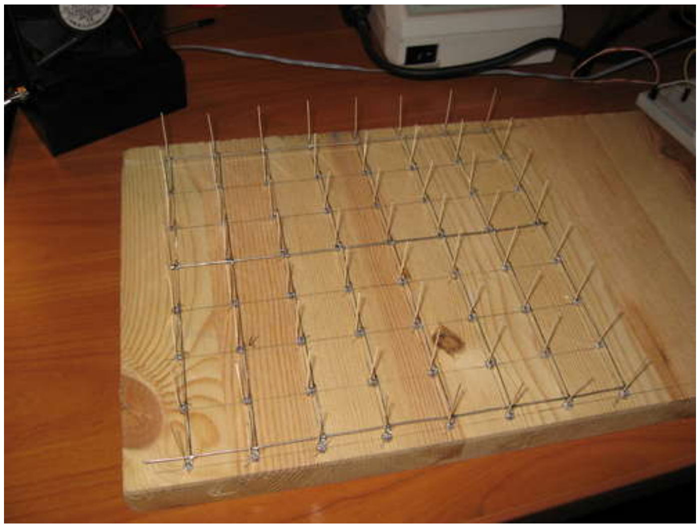
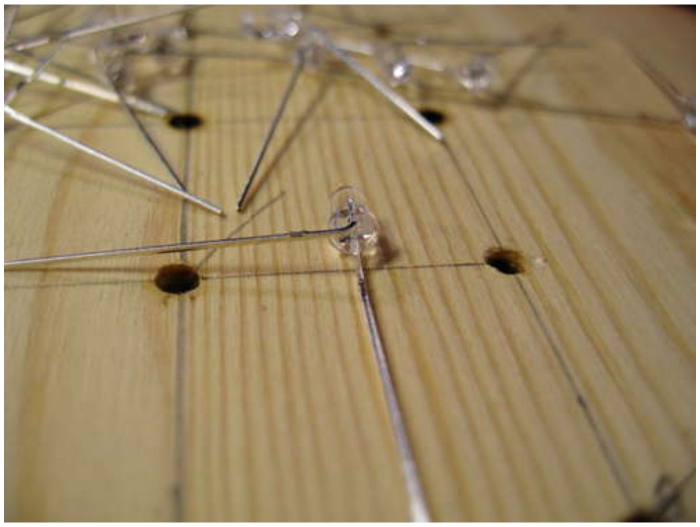
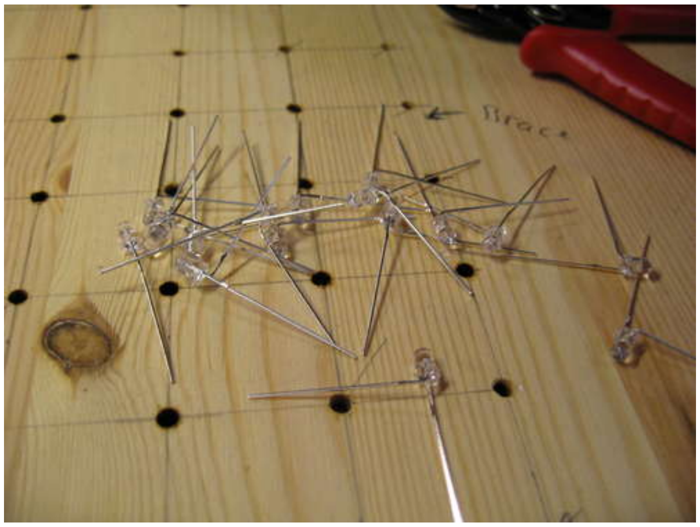


Image Notes
1. All done.







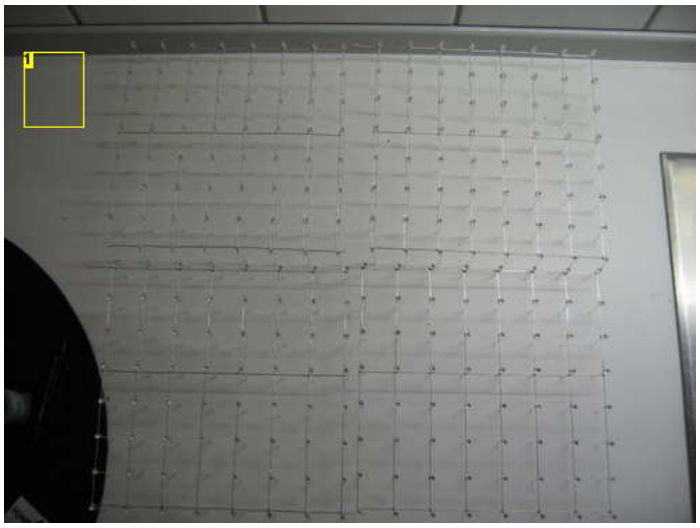


Image Notes
1. 4 down 4 to go!

Step 22: Build the cube: test the layer

Soldering that close to the body of the LED can damage the electronics inside. We strongly recommend that you test all LEDs before proceeding.

Connect ground to the tab you left sticking out at the upper right corner. Connect a wire to 5V through a resistor. Use any resistor that lights the LED up and doesn't exceed its max mA rating at 5V. 470 Ohm would probably work just fine.

Take the wire and tap it against all 64 anode legs that are sticking up from your template. If a LED doesn't flash when you tap it, that means that something is wrong.

1) Your soldering isn't conducting current.
2) The LED was overheated and is broken.
3) You didn't make a proper connection between the test wire and the led. (try again).

If everything checks out, pull the layer from the cube and start soldering the next one.

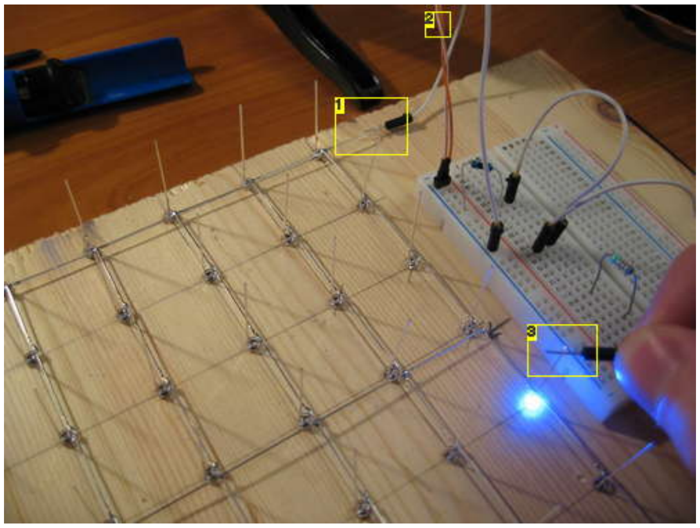


Image Notes
1. Ground connected to the layer
2. 5v from power supply
3. 5 volts via resistor.

Step 23: Build the cube: straigthen the pins

In our opinion, a LED cube is a piece of art and should be perfectly symmetrical and straight. If you look at the LEDs in your template from the side, they are probably bent in some direction.

You want all the legs to point straight up, at a 90 degree angle from the template.

While looking at the template from the side, straighten all the legs. Then rotate the template 90 degrees, to view it from the other side, then do the same process.

You now have a perfect layer that is ready to be removed from the template.

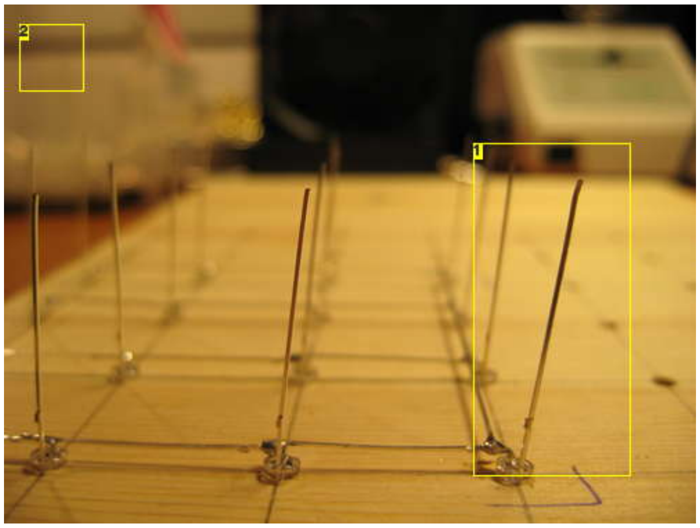


Image Notes
1. This isn't going to be a very nice LED cube!
2. We use a 4x4x4 cube here to demonstrate.

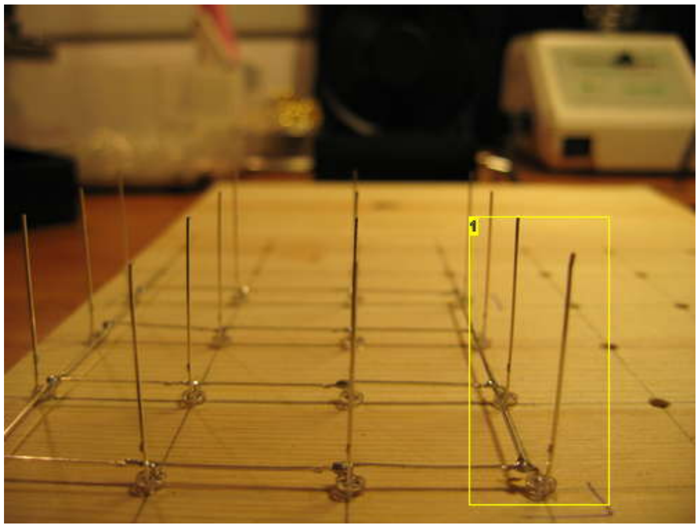


Image Notes
1. This is better

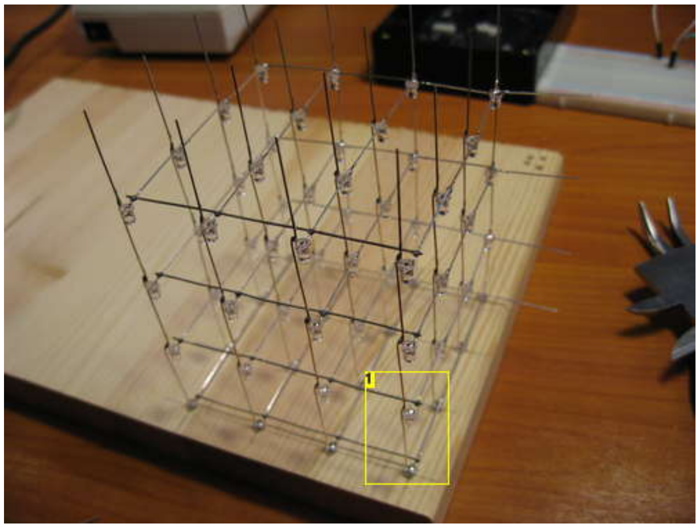
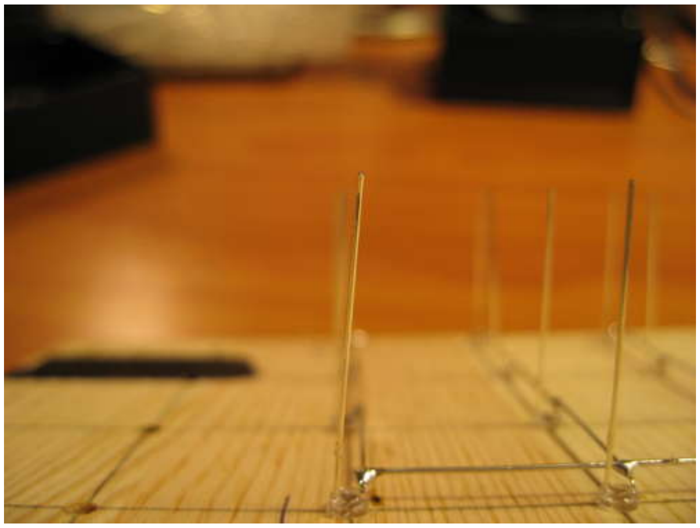
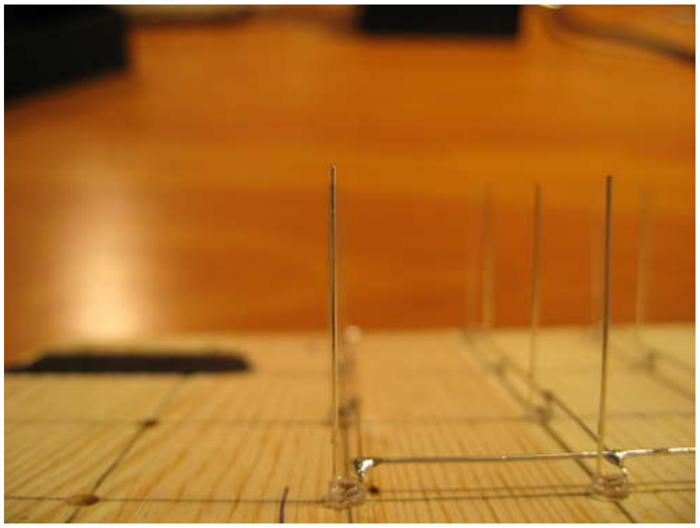


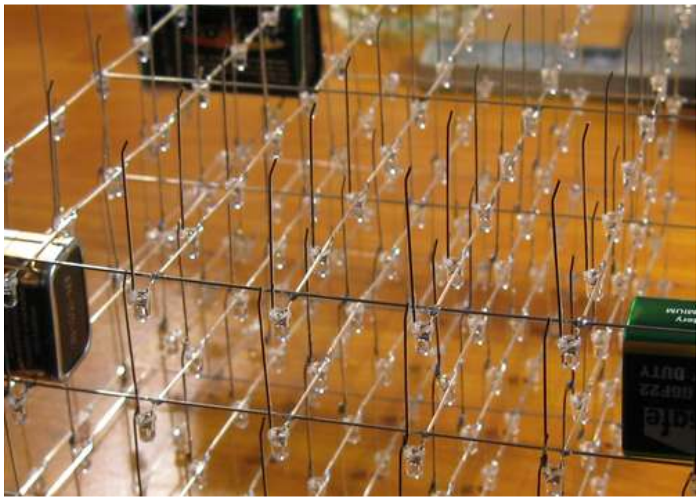
Image Notes
1. Pin straightening paid off.. see how straight the cube is

Step 24: Build the cube: bend the pins

In the LED cube columns, we want each LED to sit centered precisely above the LEDs below. The legs on the LEDs come out from the LED body half a millimeter or so from the edge. To make a solder joint, we have to bend the anode leg so that it touches the anode leg on the LED below.

Make a bend in the anode leg towards the cathode leg approximately 3mm from the end of the leg. This is enough for the leg to bend around the LED below and make contact with it's anode leg.



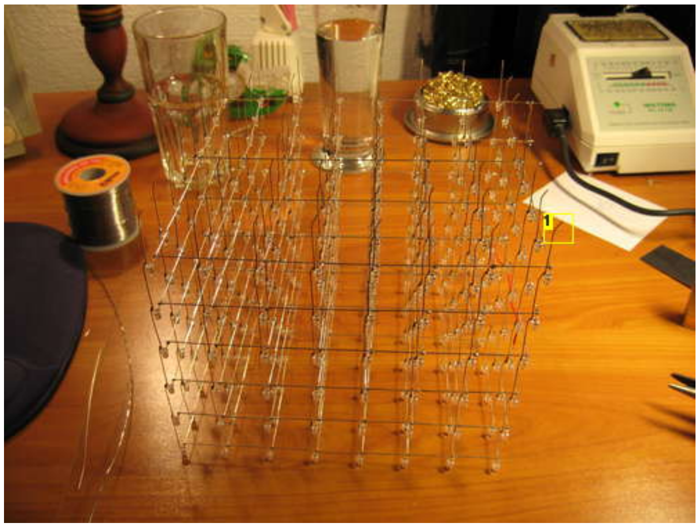


Image Notes
1. Pins are bent in order to make contact with the next LED

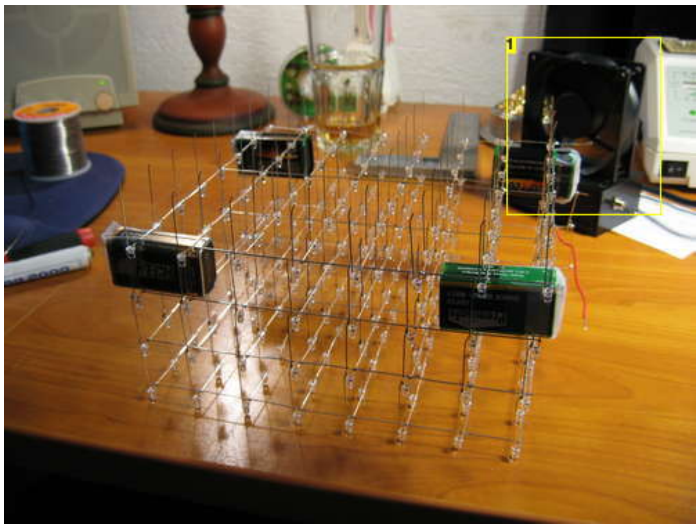


Image Notes

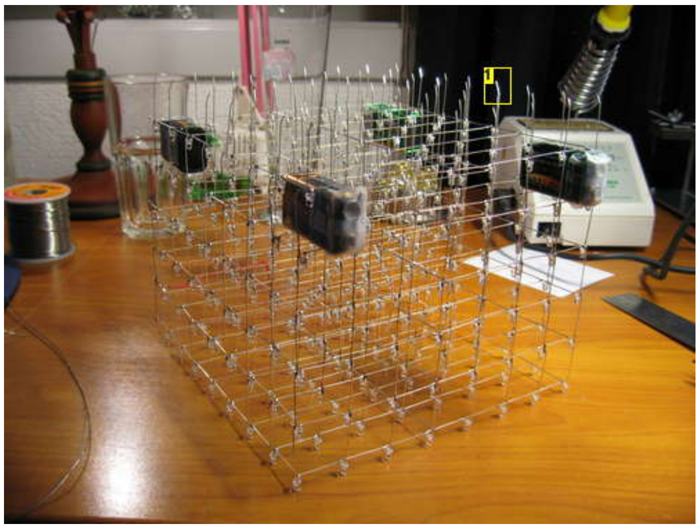


Image Notes

1. Fan to blow the fumes away from my face.

1. All the pins are bent and ready to receive the next layer.

Step 25: Build the cube: solder the layers together

Now comes the tricky part, soldering it all together!

The first two layers can be quite flimsy before they are soldered together. You may want to put the first layer back in the template to give it some stability.

In order to avoid total disaster, you will need something to hold the layer in place before it is soldered in place. Luckily, the width of a 9V battery is pretty close to 25 mm. Probably closer to 25.5-26mm, but that's OK.

Warning: The 9 volts from a 9V battery can easily overload the LEDs if the contacts on the battery comes in contact with the legs of the LEDs. We taped over the battery poles to avoid accidentally ruining the LEDs we were soldering.

We had plenty of 9V batteries lying around, so we used them as temporary supports.
Start by placing a 9V battery in each corner. Make sure everything is aligned perfectly, then solder the corner LEDs.

Now solder all the LEDs around the edge of the cube, moving the 9V batteries along as you go around. This will ensure that the layers are soldered perfectly parallel to each other.
Now move a 9V battery to the middle of the cube. Just slide it in from one of the sides. Solder a couple of the LEDs in the middle.

The whole thing should be pretty stable at this point, and you can continue soldering the rest of the LEDs without using the 9V batteries for support.

However, if it looks like some of the LEDs are sagging a little bit, slide in a 9V battery to lift them up!

When you have soldered all the columns, it is time to test the LEDs again. Remember that tab sticking out from the upper right corner of the layer, that we told you not to remove yet? Now it's time to use it. Take a piece of wire and solder the tab of the bottom layer to the tab of the layer you just soldered in place.

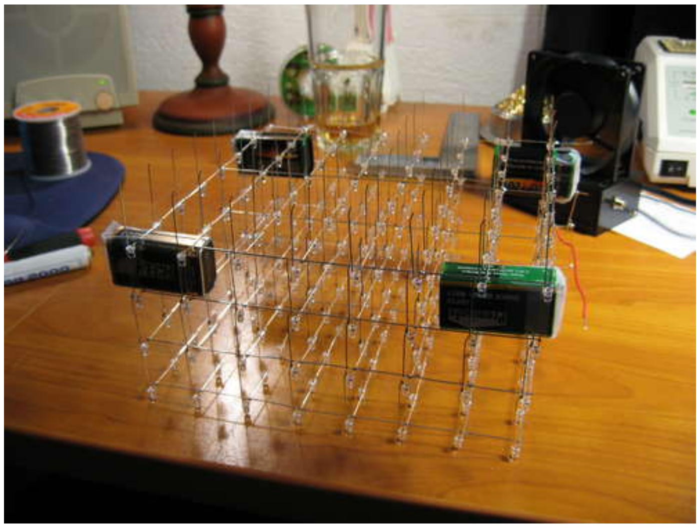
Connect ground to the the ground tab.

Test each led using the same setup as you used when testing the individual layers. Since the ground layers have been connected by the test tabs, and all the anodes in each columns are connected together, all LEDs in a column should light up when you apply voltage to the top one. If the LEDs below it does not light up, it probably means that you forgot a solder joint! It is A LOT better to figure this out at this point, rather than when all the layers are soldered together. The center of the cube is virtually impossible to get to with a soldering iron.

You now have 2/8 of your LED cube soldered together! Yay!

For the next 6 layers, use the exact same process, but spend even more time aligning the corner LEDs before soldering them. Look at the cube from above, and make sure that all the corner LEDs are on a straight line when looking at them from above.

Rinse and repeat!



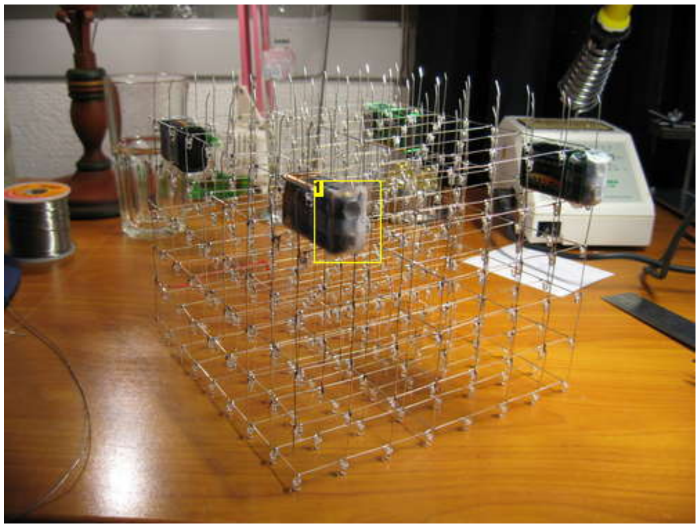


Image Notes
1. We taped over the battery terminals to avoid any disasters!

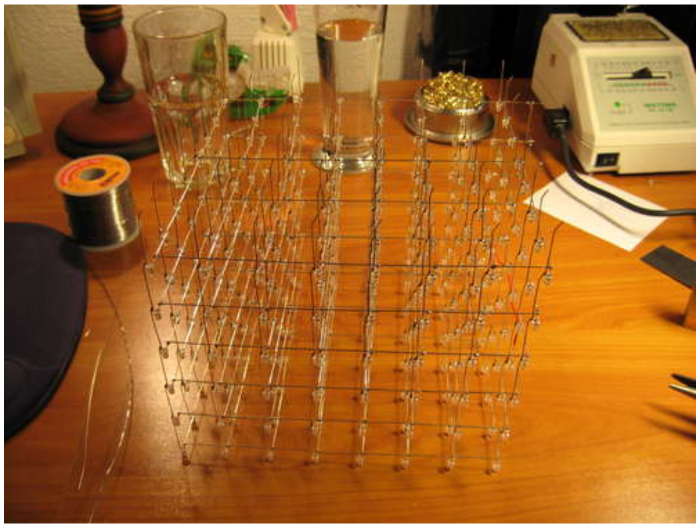
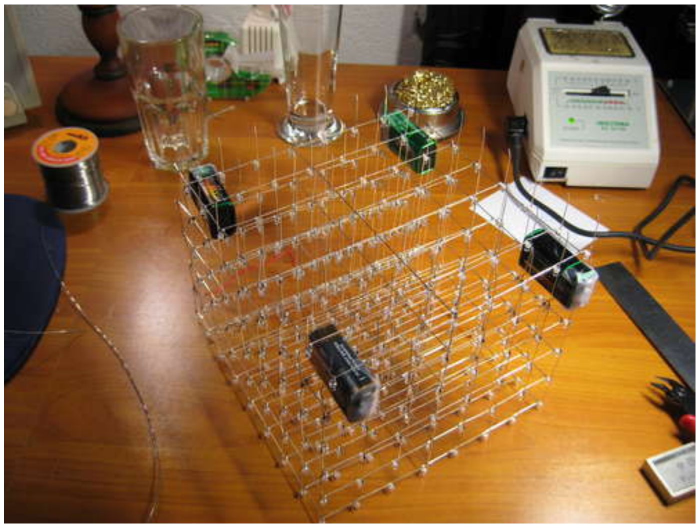
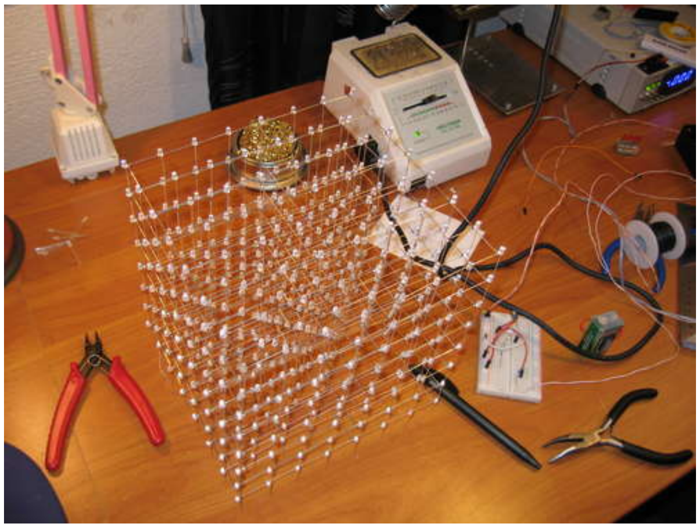
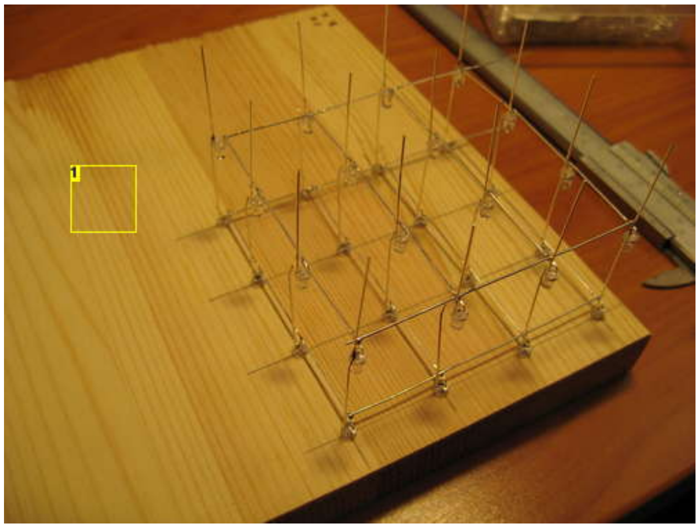
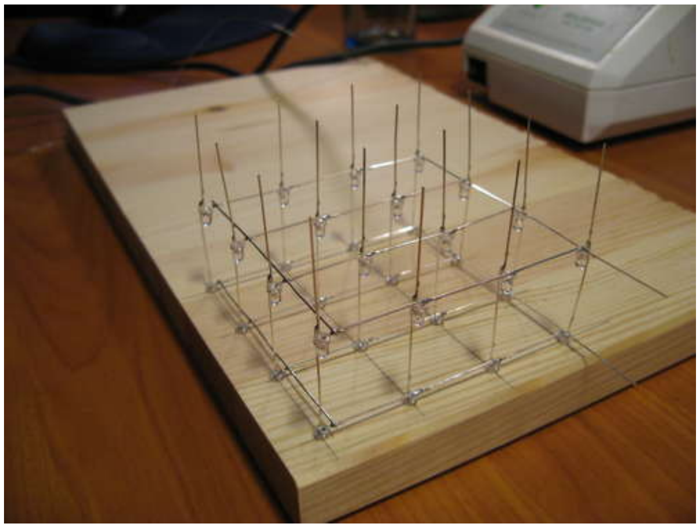
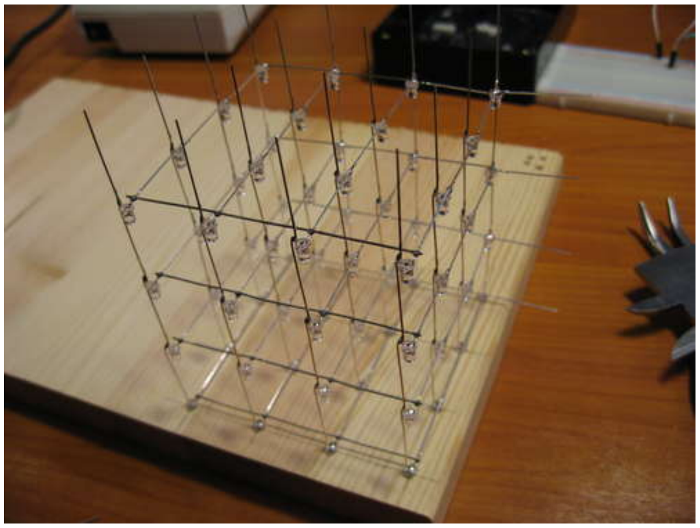
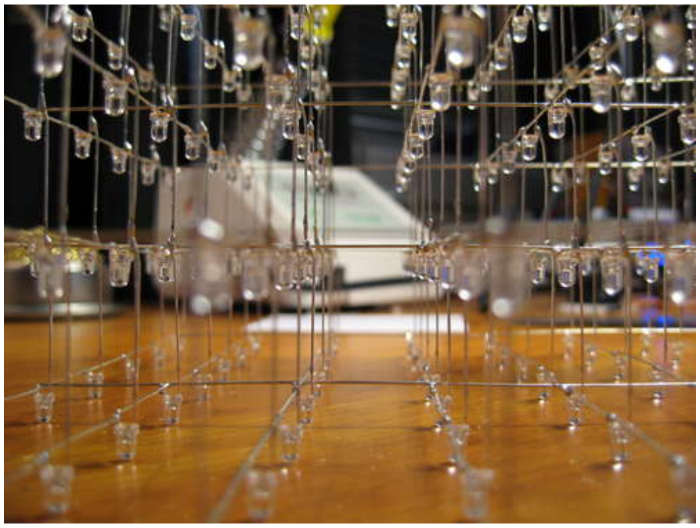





Image Notes
1. We added these 4x4x4 images to help illustrate the process.



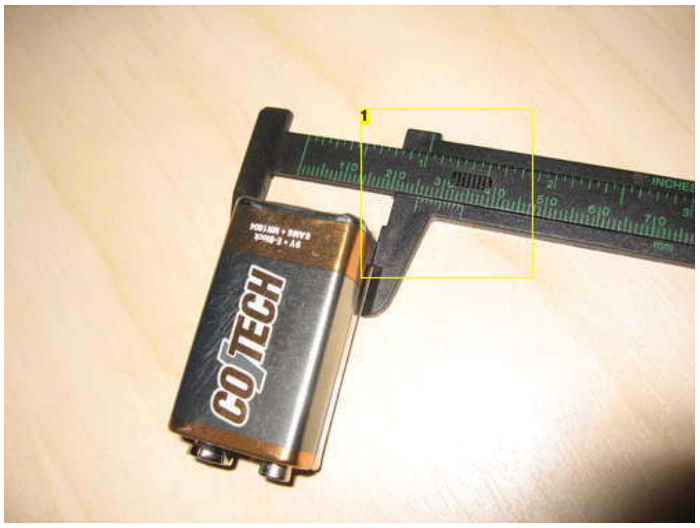


Image Notes
1. Almost exactly 25m!

Step 26: Build the cube: create the base

We didn't have any fancy tools at our disposal to create a fancy stand or box for our LED cube. Instead, we modified the template to work as a base for the cube.

We encourage you to make something cooler than we did for your LED cube!

For the template, we only drilled a couple of mm into the wood. To transform the template into a base, we just drilled all the holes through the board. Then we drilled 8 smaller holes for the 8 cathode wires running up to the 8 cathode layers.

Of course, you don't want to have your LED cube on a wood colored base. We didn't have any black paint lying around, but we did find a giant black magic marker! Staining the wood black with a magic marker worked surprisingly well! I think the one we used had a 10mm point.

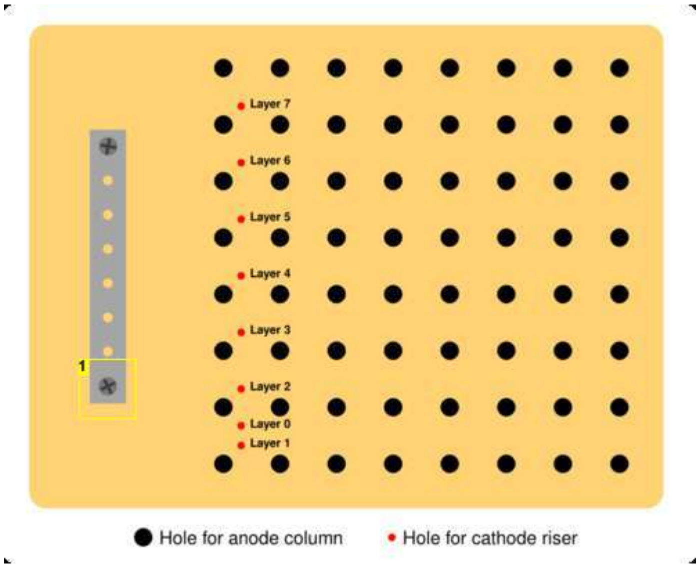


Image Notes
1. This is mounted on the underside of the board to hold the wires in place.

Image Notes
1. Drill all the way through.

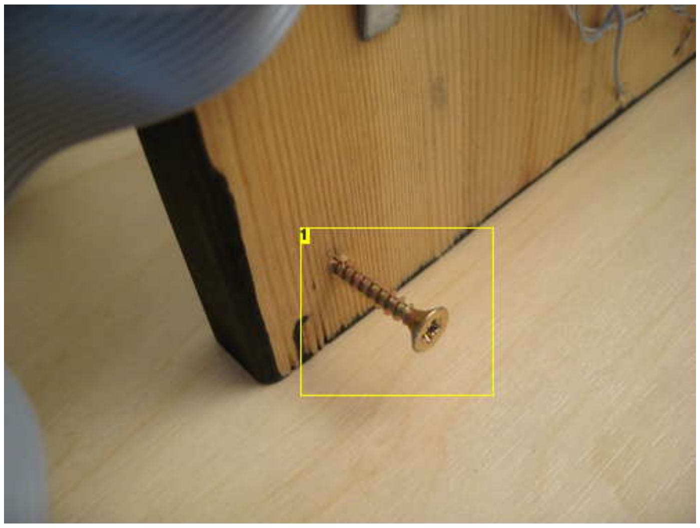


Image Notes
1. Didn't have any rubber feet that were high enough.

Image Notes
1. This is what we used to "paint" the base ;)

Step 27: Build the cube: mount the cube

Mount the cube. That sounds very easy, but it's not. You have to align 64 LED legs to slide through 64 holes at the same time. It's like threading a needle, times 64.

We found it easiest to start with one end, then gradually popping the legs into place. Use a pen or something to poke at the LED legs that miss their holes.

Once all 64 LED legs are poking through the base, carefully turn it on it's side. Then bend all 64 legs 90 degrees. This is enough to hold the cube firmly mounted to the base. No need for glue or anything else.

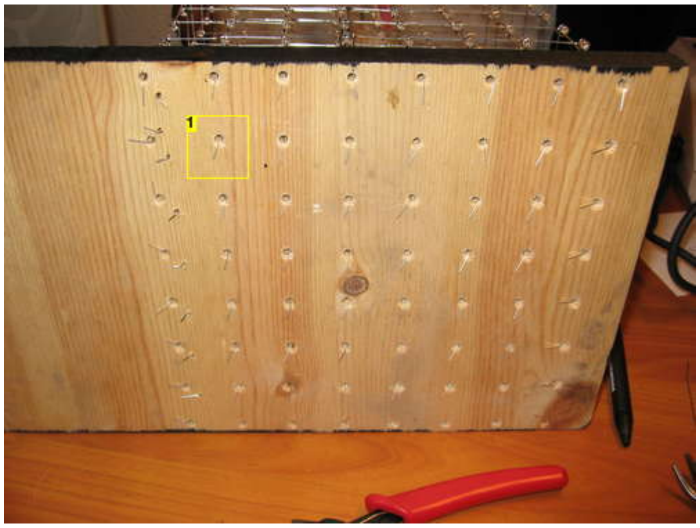


Image Notes
1. All the wires are bent 90 degrees. This is more than enough to hold the cube in place.

Step 28: Build the cube: cathode risers

You now have a LED cube with 64 anode connections on the underside of the base. But you need to connect the ground layers too.

Remember those 8 small holes you drilled in a previous step? We are going to use them now.

Make some straight wire using the method explained in a previous step.

We start with ground for layer 0. Take a short piece of straight wire, Make a bend approximately 10mm from the end. Poke it through the hole for ground layer 0. Leave 10mm poking through the underside of the base. Position it so that the bend you made rests on the back wire of ground layer 0. Now solder it in place.
Layer 1 through 7 are a little trickier. We used a helping hand to hold the wire in place while soldering.

Take a straight piece of wire and bend it 90 degrees 10mm from the end. Then cut it to length so that 10mm of wire will poke out through the underside of the base. Poke the wire through the hole and let the wire rest on the back wire of the layer you are connecting. Clamp the helping hand onto the wire, then solder it in place.

Rinse and repeat 7 more times.

Carefully turn the cube on it's side and bend the 8 ground wires 90 degrees.

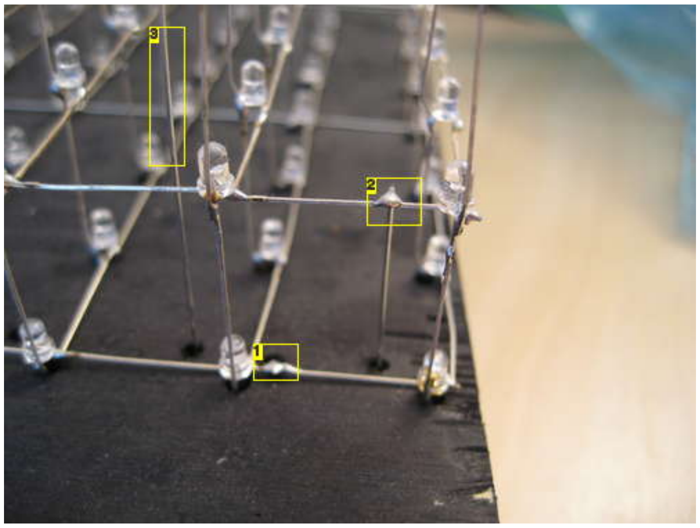


Image Notes
1. Ground wire for layer 0
2. Ground for layer 1
3. Ground for layer 2

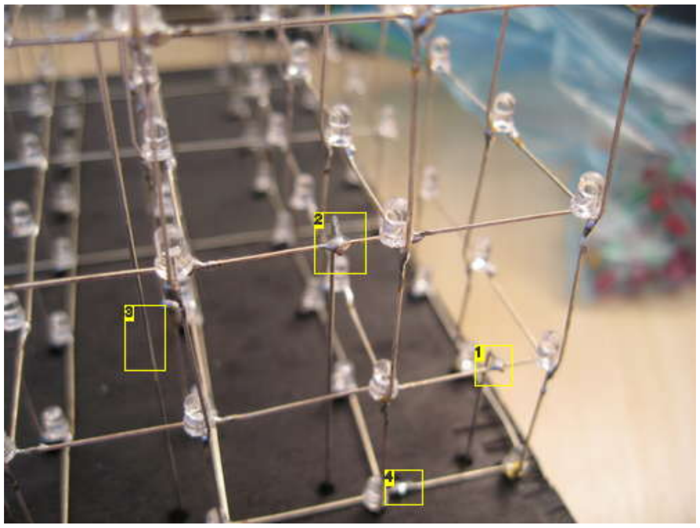


Image Notes
1. Layer 1
2. Layer 2
3. Layer 3
4. Layer 0

Step 29: Build the cube: attach cables

64+8 wires have to go from the controller to the LED cube. We used ribbon cable to make things a little easier.

The ground layers use an 8-wire ribbon cable.

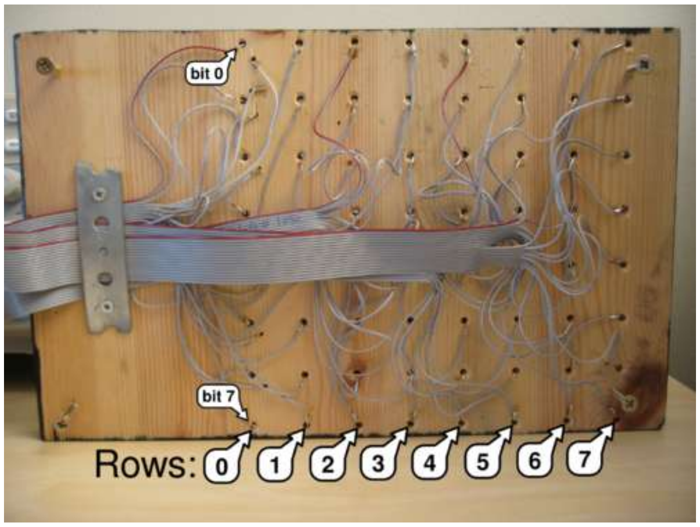
The cathodes are connected with 4 16-wire ribbon cables. Each of these ribbon cables are split in two at either end, to get two 8-wire cables.

At the controller side, we attached 0.1" female header connectors. These plug into standard 0.1" single row PCB header pins.

The header connector is a modular connector that comes in two parts, metal inserts and a plastic body.

The metal inserts are supposed to be crimped on with a tool. We didn't have the appropriate tool on hand, so we used pliers. We also added a little solder to make sure the wires didn't fall of with use.

1) Prepare one 8-wire ribbon cable and 4 16-wire ribbon cables of the desired length
2) Crimp or solder on the metal inserts.
3) Insert the metal insert into the plastic connector housing.
4) Solder the 8-wire ribbon cable to the cathode risers. Pre-tin the cables before soldering!
5) Solder in the rest of the cables. The red stripe on the first wire indicates that this is bit 0.
6) Tighten the screws on the strain relief to make sure everything stays in place.
7) Connect all the ribbon cables to the PCBs in the correct order. See pictures below. Our 8 wire ribbon cable didn't have a red wire. Just flip the connector 180 degrees if your cube is upside-down.



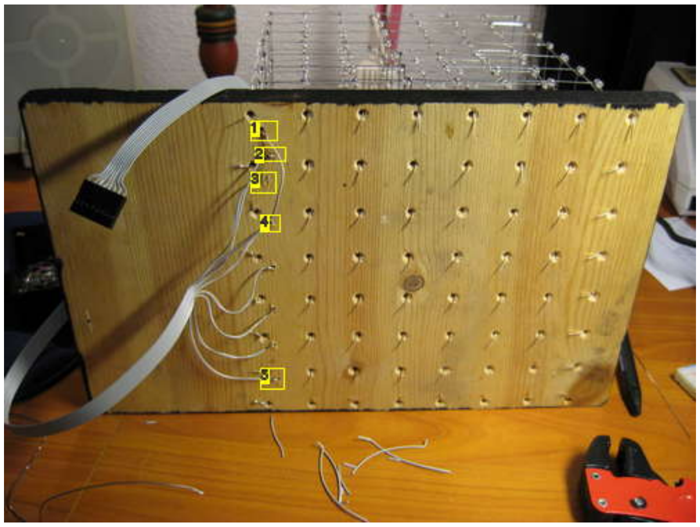


Image Notes
1. layer 1
2. Layer 0
3. layer 2
4. layer 4
5. layer 7

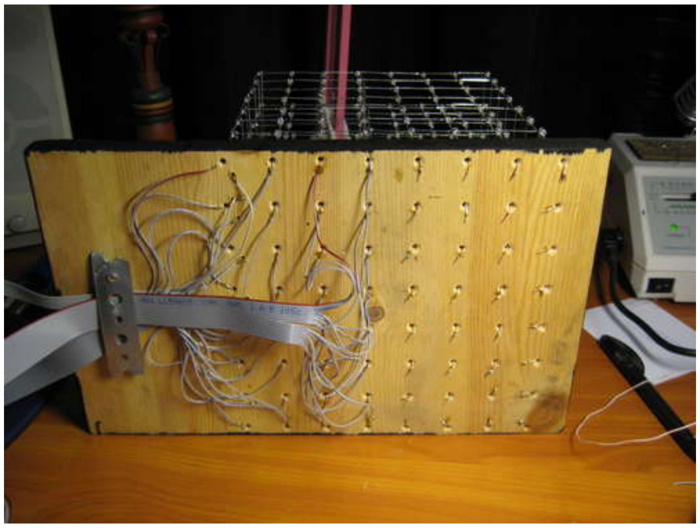
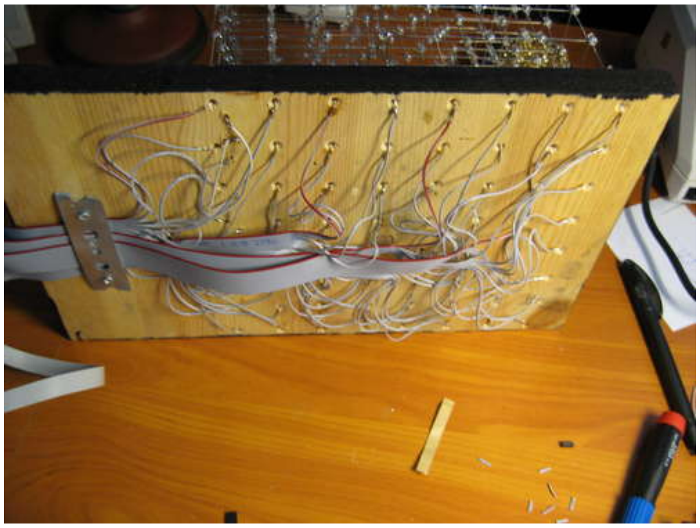
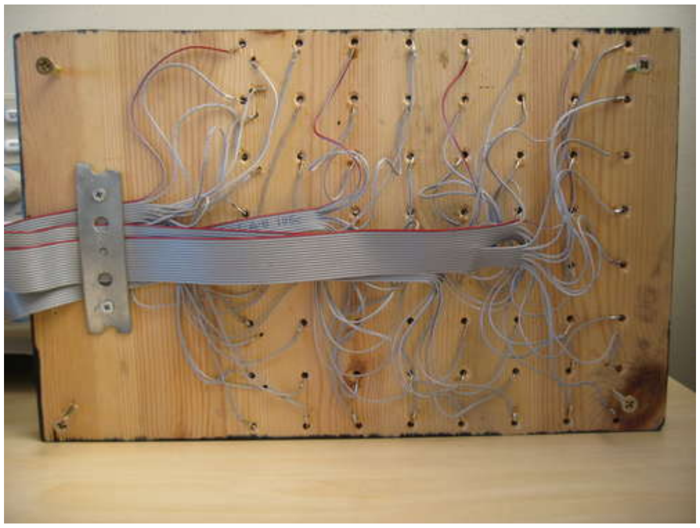
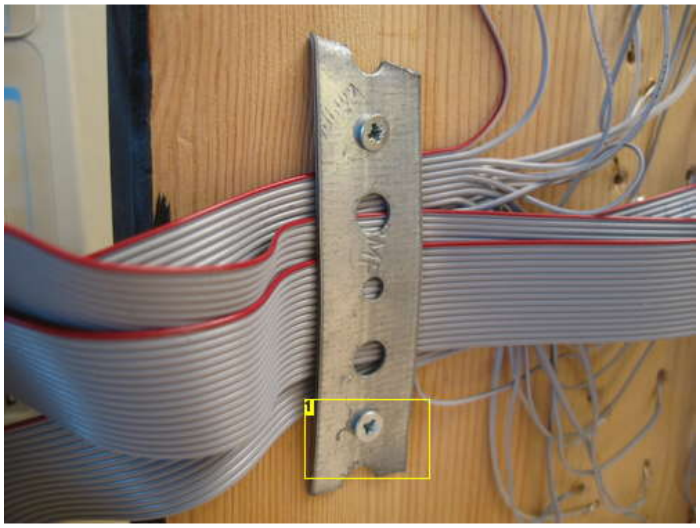
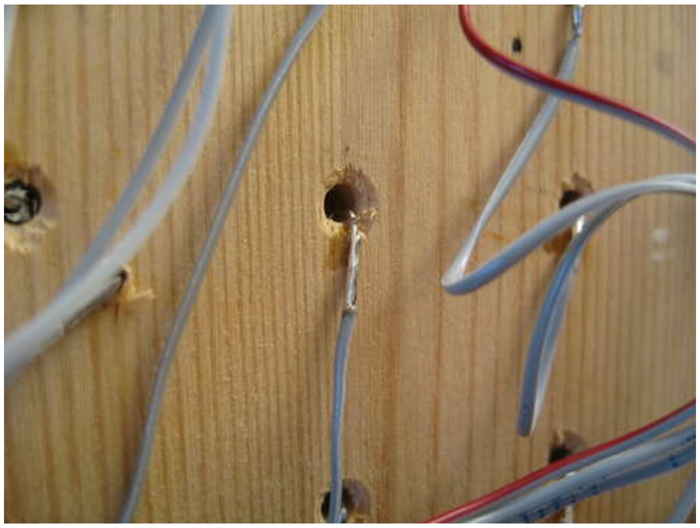
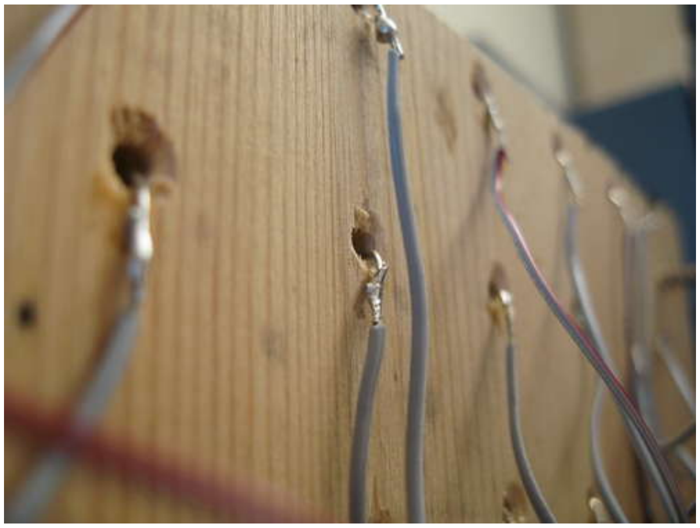
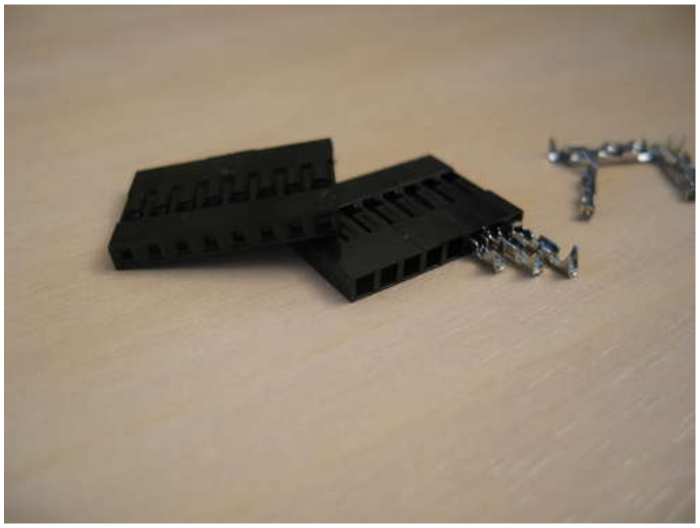





Image Notes
1. The connections are a bit flimsy. The cube will last a lot longer with this strain relief.



Step 30: Build the controller: layout

We took out the biggest type of PCB we had available (9x15cm) and started experimenting with different board layouts. It soon became clear that cramming all the components onto one board wasn't a good solution. Instead we decided to separate the latch array and power supply part of the circuit and place it on a separate board. A ribbon cable transfers data lines between the two boards.

Choosing two separate boards was a good decision. The latch array took up almost all the space of the circuit board. There wouldn't have been much space for the micro controller and other parts.

You may not have the exact same circuit boards as we do, or may want to arrange your components in a different way. Try to place all the components on your circuit board to see which layout best fits your circuit board.

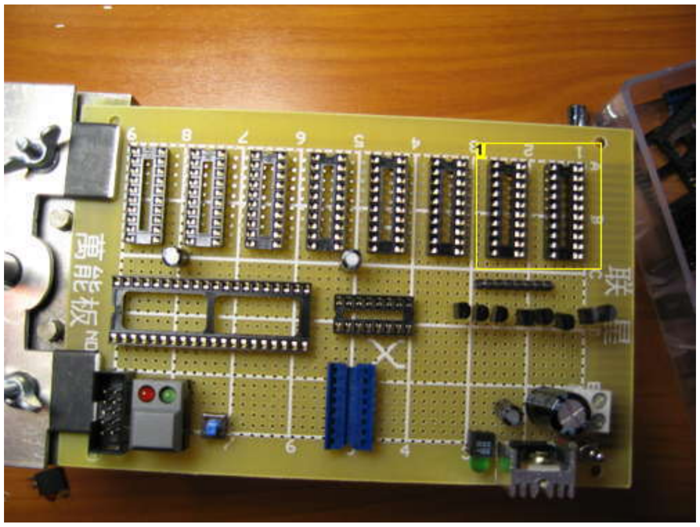


Image Notes
1. Way to little space in between the ICs. No room for resistors and connectors.

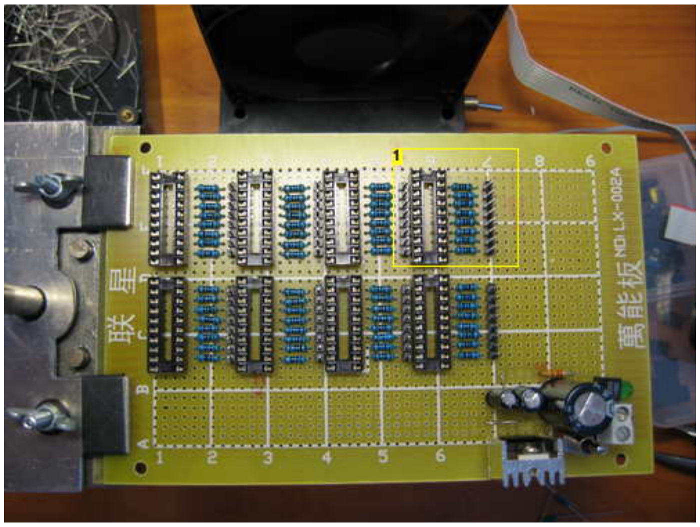
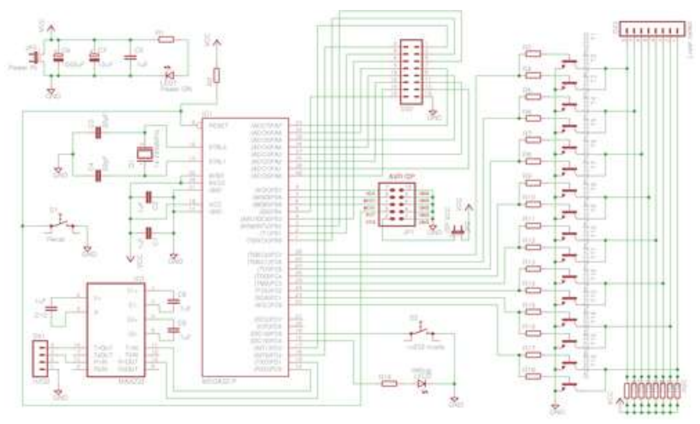
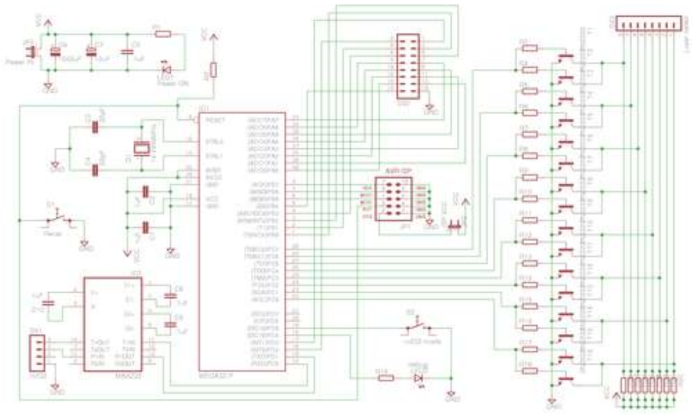
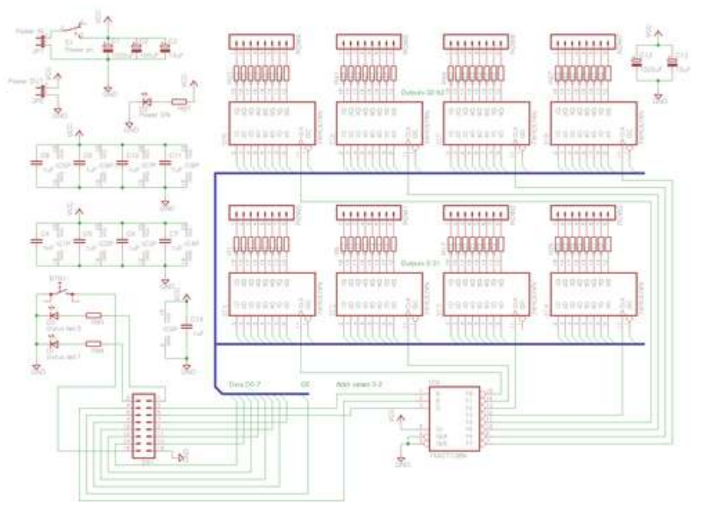


Image Notes
1. This is better.



File Downloads

multiplexer_board.sch (238 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'multiplexer_board.sch']

avr_board.sch (249 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'avr_board.sch']

Step 31: Build the controller: clock frequency

We use an external crystal of 14.7456 MHz to drive the ATmega system clock.

You may be thinking that this is an odd number to use, and why we didn't run the ATmega at the 16MHz it is rated for.

We want to be able to control the LED cube from a computer, using RS232. Serial communication requires precise timing. If the timing is off, only by a little bit, some bits are going to be missed or counted double from time to time. We won't be running any error correcting algorithms on the serial communications, so any error over the line would be represented in the LED cube as a voxel being on or off in the wrong place.

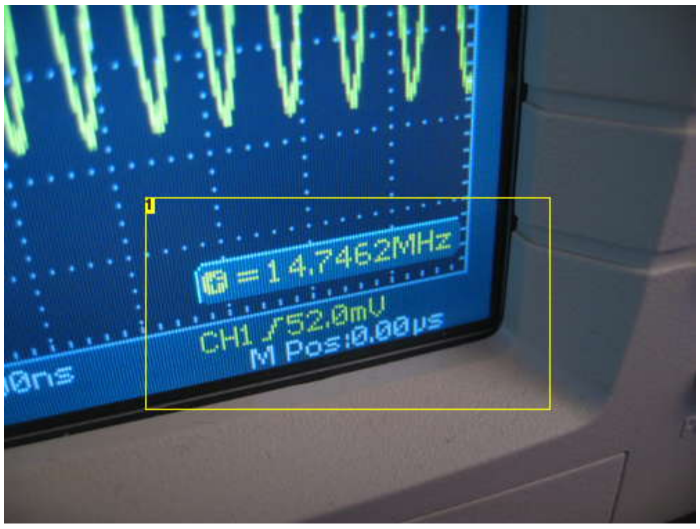
To get flawless serial communication, you have to use a clock frequency that can be divided by the serial frequency you want to use.

14.7456 MHz is dividable by all the popular RS232 baud rates.

- (14.7456MHz*1000*1000) / 9600 baud = 1536.0
- (14.7456MHz*1000*1000) / 19200 baud = 768.0
- (14.7456MHz*1000*1000) / 38400 baud = 384.0
- (14.7456MHz*1000*1000) / 115200 baud = 128.0

The formula inside the parentheses converts from MHz to Hz. First *1000 gives you KHz, the next Hz.

As you can see all of these RS232 baud rates can be cleanly divided by our clock rate. Serial communication will be error free!



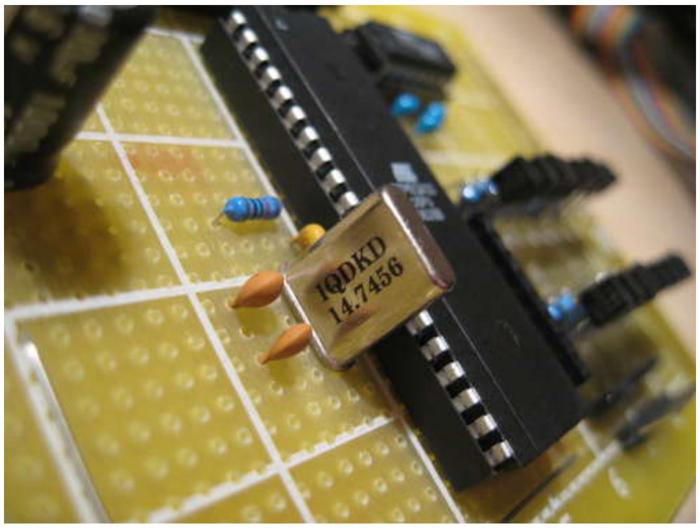


Image Notes
1. This is the frequency of the system clock

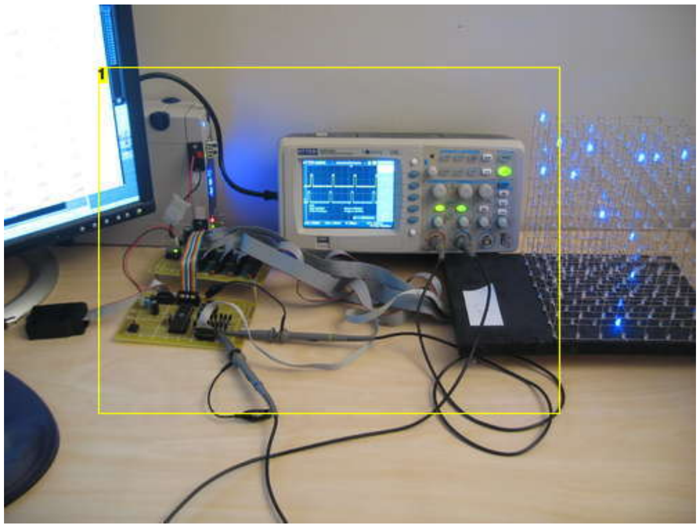


Image Notes
1. I got an oscilloscope for Christmas :D we used it to visualize some of the signals in the LED cube.

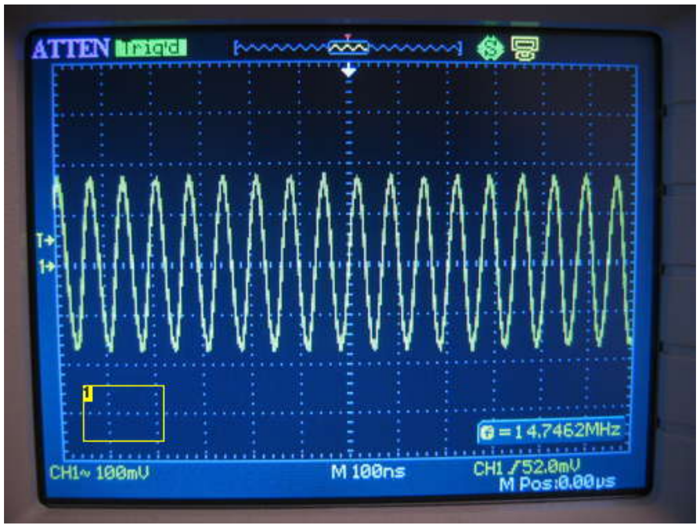


Image Notes
1. This is what the clock signal from a crystal looks like

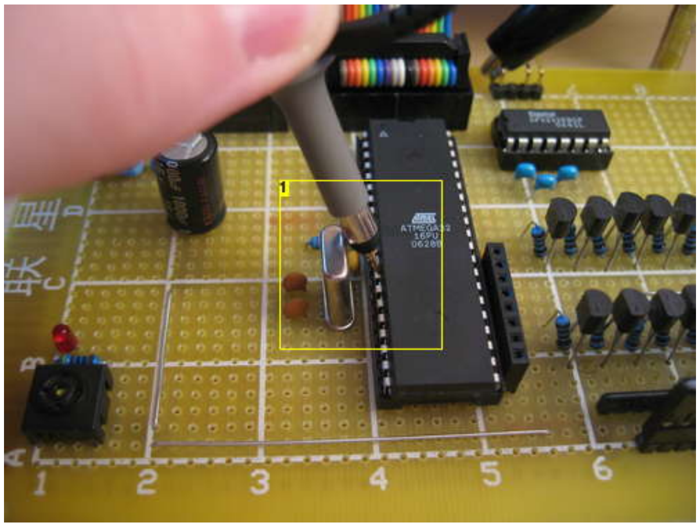


Image Notes
1. Probing the crystal

Step 32: Build the controller: protoboard soldering advice

We see people do a lot of weird stuff when they solder on prototype PCBs. Before you continue, we just want to share with you the process we use to create tracks on prototype PCBs with solder eyes. Once you master this technique, you will probably start using it a lot.

1) Fill each point of the track you want to make with solder.
2) Connect every other points by heating them and adding a little solder.
3) Connect the 2-hole long pieces you now have spanning the desired track.
4) Look how beautiful the result is.

You can see in the video how we do it. We had to touch some of the points twice to join them. It was a bit hard to have the camera in the way when we were soldering ;)

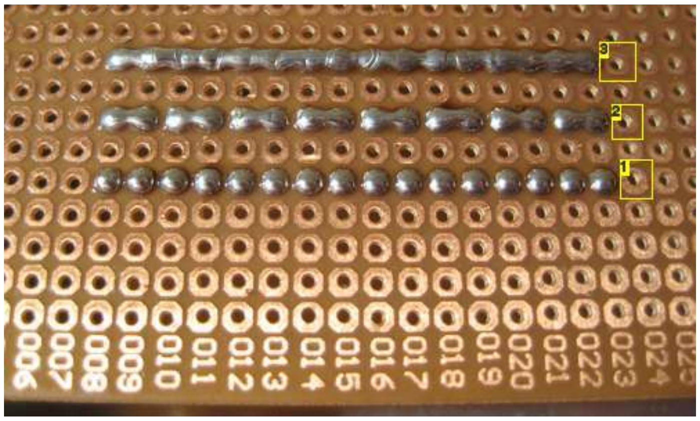


Image Notes
1. 1
2. 2
3. 3

Step 33: Build the controller: Power terminal and filtering capacitors

The cube is complete, now all that remains is a monster circuit to control the thing.

Let's start with the easiest part, the "power supply".

The power supply consists of a screw terminal where you connect the GND and VCC wires, some filtering capacitors, a switch and a an LED to indicate power on.

Initially, we had designed an on-board power supply using an LM7805 step down voltage regulator. However, this turned out to be a big fail.

We used this with a 12V wall wart. But as you may already know, most wall warts output higher voltages than the ones specified on the label. Ours outputted something like 14 volts. The LM7805 isn't a very sophisticated voltage regulator, it just uses resistance to step down the voltage. To get 5 volts output from 14 volts input means that the LM7805 has to drop 9 volts. The excess energy is dispersed as heat. Even with the heat sink that you see in the picture, it became very very hot. Way to hot to touch! In addition to that, the performance wasn't great either. It wasn't able to supply the necessary current to run the cube at full brightness.

The LM7805 was later removed, and a wire was soldered between the input and output pins. Instead we used an external 5V power source, as covered in a previous step.

Why so many capacitors?

The LED cube is going to be switching about 500mA on and off several hundred times per second. The moment the 500mA load is switched on, the voltage is going to drop across the entire circuit. Many things contribute to this. Resistance in the wires leading to the power supply, slowness in the power supply to compensate for the increase in load, and probably some other things that we didn't know about ;)

By adding capacitors, you create a buffer between the circuit and the power supply. When the 500mA load is switched on, the required current can be drawn from the capacitors during the time it takes the power supply to compensate for the increase in load.

Large capacitors can supply larger currents for longer periods of time, whereas smaller capacitors can supply small but quick bursts of energy.

We placed a 1000uF capacitor just after the main power switch. This works as our main power buffer. After that, there is a 100uF capacitor. It is common practice to have a large capacitor at the input pin of an LM7805 and a smaller capacitor at it's output pin. The 100uF capacitor probably isn't necessary, but we think capacitors make your circuit look cooler!

The LED is connected to VCC just after the main power switch, via a resistor.

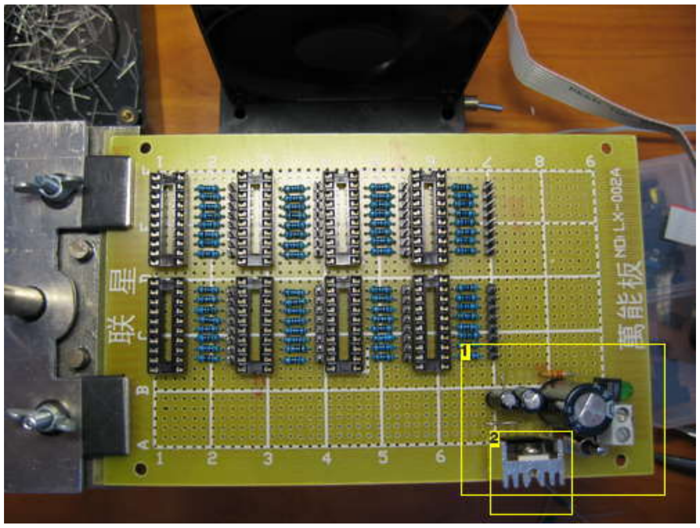


Image Notes
1. Power supply

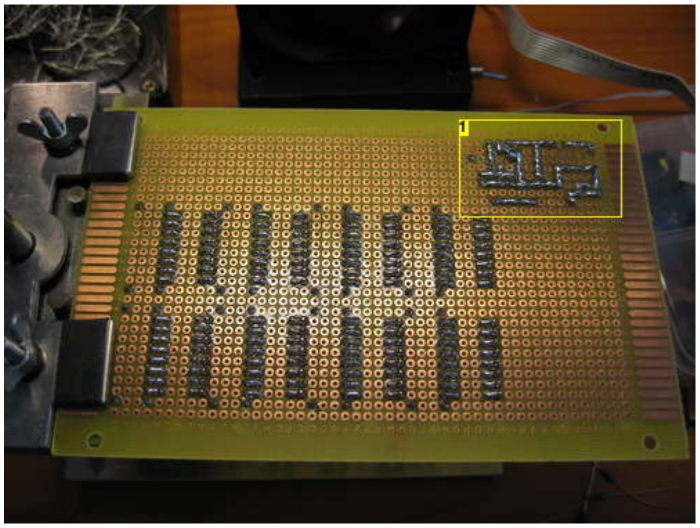


Image Notes
1. Bottom side of power supply. See, only solder traces. No wires.

2. This was removed later, because it couldn't deliver the needed amps.

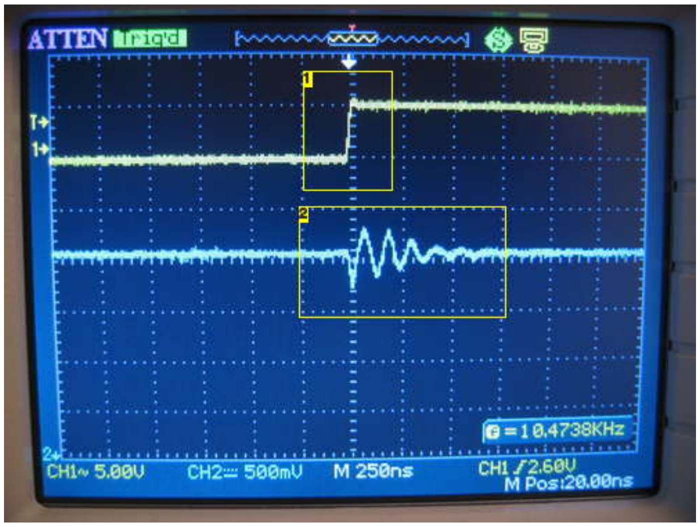


Image Notes
1. A layer in the led cube is switched on.
2. The resulting rise in current draw makes VCC fluctuate a little

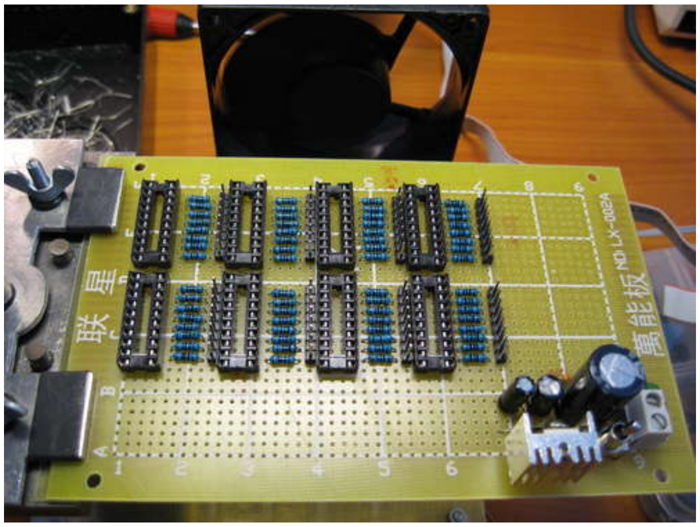
Step 34: Build the controller: IC sockets, resistors and connectors

In this step you will be soldering in the main components of the multiplexer array.

Our main design consideration here was to minimize soldering and wiring. We opted to place the connectors as close to the ICs as possible. On the output-side, there is only two solder joints per LED cube column. IC-resistor, resistor-connector. The outputs of the latches are arranged in order 0-7, so this works out great. If we remember correctly, the latch we are using is available in two versions, one with the inputs and outputs in sequential order, and one with the in- and outputs in seemingly random order. Do not get that one! ;) Don't worry, it has a different 74HC-xxx name, so you'll be good if you stick to our component list.

In the first picture, you can see that we have placed all the IC sockets, resistors and connectors. We squeezed it as tight as possible, to leave room for unforeseen stuff in the future, like buttons or status LEDs.

In the second picture, you can see the solder joints between the resistors and the IC sockets and connectors. Note that the input side of the latch IC sockets haven't been soldered yet in this picture.



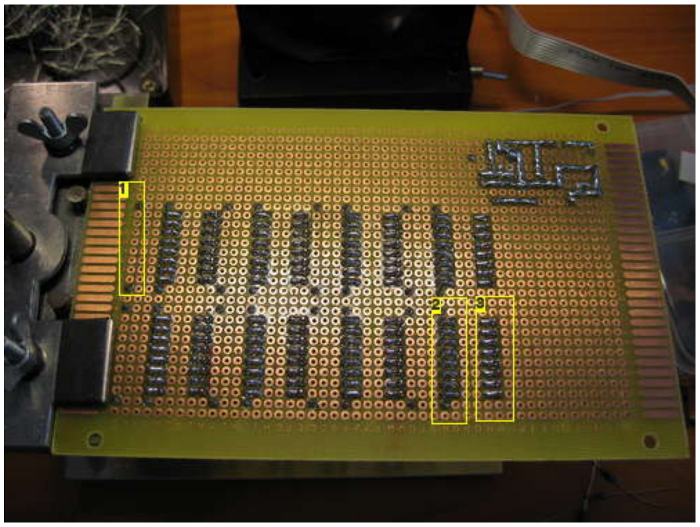


Image Notes
1. Input side not soldered yet.
2. Resistor soldered to IC
3. Resistor soldered to connector

Step 35: Build the controller: Power rails and IC power

Remember that protoboard soldering trick we showed you in a previous step? We told you it would come in handy, and here is where you use it.

Large circuit boards like this one, with lots of wires, can become quite confusing. We always try to avoid lifting the GND and VCC lines off the board. We solder them as continuous solder lines. This makes it very easy to identify what is GND/VCC and what is signal lines.

If the VCC and GND lines needs to cross paths, simply route one of them over the other using a piece of wire on the top side of the PCB.

In the first picture you can see some solder traces in place.

The two horizontal traces is the "main power bus". The lowest one is VCC and the top one is GND. For every row of ICs a GND and VCC line is forked off the main power bus. The GND line runs under the ICs, and the VCC line runs under the resistors.

We went a little overboard when making straight wire for the cube, and had some pieces left over. We used that for the VCC line that runs under the resistors.

In the bottom right corner, you can see that we have started soldering the 8+1bit bus connecting all the latch ICs. Look how easy it is to see what is signal wires and what is power distribution!

In the second picture, you can see the board right-side-up, with some additional components soldered in, just ignore them for the moment.

For every latch IC (74HC574), there is a 100nF (0.1uF) ceramic capacitor. These are noise reduction capacitors. When the current on the output pins are switched on and off, this can cause the voltage to drop enough to mess with the internal workings of the ICs, for a split second. This is unlikely, but it's better to be safe than sorry. Debugging a circuit with noise issues can be very frustrating. Besides, capacitors make the circuit look that much cooler and professional! The 100nF capacitors make sure that there is some current available right next to the IC in case there is a sudden drop in voltage. We read somewhere that it is common engineering practice to place a 100nF capacitor next to every IC, "Use them like candy". We tend to follow that principle.

Below each row of resistors, you can see a tiny piece of wire. This is the VCC line making a little jump to the top side of the board to cross the main GND line.

We also added a capacitor on the far end of the main power bus, for good measure.

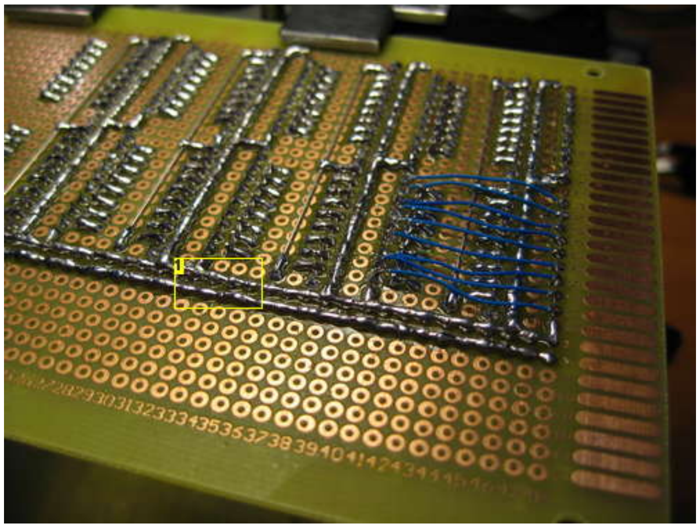


Image Notes
1. GND and VCC runs along the length of the board.

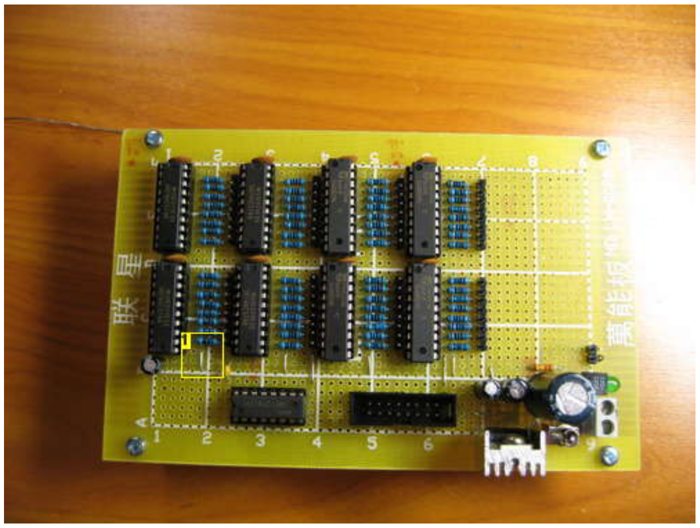


Image Notes
1. VCC crosses GND once for each row of ICs

Step 36: Build the controller: Connect the ICs, 8bit bus + OE

In the picture, you'll notice a lot of wires have come into place.

All the tiny blue wires make up the 8+1bit bus that connects all the latch ICs. 8 bits are for data, and the +1 bit is the output enable line.

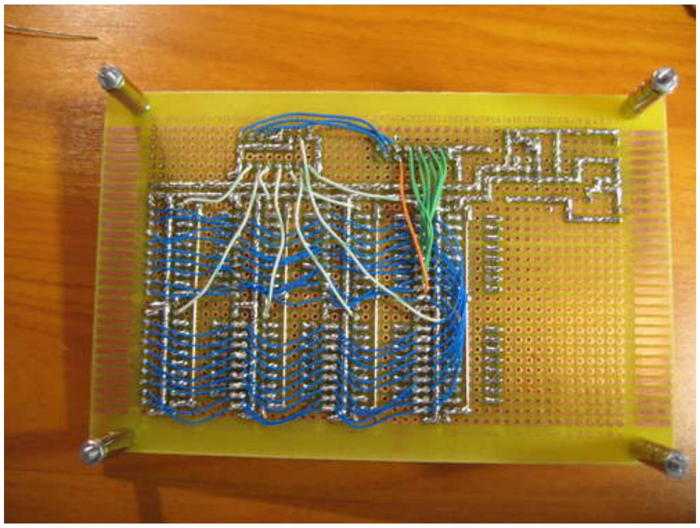
At the top of the board, we have added a 16 pin connector. This connects the latch board to the micro controller board. Next to that, you see the 74HC138.

The tiny blue wires are Kynar wire. This is a 30 or 32 AWG (american wire gauge) wire. Very tiny. We love working with this type of wire. Because it is so thin, it doesn't take up that much space on the circuit board. If we had used thicker wire, you wouldn't be able to see the board through all the wires. Kynar wire is coated with tin, so you can solder directly after stripping it. No need for pre-tinning. The tiny blue wires are connected to the same pin on every latch IC.

From the connector at the top, you can see 8 green wires connected to the bus. This is the 8 bit data bus. We used different colors for different functions to better visualize how the circuit is built.

The orange wire connected to the bus is the output enable (OE) line.

On the right hand side of the connector, the first pin is connected to ground.



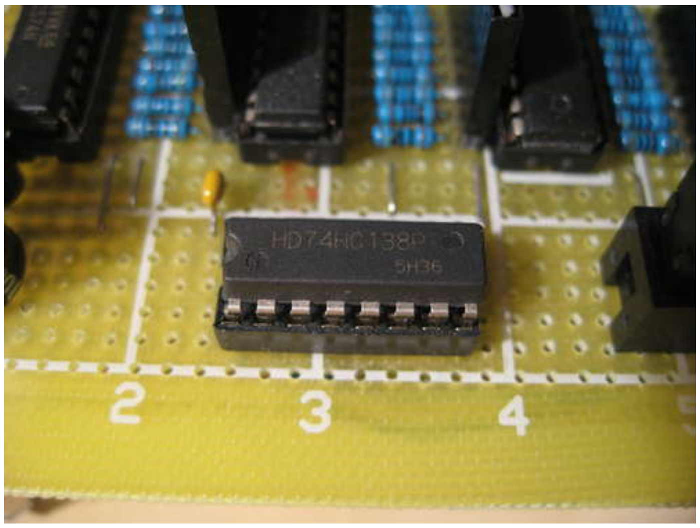
Step 37: Build the controller: Address selector

The 74HC138 is responsible for toggling the clock pin on the 74HC574 latch ICs. We call this an address selector because it selects which one of the 8 bytes in the latch array we want to write data to. The three blue wires running from the connector to the 74HC138 is the 3 bit binary input used to select which of the 8 outputs is pulled low. From each of the outputs on the 74HC138, there is a wire (white) running to the clock pin on the corresponding 74HC574 latch IC.

Start by soldering the GND and VCC connections. If you use the solder trace method to run GND/VCC lines you want to do this before you solder any other wires in place. A 100nF ceramic filtering capacitor is placed close to the VCC and GND pins of the 74HC138.

Then connect the address lines and the 8 clock lines.

If you look carefully at the connector, you can see two pins that are not used. These will be used for a button and debug LED later.



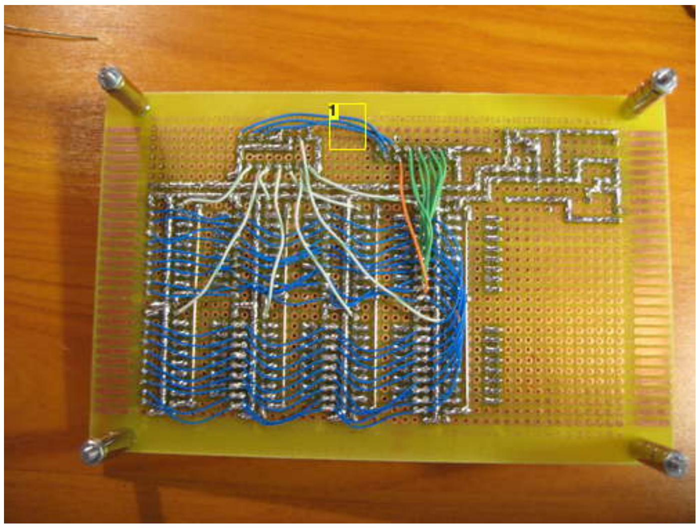


Image Notes
1. 3 bit address select bus

Step 38: Build the controller: AVR board

Braaaaainzz!!!

This board is the brain of the LED cube. The main component is an Atmel AVR ATmega32.

This is an 8 bit microcontroller with 32 KB of program memory and 2 KB RAM. The ATmega32 has 32 GPIO (General Purpose IO) pins. Two of these will be used for serial communication (TX+RX). Three IO pins are used for ISP (In-circuit Serial Programming). This leaves us with 27 GPIO to drive the LED cube, buttons and status LEDs.

A group of 8 GPIO (8 bits, one byte) is called a port. The ATmega32 has 4 ports. PORTA, PORTB, PORTC and PORTD. On PORTC and PORTD some of the pins are used for TX/RX and ISP. On PORTA and PORTB, all the pins are available. We use these ports to drive the data bus of the latch array and layer select transistor array.

PORTA is connected to the data bus on the latch array.

Each pin on PORTC is connected to a pair of transistors that drive a ground layer.

The address selector on the latch array (74HC138) is connected to bit 0-2 on PORTB. Output enable (OE) is connected to PORTB bit 3.

In the first image, you see the AVR board right-side-up.

The large 40 pin PDIP (Plastic Dual Inline Package) chip in the center of the board is the ATmega32, the brainz! Just to the left of the ATmega, you see the crystal oscillator and it's two capacitors. On either side of the ATmega there is a 100nF filtering capacitor. One for GND/VCC and one for AVCC/GND.

In the top left corner, there is a two pin connectors and two filtering capacitors. One 10uF and one 100nF. The LED is just connected to VCC via a resistor, and indicates power on.

The large 16 pin connector directly above the ATmega connects to the latch array board via a ribbon cable. The pinout on this corresponds to the pinout on the other board.

The smaller 10 pin connector to the left, is a standard AVR ISP programming header. It has GND, VCC, RESET, SCK, MISO and MOSI, which are used for programming. Next to it, there is a jumper. When this is in place, the board can be powered from the programmer.

Caution: DO NOT power the board from the programmer when the actual LED cube is connected to the controller. This could possibly blow the programmer and even the USB port the programmer is connected to!

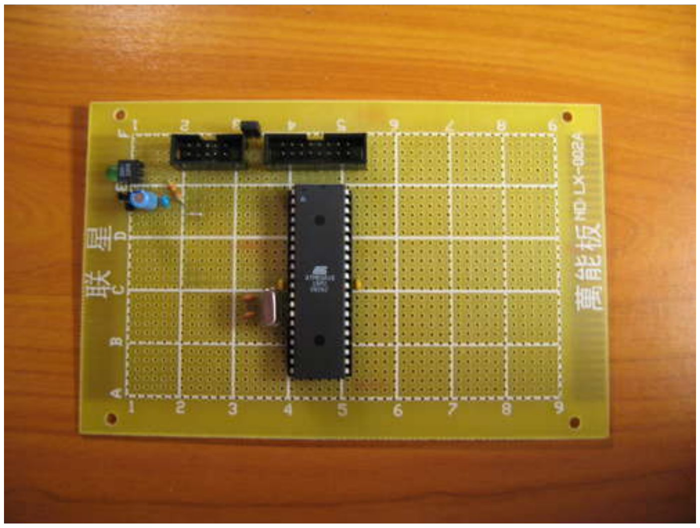
The second image shows the underside. Again all GND and VCC lines are soldered as traces on the protoboard or bare wire. We had some more left over straight metal wire, so we used this.

The orange wires connect the ATmega's RESET, SCK, MOSI and MISO pins to the ISP programming header.

The Green wires connect PORTA to the data bus.

The blue wires are the address select lines for the 74HC138 and output enable (OE) for the latch array.

1) Start by placing the 40 pin IC socket, the 10 pin ISP connector with a jumper next to it and the 16 pin data bus connector.
2) Solder in place the power connector, capacitors and power indicator LED.
3) Connect all the GND and VCC lines using solder traces or wire. Place a 100nF capacitor between each pair of GND/VCC pins on the ATmega.
4) Solder in the crystal and the two 22pF capacitors. Each capacitor is connected to a pin on the crystal and GND.
5) Run all the data bus, address select and OE wires, and the ISP wires.
Transistors, buttons and RS232 will be added in later steps.
At this time, the AVR board can be connected to an ISP programmer and the ATmega should be recognized.



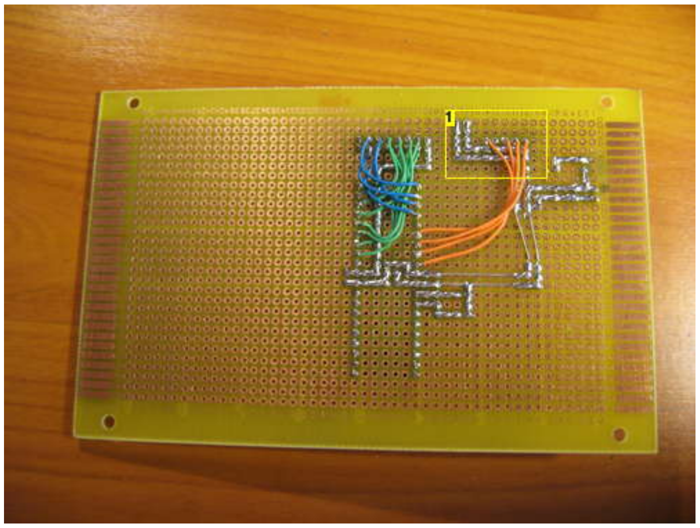
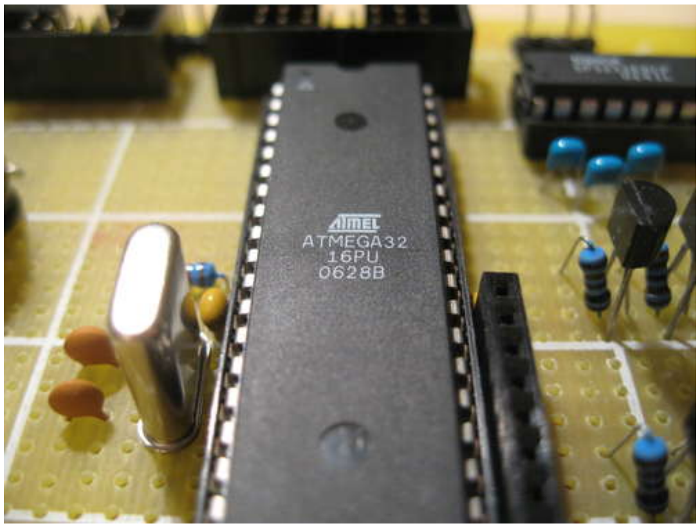
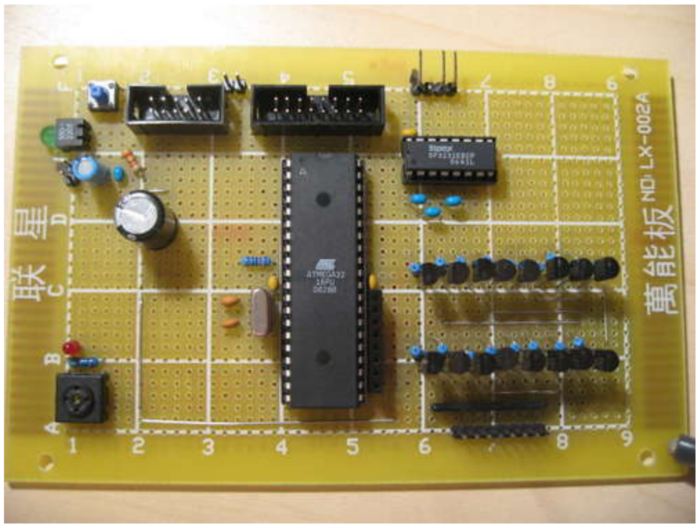


Image Notes
1. In circuit serial programming header.





Step 39: Build the controller: Transistor array

The transistor array is responsible for switching on and off GND for each layer in the LED cube.

Our first attempt at this was an epic fail. We bought some transistors rated for over 500mA, thinking that would be plenty of juice. We don't remember the model number.

The LED cube worked, but it wasn't very bright, and the brightness was inversely proportional to the number of LEDs switched on in any given layer. In addition to that, there was some ghosting. Layers didn't switch completely off when they were supposed to be off.

Needless to say, we were kind of disappointed, and started debugging. The first thing we did was to add pull-up resistors to try to combat the ghosting. This removed almost all the ghosting, yay! But the cube was still very dim, bah!

We didn't have any powerful transistors or MOSFETs lying around, so we had to come up with another solution.

We posted a thread in the electronics section of the AVRFreaks.net forum, asking if it was possible to use two smaller transistors in parallel. This is the only option available to us using the parts we had on hand. The general response was, this will never work so don't even bother trying. They even had valid theories and stuff, but that didn't deter us from trying. It was our only solution that didn't involve waiting for new parts to arrive in the mail.

We ended up trying PN2222A, NPN general purpose amplifier. Ideally, you'd want a switching transistor for this kind of application, but we needed 16 transistors of the same type. This transistor was rated at 1000mA current, so we decided to give it a try.

For each layer, we used two PN2222As in parallel. The collectors connected together to GND. The emitters connected together, then connected to a ground layer. The base of each transistors was connected to it's own resistor, and the two resistors connected to an output pin on the ATmega.

We soldered in all the transistors and turned the thing on again, and it worked, perfectly!

If you know what you are doing, you should probably do some research and find a more suitable transistor or MOSFET. But our solution is tried and tested and also does the trick!

1) Start by placing all 8 all transistors on the PBC and soldering each of their pins.

2) Run a solder trace between the the emitters of all 16 transistors. Connect this solder trace to GND.

3) Solder in a resistor for each transistor, the solder the resistors together in pairs of two.

4) Run kynar wire from the output pins on the ATmega to each of the 8 resistor pairs.

5) Solder together the collectors of the transistors in pairs of two and run solder trace or wire from the collector pairs to an 8 pin header.

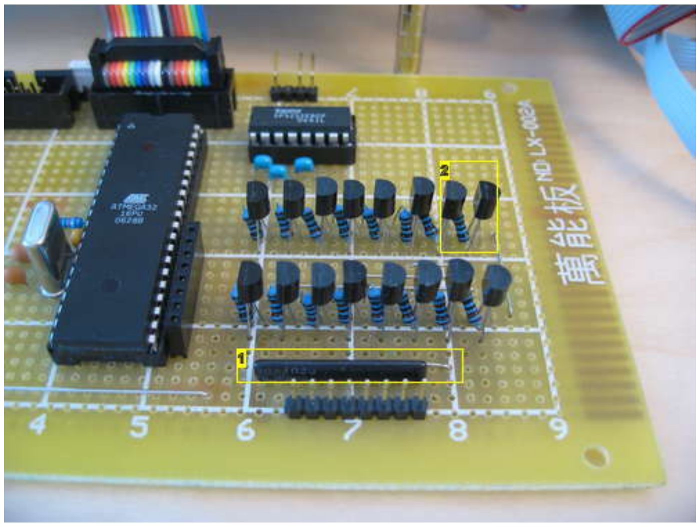


Image Notes
1. Pull up resistors. This type of resistor is called a resistor network. It just has a bunch of resistors connected to a common pin.
2. Two and two resistors work together.

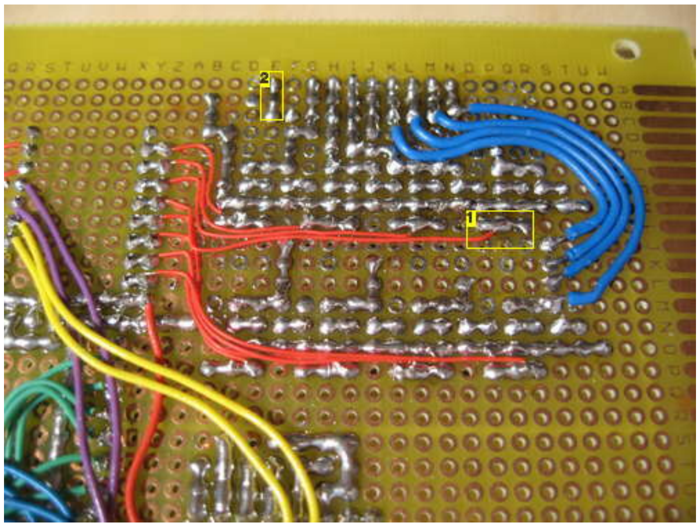
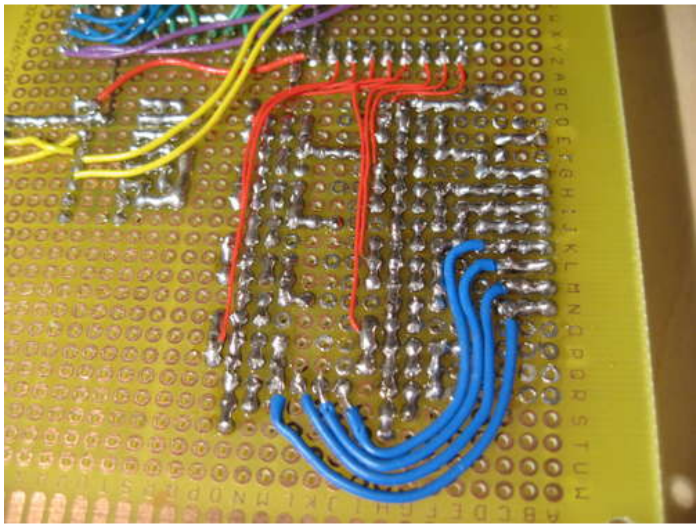


Image Notes
1. Signal goes to two transistors.
2. This point was connected to VCC after this picture was taken.



Step 40: Build the controller: Buttons and status LEDs

You can make a LED cube without any buttons at all, but it's nice to have at least one button and some status LEDs for debugging.

We added one awesome looking button with two built in LEDs, and one regular button with an LED.

The first button is mounted on the latch array PCB, since this will sit on top of the AVR board, and we want the button easily accessible. The wires are routed through the ribbon cable. The second button and LED sits on the AVR board and was mostly used for debugging during construction.

The buttons are connected between GND and the IO pin on the ATmega. An internal pull-up resistor inside the ATmega is used to pull the pin high when the button is not pressed. When the button is pressed, the IO pin is pulled low. A logic 0 indicates that a button has been pressed.

The LEDs are also connected between GND and the IO pin via a resistor of appropriate size. Don't connect an LED to a micro controller IO pin without having a resistor connected in series. The resistor is there to limit the current, and skipping it can blow the IO port on your micro controller.

To find the appropriate resistor, just plug the led into a breadboard and test different resistors with a 5v power supply. Choose the resistors that make the LED light up with the brightness you want. If you use LEDs with different colors, you should test them side by side. Different color LEDs usually require different resistors to reach the same level of brightness.

We will leave it up to you to decide the placement of your status LEDs, but you can see in the pictures below how we did it:

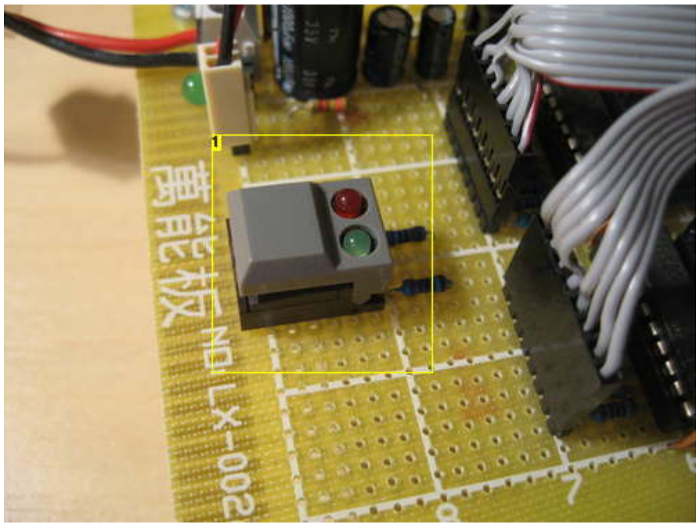


Image Notes
1. Start the cube in autonomous mode

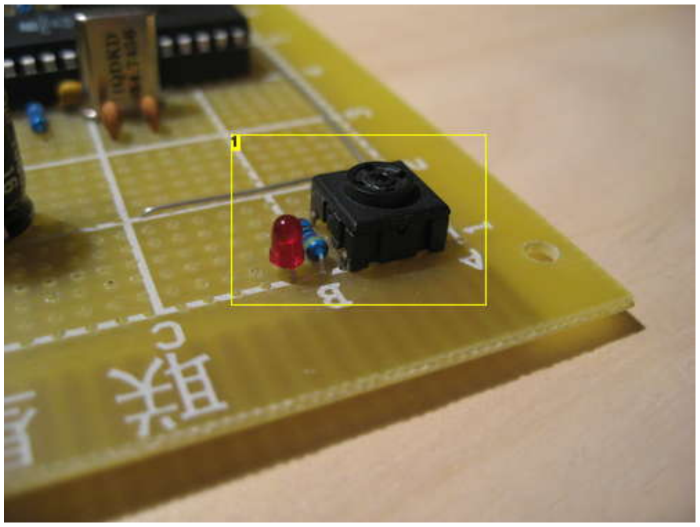


Image Notes
1. Start the cube in rs232-mode

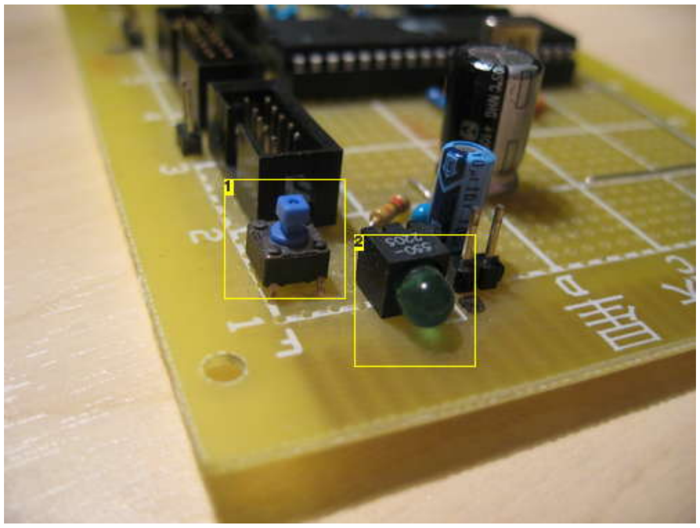


Image Notes
1. Reset
2. Power on

Step 41: Build the controller: RS-232

To get the truly amazing animations, we need to connect the LED cube to a PC. The PC can do floating point calculations that would have the AVR working in slow motion.

The ATmega has a built in serial interface called USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter).

The USART communicates using TTL levels (0/5 volts). The computer talks serial using RS232. The signal levels for RS232 are anywhere from +/- 5 volts to +/- 15 volts.

To convert the serial signals from the micro controller to something the RS232 port on a PC can understand, and vice versa, we use the Maxim MAX232 IC. Actually, the chip we are using isn't from Maxim, but it is a pin-compatible clone.

There are some 100nF ceramic capacitors surrounding the MAX232. The MAX232 uses internal charge-pumps and the external capacitors to step up the voltage to appropriate RS232 levels. One of the 100nF capacitors is a filter capacitor.

The RS232 connector is at a 90 degree angle for easy access when the latch array board is mounted on top of the AVR board. We used a 4 pin connector and cut one of the pins out to make a polarized connector. This removes any confusion as to which way to plug in the RS232 cable.

In the second picture you can see two yellow wires running from the ATmega to the MAX232. These are the TTL level TX and RX lines.

1) Connect the GND and VCC pins using solder trace or wire. Place a 100nF capacitor close to the GND and VCC pins.

2) Solder in place the rest of the 100nF capacitors. You can solder these with solder traces, so its best to do this before you connect the tx/rx wires.

3) Solder in place a 4 pin 0.1" header with one pin removed. Connect the pin next to the one that was removed to GND.

4) Connect the tx/rx input lines to the micro controller, and the tx/rx output lines to the 4 pin header.

The wires going to the 4 pin header are crossed because the first serial cable we used had this pinout.

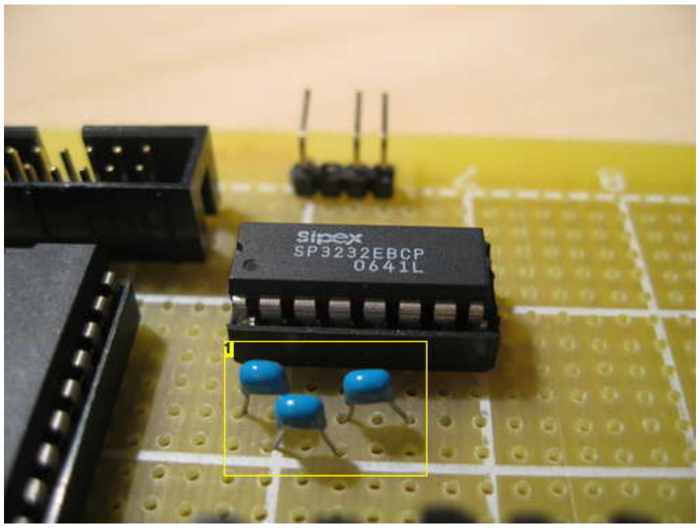


Image Notes
1. These capacitors helps the max232 bump the voltage up to rs232 levels.

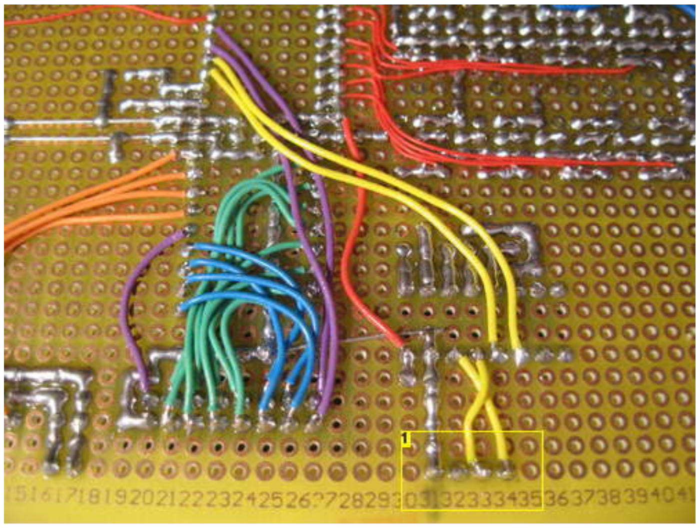


Image Notes
1. RS232 connector

Step 42: Build the controller: Make an RS-232 cable

To connect the LED cube to a serial port on your computer, you need to make a serial cable with a female D-Sub 9 pin connector.

Our employer deployed 70 Ethernet switches with management last year. With each switch comes an RS232 cable that is never used. We literally had a big pile of RS232 cable, so we decided to modify one of those.

On the LED cube, a 0.1" pin header is used, so the RS232 cable needs a new connector on the cube side.

We didn't have a 4 pin female 0.1" connector, so we used a 4 pin female PCB header instead.

The connector on the LED cube PCB has one pin removed, to visualize the directionality of the connector. The pin numbers goes from right to left.

Pinout of the RS232 connector:

1) GND (DSub9 pin 5)
2) Not connected
3) RX (DSub9 pin 3)
4) TX (DSub9 pin 2)

Follow these steps to make your own RS232 cable:

1) Cut of the connector at one end of the cable. If your cable has a female and a male connector, make sure to remove the male connector!

2) Strip away the outer sheath on the end where you removed the connector.

3) Strip all the wires inside.

4) Set your multimeter to continuity test mode. This makes the multimeter beep when the probes are connected. If your multimeter doesn't have this option, use the resistance mode. It should get close to 0 ohm when you connect the probes.

5) Connect one multimeter probe to the DSub9's pin 5, then probe all the wires until you the multimeter beeps. You have now identified the color of GND in your cable. Repeat for pin 2 and 3 (TX and RX).

6) Write down the colors you identified, then cut off the other wires.

7) Cut the three wires down to size, 30mm should do.

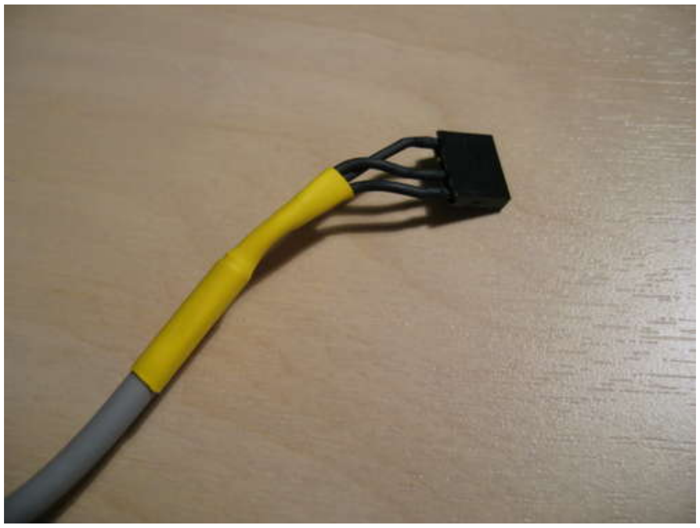
8) Pre-tin the wires to make soldering easier. Just apply heat and solder to the stripped wires.

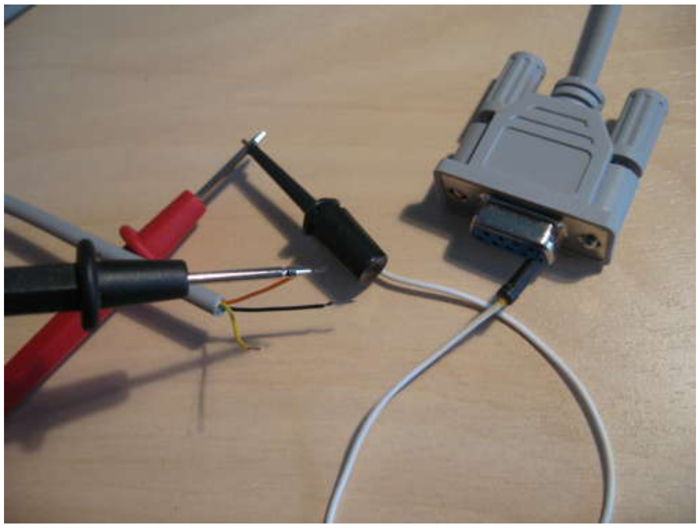
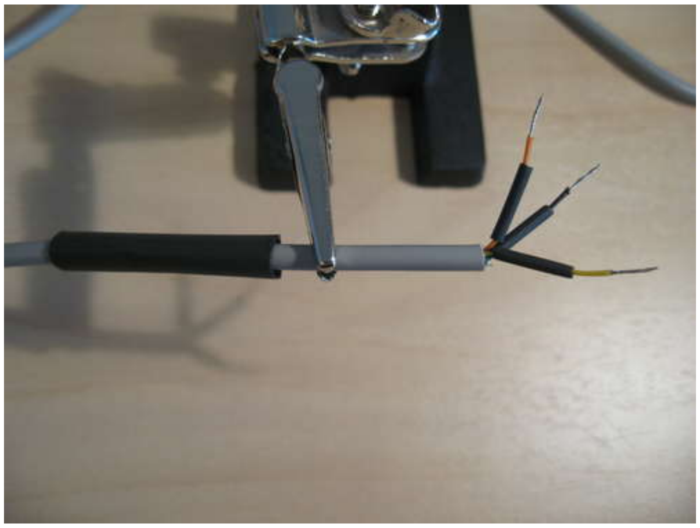
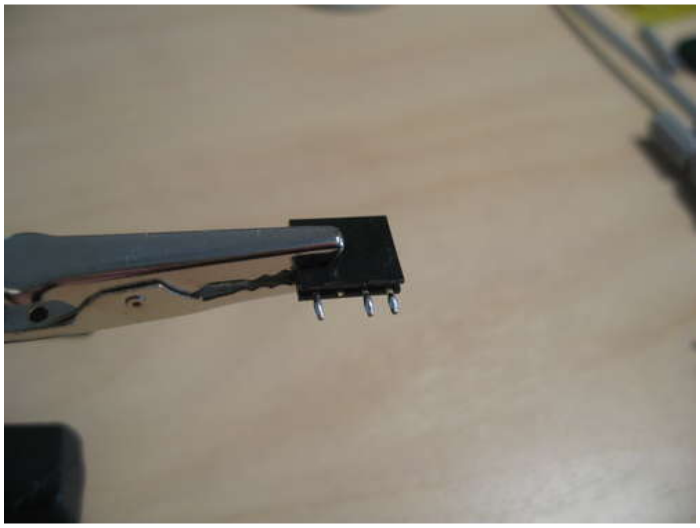
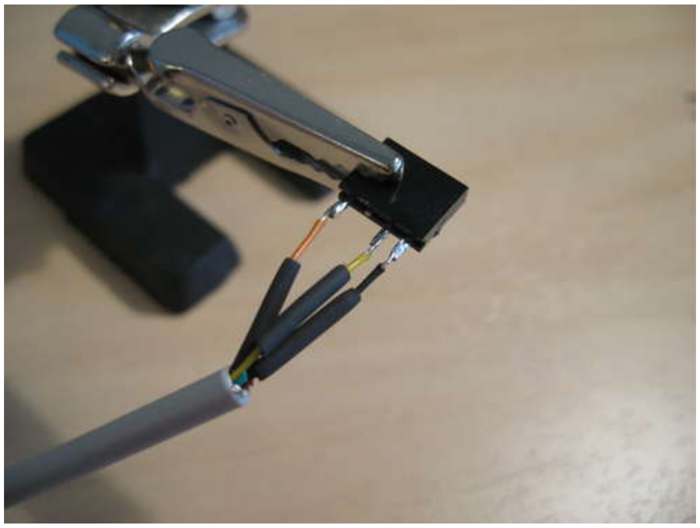
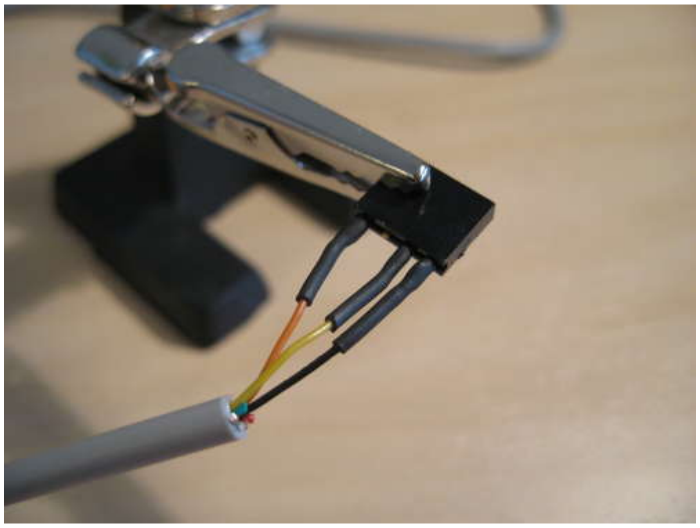
9) Slide a shrink tube over the cable. Slide three smaller shrink tubes over the individual wires.

10) Solder the wires to the connector.

11) Shrink the smaller tubes first, then the large one. If you use a lighter, don't hold the shrink tube above the flame, just hold it close to the side of the flame.

Don't make your cable based on the colors we used. Test the cable to find the correct colors.



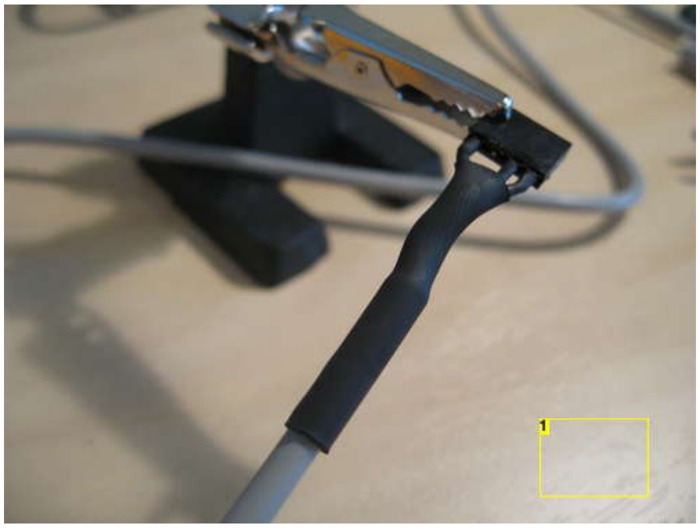


Image Notes
1. We managed to get the colors wrong on the first try. That's why the cable in the first picture has a yellow shrink tube ;)

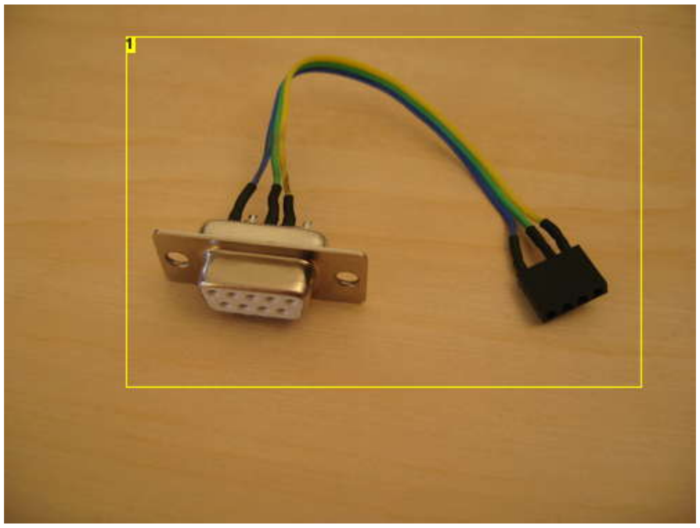


Image Notes
1. This is another way of doing it..

Step 43: Build the controller: Connect the boards

The two boards are connected by two cables:

- A ribbon cable for the DATA and Address BUS.
- A 2 wire cable for GND and VCC.

After connecting these two cables, your board is complete.

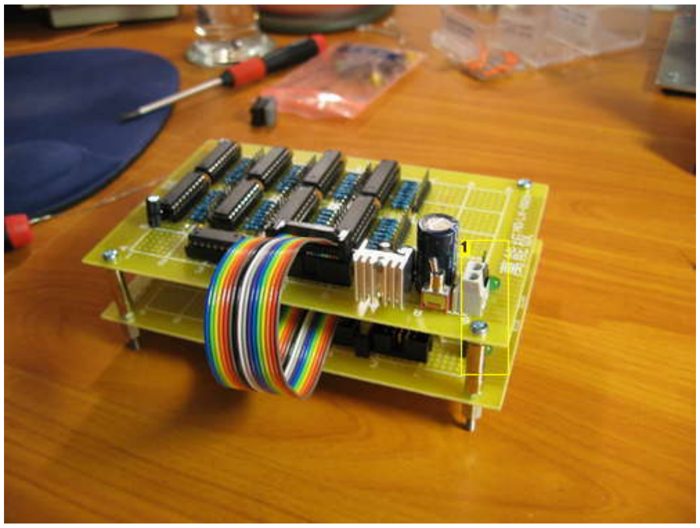
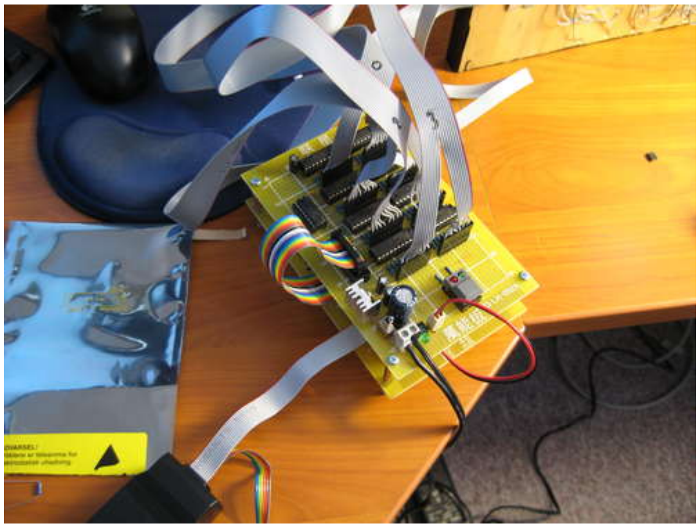
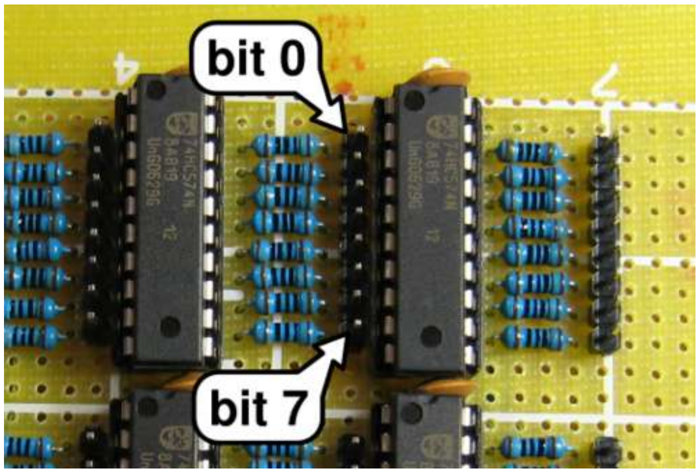


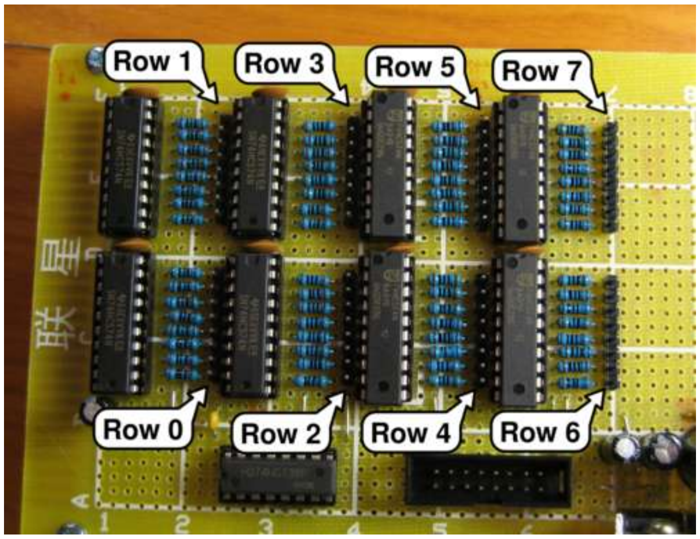
Image Notes
1. The GND/VCC cable connects between the two 2pin headers here.

Step 44: Build the controller: Connect the cube

Connect the ribbon cables according to the pin-outs shown in picture 2 and 3.
The ground layer ribbon cable connects to the pin header near the transistor array. If the cube is upside-down, just plug it in the other way.







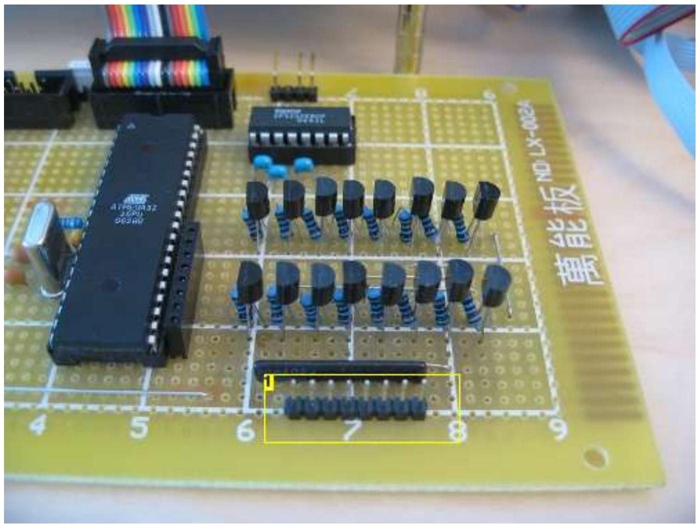


Image Notes
1. The ground layer ribbon cable connects here. Just connect it the other way if your LED cube is upside-down ;)

Step 45: Program the AVR: Set the fuse bits

The ATmega32 has two fuse bytes. These contain settings that have to be loaded before the CPU can start, like clock source and other stuff. You have to program your ATmega to use an external high speed crystal oscillator and disable JTAG.

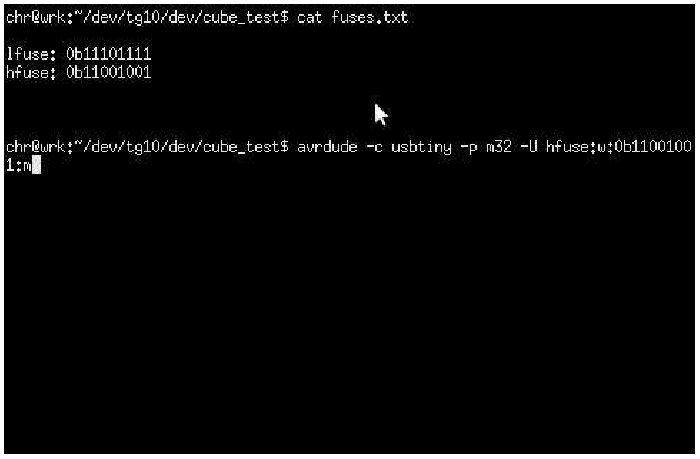
We set the lower fuse byte (lfuse) to 0b11101111, and the high fuse byte to 0b11001001. (0b means that everything after the b is in binary).

We used avrdude and USBtinyISP (http://www.ladyada.net/make/usbtinyisp/) to program our ATmega.

In all the following examples, we will be using an Ubuntu Linux computer. The commands should be identical if you run avrdude on Windows.

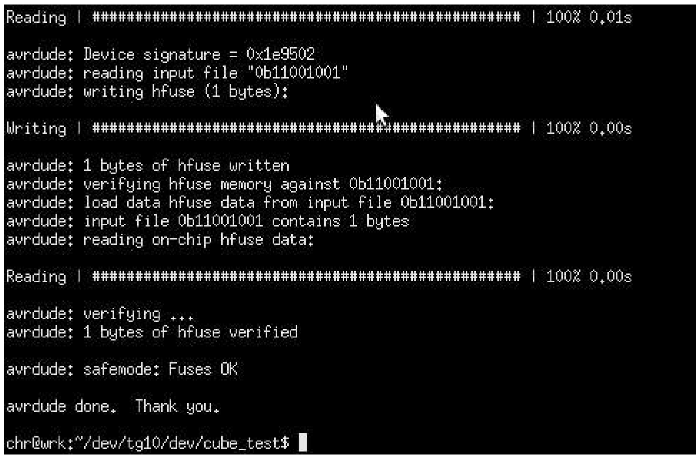
- avrdude -c usbtiny -p m32 -U lfuse:w:0b11101111:m
- avrdude -c usbtiny -p m32 -U hfuse:w:0b11001001:m

Warning: If you get this wrong, you could easily brick your ATmega! If you for example disable the reset button, you won't be able to re-program it. If you select the wrong clock source, it might not boot at all.

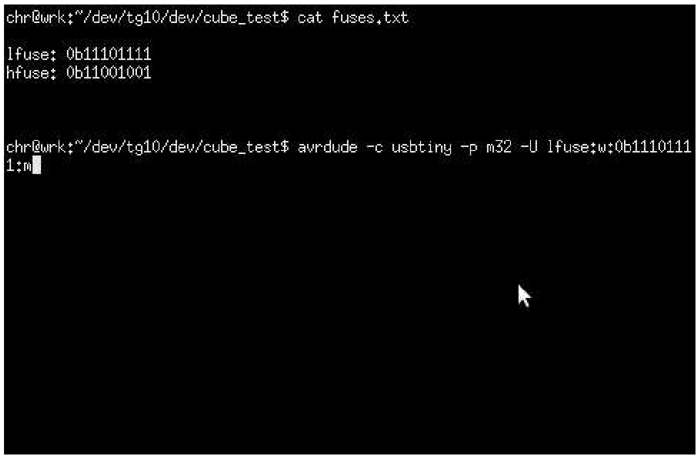


```
chr@wrk:~/dev/tg10/dev/cube_test$ cat fuses.txt

lfuse: 0b11101111
hfuse: 0b11001001


chr@wrk:~/dev/tg10/dev/cube_test$ avrdude -c usbtiny -p m32 -U hfuse:w:0b1100100
1:m
```

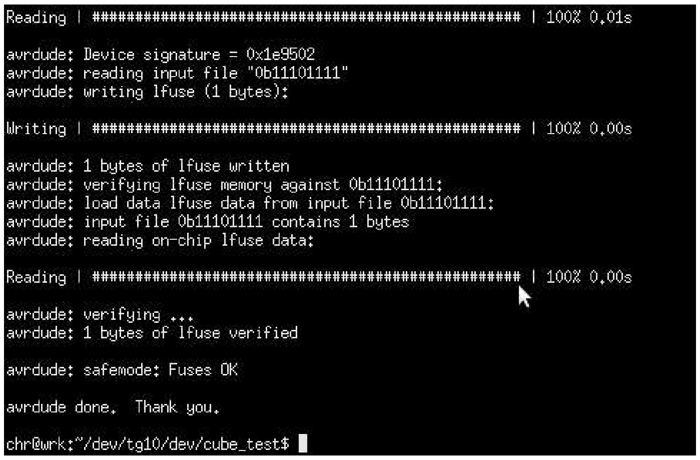
Image Notes
1. USBtinyISP



```
Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9502
avrdude: reading input file "0b11001001"
avrdude: writing hfuse (1 bytes):

Writing | ################################################## | 100% 0.00s

avrdude: 1 bytes of hfuse written
avrdude: verifying hfuse memory against 0b11001001:
avrdude: load data hfuse data from input file 0b11001001:
avrdude: input file 0b11001001 contains 1 bytes
avrdude: reading on-chip hfuse data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of hfuse verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.

chr@wrk:~/dev/tg10/dev/cube_test$
```



```
chr@wrk:~/dev/tg10/dev/cube_test$ cat fuses.txt

lfuse: 0b11101111
hfuse: 0b11001001


chr@wrk:~/dev/tg10/dev/cube_test$ avrdude -c usbtiny -p m32 -U lfuse:w:0b1110111
1:m
```



```
Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9502
avrdude: reading input file "0b11101111"
avrdude: writing lfuse (1 bytes):

Writing | ################################################## | 100% 0.00s

avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0b11101111:
avrdude: load data lfuse data from input file 0b11101111:
avrdude: input file 0b11101111 contains 1 bytes
avrdude: reading on-chip lfuse data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of lfuse verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.

chr@wrk:~/dev/tg10/dev/cube_test$
```

Step 46: Program the AVR with test code

Time to test if your brand new LED cube actually works!

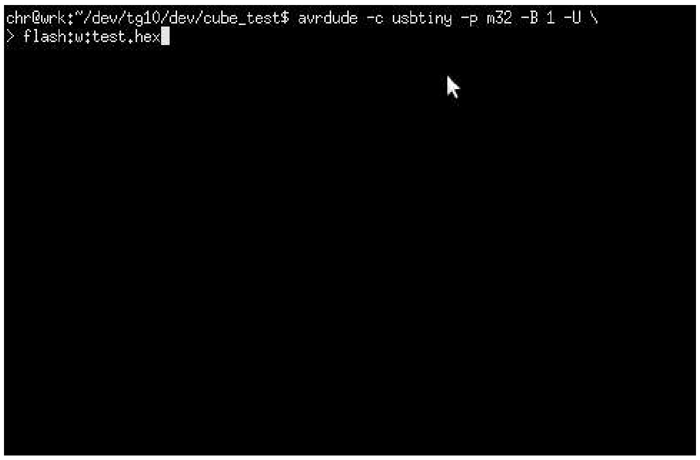
We have prepared a simple test program to check if all the LEDs work and if they are wired correctly.

You can download the firmware test.hex in this step, or download the source code and compile it yourself.

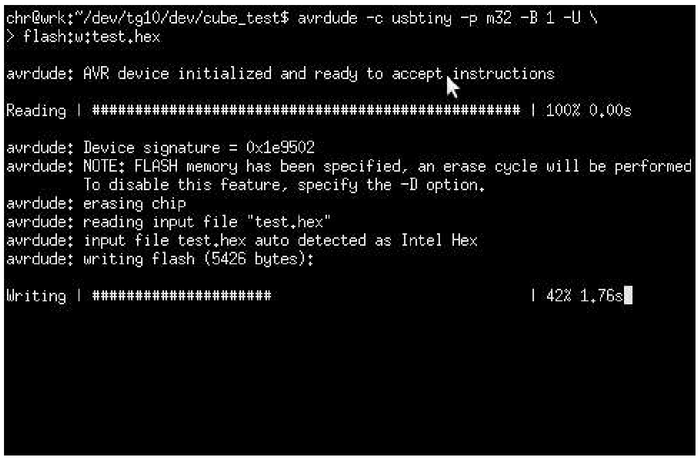
As in the previous step, we use avrdude for programming:

- avrdude -c usbtiny -p m32 -B 1 -U flash:w:test.hex

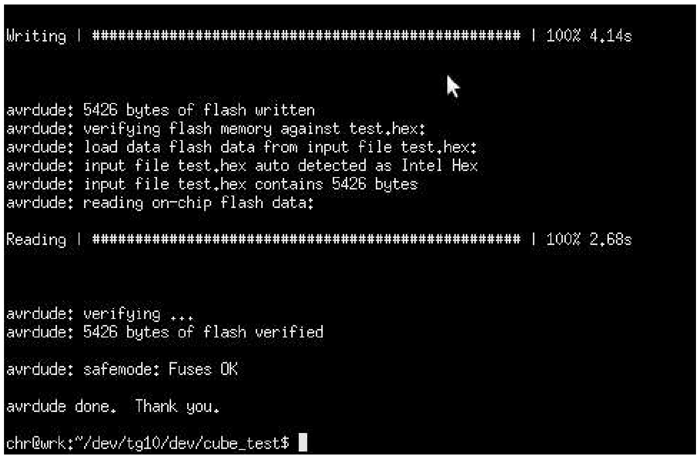
-c usbtiny specifies that we are using the USBtinyISP from Ladyada -p m32 specifies that the device is an ATmega32 -B 1 tells avrdude to work at a higher than default speed. -U flash:w:test.hex specifies that we are working on flash memory, in write mode, with the file test.hex.



```
chr@wrk:~/dev/tg10/dev/cube_test$ avrdude -c usbtiny -p m32 -B 1 -U \
> flash:w:test.hex
```



```
chr@wrk:~/dev/tg10/dev/cube_test$ avrdude -c usbtiny -p m32 -B 1 -U \
> flash:w:test.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9502
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "test.hex"
avrdude: input file test.hex auto detected as Intel Hex
avrdude: writing flash (5426 bytes):

Writing | #####################                              | 42% 1.76s
```



```
Writing | ################################################## | 100% 4.14s


avrdude: 5426 bytes of flash written
avrdude: verifying flash memory against test.hex:
avrdude: load data flash data from input file test.hex:
avrdude: input file test.hex auto detected as Intel Hex
avrdude: input file test.hex contains 5426 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 2.68s


avrdude: verifying ...
avrdude: 5426 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.

chr@wrk:~/dev/tg10/dev/cube_test$
```

File Downloads

test.hex (14 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'test.hex']

Step 47: Test the cube

The test code you programmed in the previous step will let you confirm that everything is wired up correctly.

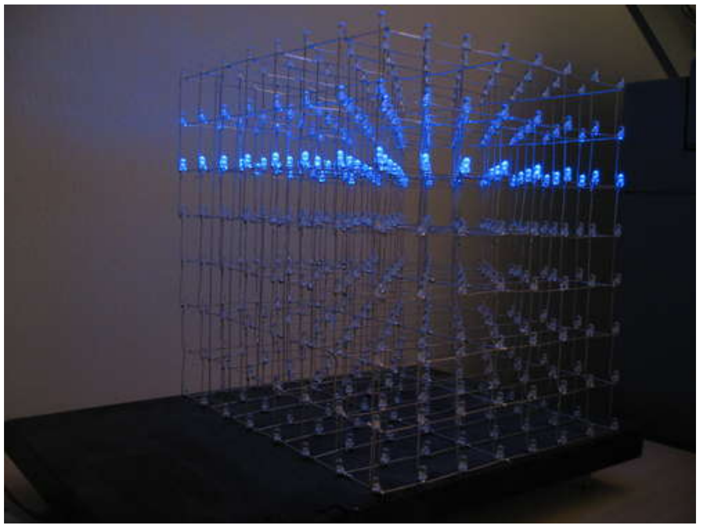
It will start by drawing a plane along one axis, then moving it along all 8 positions of that axis. (by plane we mean a flat surface, not an airplane :p) The test code will traverse a plane through all three axis.

After that, it will light the LEDs in a layer one by one, starting at the bottom layer.

If any of the layers or columns seem to light up in the wrong order, you have probably soldered the wrong wire to the wrong layer or column. We had one mistake in our cube ;)

If you find anything that is out of order, just de-solder the wires and solder them back in the right order. You could of course make a workaround in software, but that would eat CPU cycles every time the interrupt routine runs.

You can compare your cube to the test video below:



Step 48: Program the AVR with real code

So everything checked out in the test. It's time to program the ATmega with the real firmware!

For the most part, the process is the same as in the previous programming step. But in addition you have to program the EEPROM memory. The LED cube has a basic bitmap font stored in EEPROM, along with some other data.

Firmware is programmed using the same procedure as with the test code.

Firmware:

- avrdude -c usbtiny -p m32 -B 1 -U flash:w:main.hex

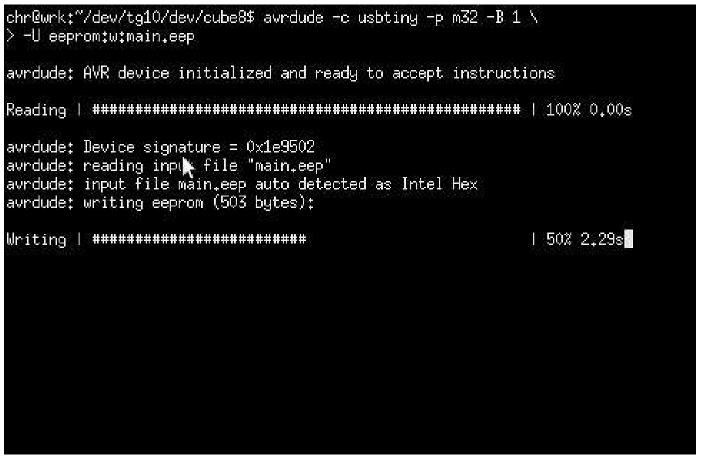
EEPROM:

- avrdude -c usbtiny -p m32 -B 1 -U eeprom:w:main.eep

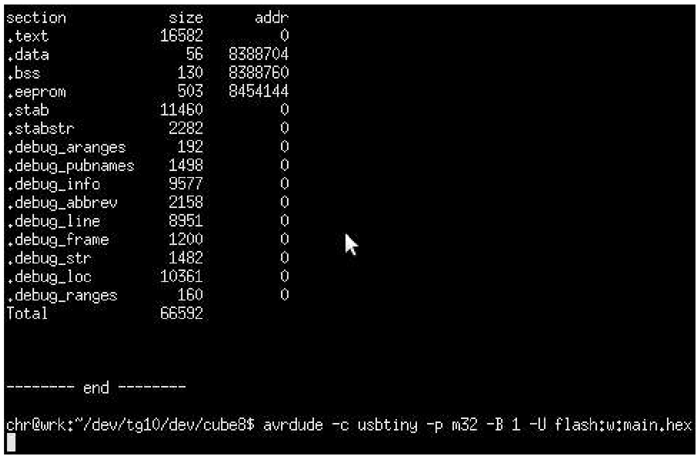
-U eeprom:w:main.eep specifies that we are accessing EEPROM memory, in write mode. Avr-gcc puts all the EEPROM data in main.eep.

If you don't want to play around with the code, your LED cube is finished at this point. But we recommend that you spend some time on the software side of things as well. That's at least as much fun as the hardware!

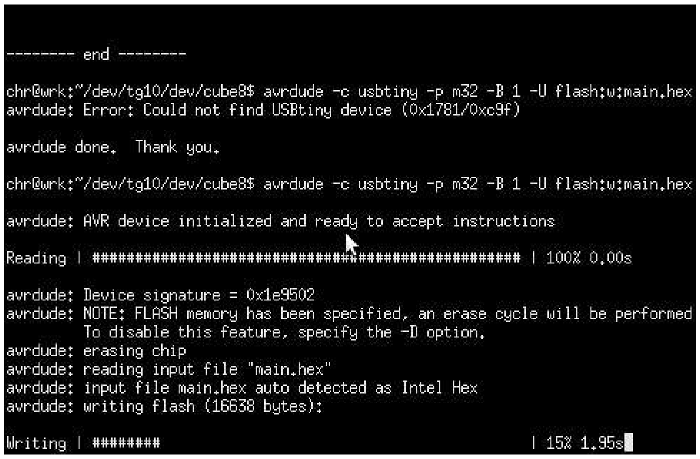
If you download the binary files, you have to change the filenames in the commands to the name of the files you downloaded. If you compile from source the name is main.hex and main.eep.



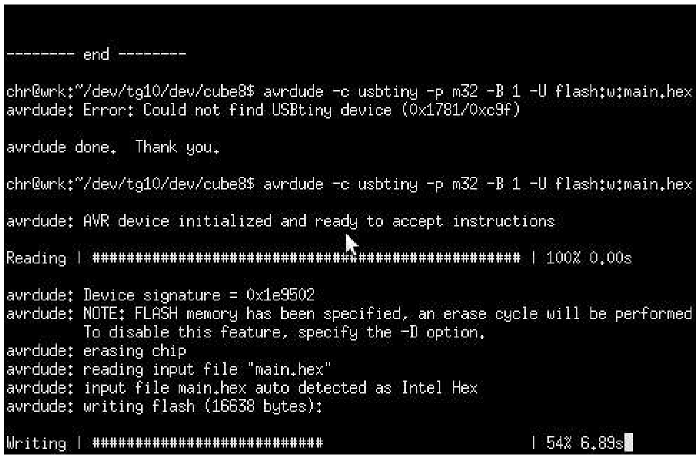
```
chr@wrk:~/dev/tg10/dev/cube8$ avrdude -c usbtiny -p m32 -B 1 \
> -U eeprom:w:main.eep

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9502
avrdude: reading input file "main.eep"
avrdude: input file main.eep auto detected as Intel Hex
avrdude: writing eeprom (503 bytes):

Writing | #########################                          | 50% 2.29s
```



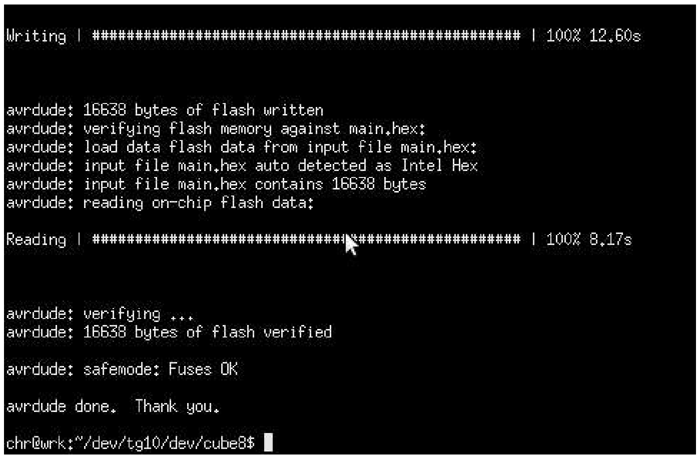
```
section             size      addr
.text              16582         0
.data                 56   8388704
.bss                 130   8388760
.eeprom              503   8454144
.stab              11460         0
.stabstr            2282         0
.debug_aranges       192         0
.debug_pubnames     1498         0
.debug_info         9577         0
.debug_abbrev       2158         0
.debug_line         8951         0
.debug_frame        1200         0
.debug_str          1482         0
.debug_loc         10361         0
.debug_ranges        160         0
Total              66592


-------- end --------

chr@wrk:~/dev/tg10/dev/cube8$ avrdude -c usbtiny -p m32 -B 1 -U flash:w:main.hex
```



```
-------- end --------

chr@wrk:~/dev/tg10/dev/cube8$ avrdude -c usbtiny -p m32 -B 1 -U flash:w:main.hex
avrdude: Error: Could not find USBtiny device (0x1781/0xc9f)

avrdude done.  Thank you.

chr@wrk:~/dev/tg10/dev/cube8$ avrdude -c usbtiny -p m32 -B 1 -U flash:w:main.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9502
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: input file main.hex auto detected as Intel Hex
avrdude: writing flash (16638 bytes):

Writing | ########                                           | 15% 1.95s
```



```
-------- end --------

chr@wrk:~/dev/tg10/dev/cube8$ avrdude -c usbtiny -p m32 -B 1 -U flash:w:main.hex
avrdude: Error: Could not find USBtiny device (0x1781/0xc9f)

avrdude done.  Thank you.

chr@wrk:~/dev/tg10/dev/cube8$ avrdude -c usbtiny -p m32 -B 1 -U flash:w:main.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9502
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: input file main.hex auto detected as Intel Hex
avrdude: writing flash (16638 bytes):

Writing | ###########################                        | 54% 6.89s
```



```
Writing | ################################################## | 100% 12.60s



avrdude: 16638 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex auto detected as Intel Hex
avrdude: input file main.hex contains 16638 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 8.17s



avrdude: verifying ...
avrdude: 16638 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.

chr@wrk:~/dev/tg10/dev/cube8$
```

File Downloads

ledcube_8x8x8_eeprom.eep (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'ledcube_8x8x8_eeprom.eep']

ledcube_8x8x8.hex (46 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'ledcube_8x8x8.hex']

Step 49: Software: Introduction

The software is written in C and compiled with the open source compiler avr-gcc. This is the main reason we use Atmel AVR micro controllers. The PIC series from Microchip is also a nice choice, but most of the C compilers cost money, and the free versions have limitations on code size.

The AVR route is much more hassle free. Just apt-get install the avr-gcc compiler, and you're in business.

The software on the AVR consists of two main components, the cube interrupt routine and effect code for making fancy animations.

When we finally finished soldering, we thought this would be the easy part. But it turns out that making animations in monochrome at low resolutions is harder than it sounds.

If the display had a higher resolution and more colors, we could have used sin() and cos() functions and all that to make fancy eye candy. With two colors (on and off) and low resolution, we have to use a lot of if() and for() to make anything meaningful.

In the next few steps, we will take you on a tour of some of the animations we made and how they work. Our goal is to give you an understanding of how you can make animations, and inspire you to create your own! If you do, please post a video in the comments!

File Downloads

ledcube_8x8x8-v0.1.2.tar.gz (20 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'ledcube_8x8x8-v0.1.2.tar.gz']

Step 50: Software: How it works

As mentioned in the previous step, the software consists of two pars. The interrupt routine and the effect code.

Communication between these two happens via a voxel array. This array has a bit for every LED in the LED cube. We will refer to this as the cube array or cube buffer from now on.

The cube array is made of 8x8 bytes. Since each byte is 8 bits, this gives us a buffer that is 8 voxels wide, 8 woxels high and 8 voxels deep (1 byte deep).

volatile unsigned char cube[8][8];

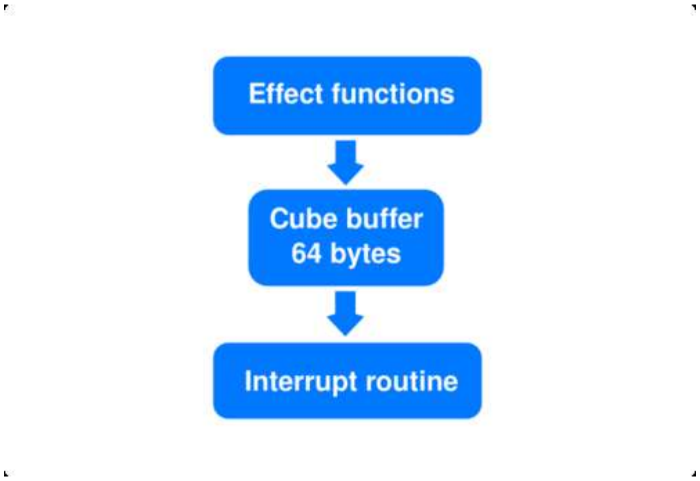
The interrupt routine reads from the cube array at given intervals and displays the information on the LED cube.

The effect functions writes the desired LED statuses to this array.

We did not use any synchronization or double buffering, since there is only one producer (either the effects currently running, or input from RS232) and one consumer (the interrupt-code that updates the cube). This means that some voxels could be from the next or previous "frame", but this is not a problem, since the frame rate is so high.

When working with micro controllers, code size is critical. To save code size and programming work, and to make the code easier to read, we have tried to write re-usable code as often as possible.

The LED cube code has a base of low level drawing functions that are used by the higher level effect functions. The draw functions can be found in draw.c. Draw functions include everything from setting or clearing a single voxel to drawing lines and wireframe boxes.



Step 51: Software: IO initialization

The first thing the ATmega does after boot, is to call the ioinit() function.

This function sets up IO ports, timers, interrupts and serial communications.
All IO ports on the ATmega are bi-directional. They can be used either as an input or an output. We configure everything as outputs, except the IO pins where the two buttons are connected. The RX pin for the serial line automatically becomes an input when USART RX is enabled.

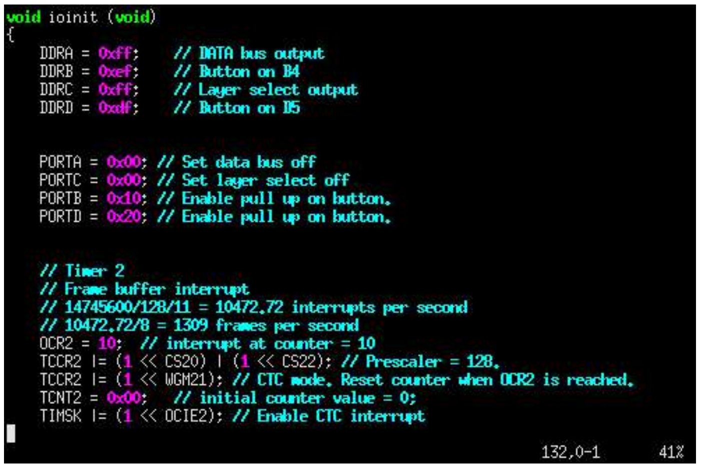
1) DDRx sets the data direction of the IO pins. (Data Direction Register). 1 means output, 0 means input.

2) After directionality has been configured, we set all outputs to 0 to avid any blinking LEDs etc before the interrupt has started.

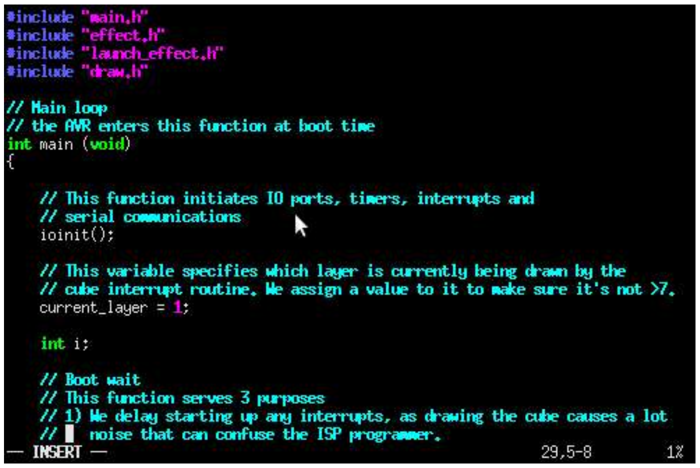
3) For pins configured as inputs, the PORTx bit changes its function. Setting a 1 in the PORTx register bit enables an internal pull up resistor. The port is pulled up to VCC. The buttons are connected between the port and GND. When a button is pressed the corresponding PINx bit reads a logic 0.

4) Timer 2 is configured and a timer interrupt enabled. This is covered in a separate step.

5) Serial communications is configured and enabled. This is also covered in a separate step.



```
void ioinit (void)
{
    DDRA = 0xff;    // DATA bus output
    DDRB = 0xef;    // Button on B4
    DDRC = 0xff;    // Layer select output
    DDRD = 0xdf;    // Button on D5


    PORTA = 0x00; // Set data bus off
    PORTC = 0x00; // Set layer select off
    PORTB = 0x10; // Enable pull up on button.
    PORTD = 0x20; // Enable pull up on button.


    // Timer 2
    // Frame buffer interrupt
    // 14745600/128/11 = 10472.72 interrupts per second
    // 10472.72/8 = 1309 frames per second
    OCR2 = 10;  // interrupt at counter = 10
    TCCR2 |= (1 << CS20) | (1 << CS22); // Prescaler = 128.
    TCCR2 |= (1 << WGM21); // CTC mode. Reset counter when OCR2 is reached.
    TCNT2 = 0x00;   // initial counter value = 0;
    TIMSK |= (1 << OCIE2); // Enable CTC interrupt

                                                      132,0-1       41%
```



```
#include "main.h"
#include "effect.h"
#include "launch_effect.h"
#include "draw.h"

// Main loop
// the AVR enters this function at boot time
int main (void)
{

    // This function initiates IO ports, timers, interrupts and
    // serial communications
    ioinit();

    // This variable specifies which layer is currently being drawn by the
    // cube interrupt routine. We assign a value to it to make sure it's not >7.
    current_layer = 1;

    int i;

    // Boot wait
    // This function serves 3 purposes
    // 1) We delay starting up any interrupts, as drawing the cube causes a lot
    //    noise that can confuse the ISP programmer.
-- INSERT --                                          29,5-8        1%
```

Step 52: Software: Mode selection and random seed

When we first started writing effects and debugging them, we noticed that the functions using random numbers displayed the exact same animations every time. It was random alright, but the same random sequence every time. Turns out the random number generator in the ATmega needs to be seeded with a random number to create true random numbers.

We wrote a small function called bootwait(). This function serves two purposes.

1) Create a random seed. 2) Listen for button presses to select mode of operation.

It does the following:

1) Set counter x to 0.

2) Start an infinite loop, while(1).

3) Increment counter x by one.

4) Use x as a random seed.

5) Delay for a while and set red status led on.

6) Check for button presses. If the main button is pressed, the function returns 1. If the PGM button is pressed it returnes 2. The return statements exits the function thus ending the infinite loop.

7) Delay again and set green led on.

8) Check for button presses again.

9) Loop forever until a button is pressed.

The loop loops very fast, so the probability that you will stop it at the same value of x two times in a row is very remote. This is a very simple but effective way to get a good random seed.

Bootwait() is called from the main() function and its return value assigned to the variable i.

If i == 1, the main loop starts a loop that displays effects generated by the ATmega. If i == 2, it enters into RS232 mode and waits for data from a computer.

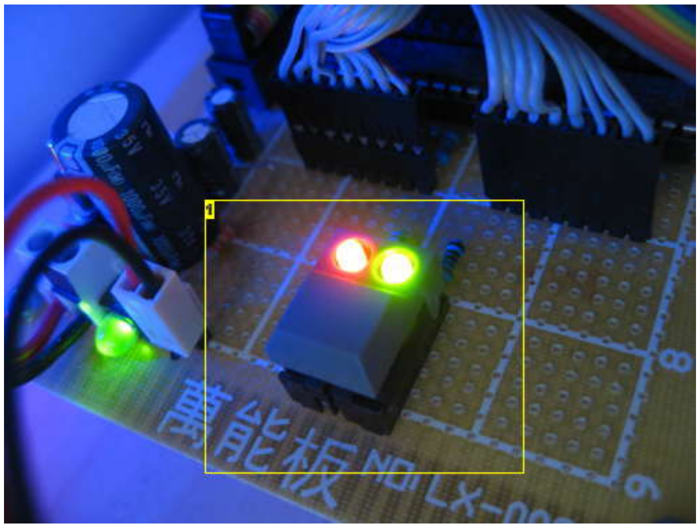


Image Notes
1. blink blink blink

Step 53: Software: Interrupt routine

The heart of the LED cube code is the interrupt routine.

Every time this interrupt runs, the cube is cleared, data for the new layer is loaded onto the latch array, and the new layer is switched on. This remains on until the next time the interrupt runs, where the cube is cleared again, data for the next layer is loaded onto the latch array, and the next layer is switched on.

The ATmega32 has 3 timer/counters. These can be set to count continuously and trigger an interrupt routine every time they reach a certain number. The counter is reset when the interrupt routine is called.

We use Timer2 with a prescaler of 128 and an Output Compare value of 10. This means that the counter is incremented by 1 for every 128th cpu cycle. When Timer2 reaches 10, it is reset to 0 and the interrupt routine is called. With a cpu frequency of 14745600 Hz, 128 prescaler and output compare of 10, the interrupt routine is called every 1408th CPU cycle (128*11) or 10472.7 times per second. It displays one layer at a time, so it takes 8 runs of the interrupt to draw the entire cube once. This gives us a refresh rate of 1309 FPS (10472.7/8). At this refresh rate, the LED cube is 100% flicker free. Some might say that 1300 FPS is overkill, but the interrupt routine is quite efficient. At this high refresh rate, it only uses about 21% of the CPU time. We can measure this by attaching an oscilloscope to the output enable line (OE). This is pulled high at the start of each interrupt and low at the end, so it gives a pretty good indication of the time spent inside the interrupt routine.

Before any timed interrupts can start, we have to set up the Timer 2. This is done in the ioinit() function.

TCCR2 (Timer Counter Control Register 2) is an 8 bit register that contains settings for the timer clock source and mode of operation. We select a clock source with a 1/128 prescaler. This means that Timer/counter 2 is incrementet by 1 every 128th CPU cycle.

We set it to CTC mode. (Clear on Timer Compare). In this mode, the counter value TCNT2 is continuously compared to OCR2 (Output Compare Register 2). Every time TCNT2 reaches the value stored in OCR2, it is reset to 0 and starts counting from from 0. At the same time, an interrupt is triggered and the interrupt routine is called.

For every run of the interrupt, the following takes place:

1) All the layer transistors are switched off.

2) Output enable (OE) is pulled high to disable output from the latch array.

3) A loop runs through i = 0-7. For every pass a byte is outputed on the DATA bus and the i+1 is outputed on the address bus. We add the +1 because the 74HC138 has active low outputs and the 74HC574 clock line is triggered on the rising edge (transition from low to high).

4) Output enable is pulled low to enable output fro the latch array again.

5) The transistor for the current layer is switched on.

6) current_layer is incremented or reset to 0 if it moves beyond 7.

That's it. The interrupt routine is quite simple. I'm sure there are some optimizations we could have used, but not without compromising human readability of the code. For the purpose of this instructable, we think readability is a reasonable trade-off for a slight increase in performance.

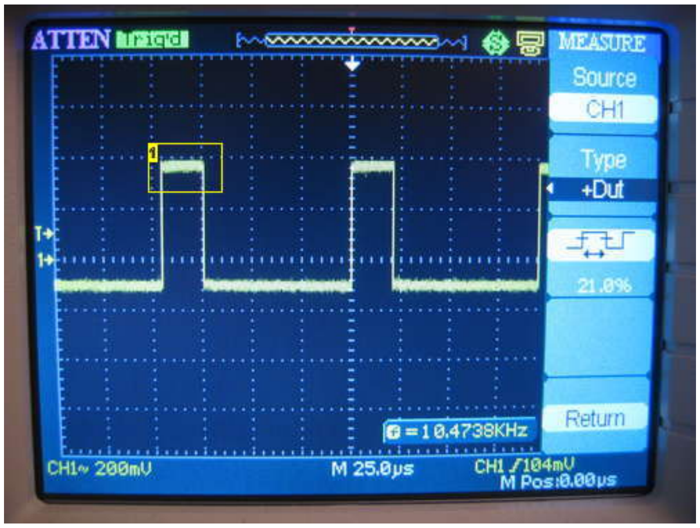


Image Notes
1. The interrupt routine pulls Output Enable high while running to disable the output of the latch array.

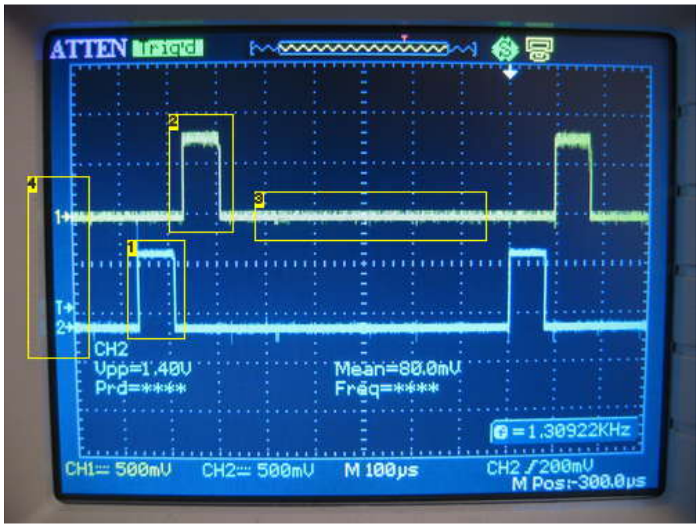


Image Notes
1. Layer 0 is on
2. Layer 1 is on
3. My oscilloscope doesn't have 8 channels, so I can only show the first two layers.
4. Output from the layer transistor lines.

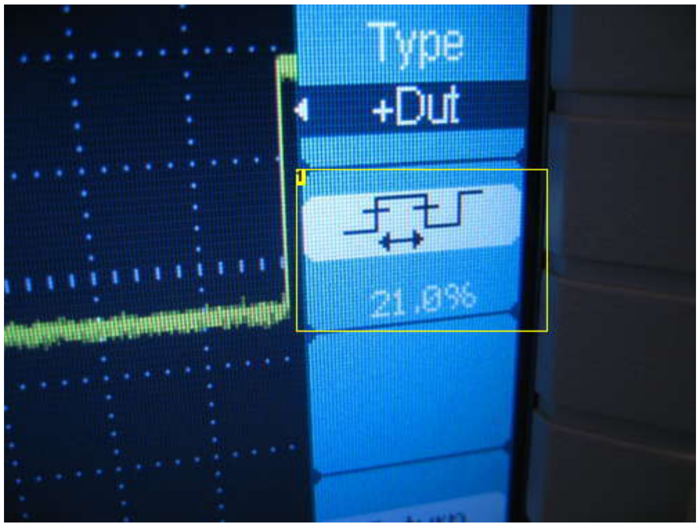


Image Notes
1. The interrupt routine runs roughly 21% of the time. This leaves the remaining 79% for effect code!

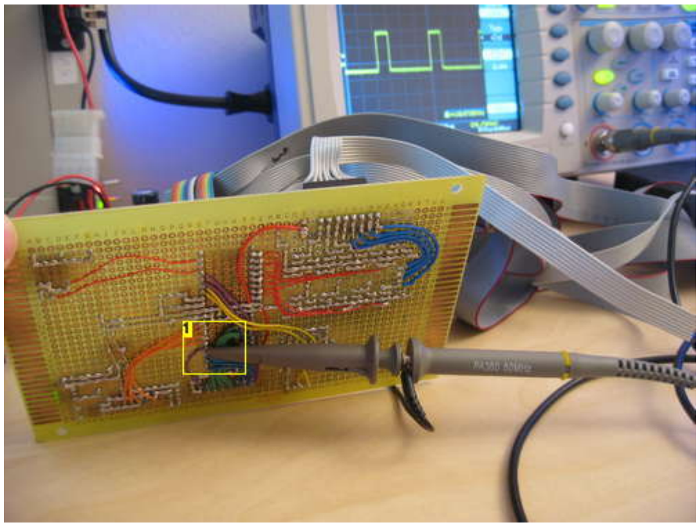
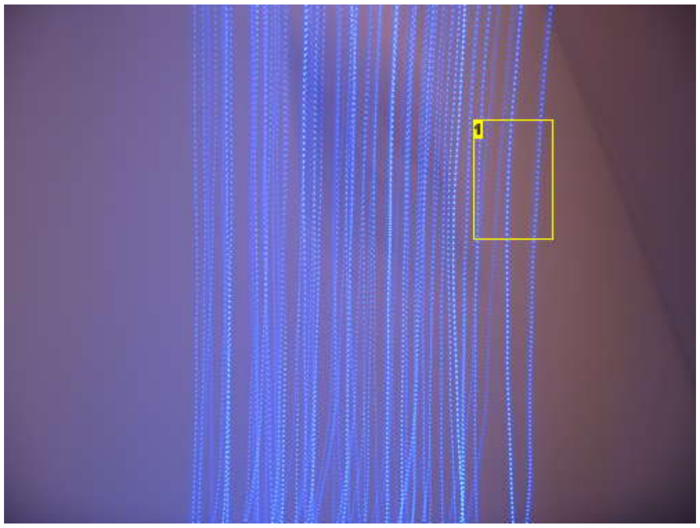
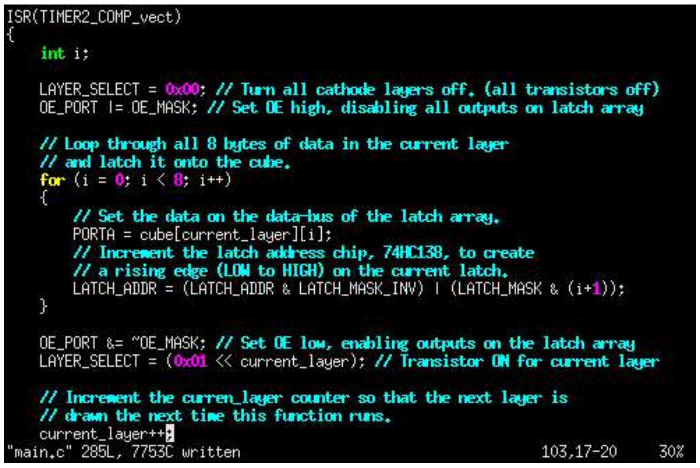


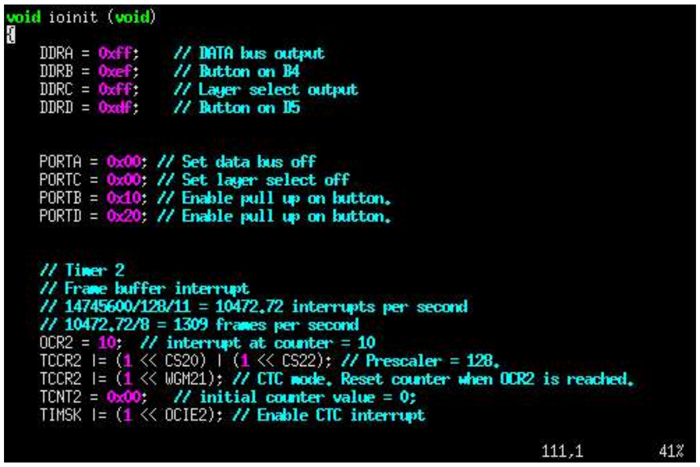
Image Notes
1. Probing output enable





```
ISR(TIMER2_COMP_vect)
{
    int i;

    LAYER_SELECT = 0x00; // Turn all cathode layers off. (all transistors off)
    OE_PORT |= OE_MASK; // Set OE high, disabling all outputs on latch array

    // Loop through all 8 bytes of data in the current layer
    // and latch it onto the cube.
    for (i = 0; i < 8; i++)
    {
        // Set the data on the data-bus of the latch array.
        PORTA = cube[current_layer][i];
        // Increment the latch address chip, 74HC138, to create
        // a rising edge (LOW to HIGH) on the current latch.
        LATCH_ADDR = (LATCH_ADDR & LATCH_MASK_INV) | (LATCH_MASK & (i+1));
    }

    OE_PORT &= ~OE_MASK; // Set OE low, enabling outputs on the latch array
    LAYER_SELECT = (0x01 << current_layer); // Transistor ON for current layer

    // Increment the curren_layer counter so that the next layer is
    // drawn the next time this function runs.
    current_layer++;
"main.c" 285L, 7753C written                                    103,17-20      30%
```

Image Notes
1. If you move the camera when taking a picture of any POV gadget, you can see the POV action. I moved the camera very fast in this picture. Yet you can barely see the effect. With a lower refresh rate, the dots and spaces would be longer



```
void ioinit (void)
{
    DDRA = 0xff;    // DATA bus output
    DDRB = 0xef;    // Button on B4
    DDRC = 0xff;    // Layer select output
    DDRD = 0xdf;    // Button on D5


    PORTA = 0x00; // Set data bus off
    PORTC = 0x00; // Set layer select off
    PORTB = 0x10; // Enable pull up on button.
    PORTD = 0x20; // Enable pull up on button.


    // Timer 2
    // Frame buffer interrupt
    // 14745600/128/11 = 10472.72 interrupts per second
    // 10472.72/8 = 1309 frames per second
    OCR2 = 10;  // interrupt at counter = 10
    TCCR2 |= (1 << CS20) | (1 << CS22); // Prescaler = 128.
    TCCR2 |= (1 << WGM21); // CTC mode. Reset counter when OCR2 is reached.
    TCNT2 = 0x00;   // initial counter value = 0;
    TIMSK |= (1 << OCIE2); // Enable CTC interrupt
                                                    111,1          41%
```

Step 54: Software: Low level functions

We have made a small library of low level graphic functions.
There are three main reasons for doing this.

Memory footprint

The easiest way to address each voxel would be through a three dimensional buffer array. Like this:

unsigned char cube[x][y][z]; (char means an 8 bit number, unsigned means that it's range is from 0 to 255. signed is -128 to +127)

Within this array each voxel would be represented by an integer, where 0 is off and 1 is on. In fact, you could use the entire integer and have 256 different brightness levels. We actually tried this first, but it turned out that our eBay LEDs had very little change in brightness in relation to duty cycle. The effect wasn't noticeable enough to be worth the trouble. We went for a monochrome solution. On and off.

With a monochrome cube and a three dimensional buffer, we would be wasting 7/8 of the memory used. The smallest amount of memory you can allocate is one byte (8 bits), and you only need 1 bit to represent on and off. 7 bits for each voxel would be wasted. 512*(7/8) = 448 bytes of wasted memory. Memory is scarce on micro controllers, so this is a sub-optimal solution.

Instead, we created a buffer that looks like this:

unsigned char cube[z][y];

In this buffer the X axis is represented within each of the bytes in the buffer array. This can be quite confusing to work with, which brings us to the second reason for making a library of low level drawing functions:

Code readability

Setting a voxel with the coordinates x=4, y=3, z=5 will require the following code:

cube[5][3] |= (0x01 << 4);

You can see how this could lead to some serious head scratching when trying to debug your effect code ;)

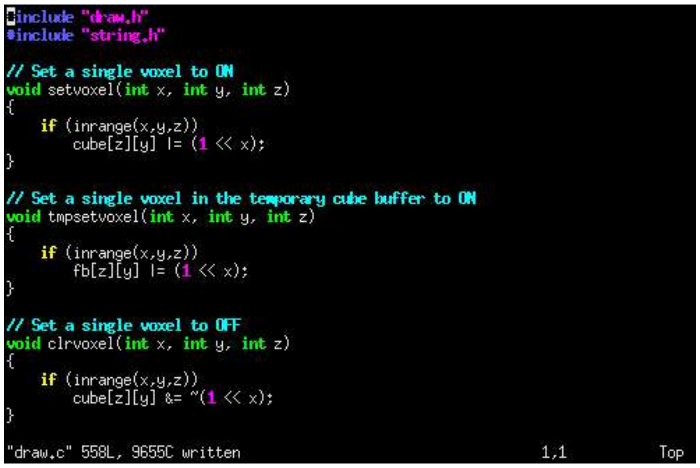
In draw.c we have made a bunch of functions that takes x,y,z as arguments and does this magic for you.

Setting the same voxel as in the example above is done with setvoxel(4,3,5), which is _a lot_ easier to read!

draw.c contains many more functions like this. Line drawing, plane drawing, box drawing, filling etc. Have a look in draw.c and familiarize yourself with the different functions.

Reusable code and code size

As you can see in draw.c, some of the functions are quite large. Writing that code over and over again inside effect functions would take up a lot of program memory. We only have 32 KB to work with. Its also boring to write the same code over and over again ;)


```
#include "draw.h"
#include "string.h"

// Set a single voxel to ON
void setvoxel(int x, int y, int z)
{
    if (inrange(x,y,z))
        cube[z][y] |= (1 << x);
}

// Set a single voxel in the temporary cube buffer to ON
void tmpsetvoxel(int x, int y, int z)
{
    if (inrange(x,y,z))
        fb[z][y] |= (1 << x);
}

// Set a single voxel to OFF
void clrvoxel(int x, int y, int z)
{
    if (inrange(x,y,z))
        cube[z][y] &= ~(1 << x);
}
"draw.c" 558L, 9655C written                          1,1           Top
```

Step 55: Software: Cube virtual space

Now that we have a cube buffer and a nice little collection of low level draw functions to populate it, we need to agree on which ways is what, and what is up and what is down ;)

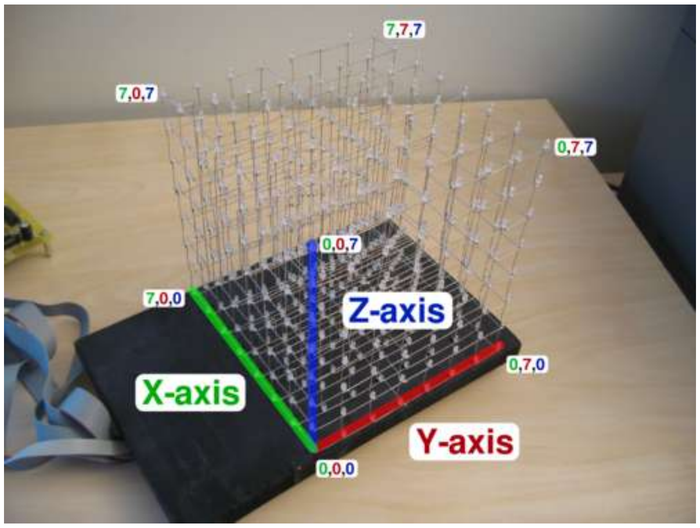
From now on, the native position of the LED cube will be with the cables coming out to the left.

In this orientation, the Y axis goes from left to right. The X axis goes from front to back. The Z axis goes from bottom to top.

Coordinates in this instructable is always represented as x,y,z.

Position 0,0,0 is the bottom left front corner. Position 7,7,7 is the top right back corner.

Why did we use the Y axis for left/right and X for back/front? Shouldn't it be the other way around? Yes, we think so too. We designed the the LED cube to be viewed from the "front" with the cables coming out the back. However, this was quite impractical when having the LED cube on the desk, it was more practical to have the cables coming out the side, and having cube and controller side by side. All the effect functions are designed to be viewed from this orientation.



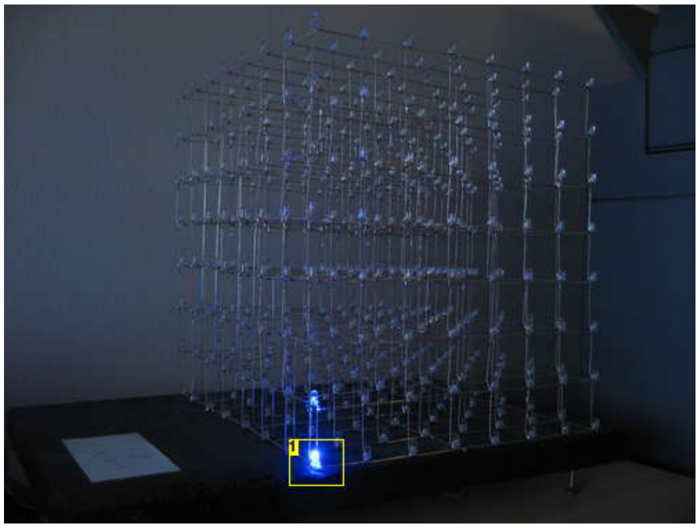


Image Notes
1. 0,0,0

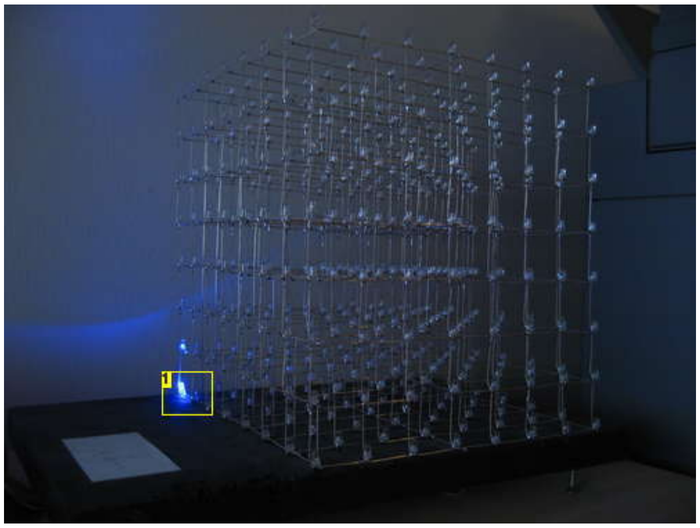


Image Notes
1. 7,0,0

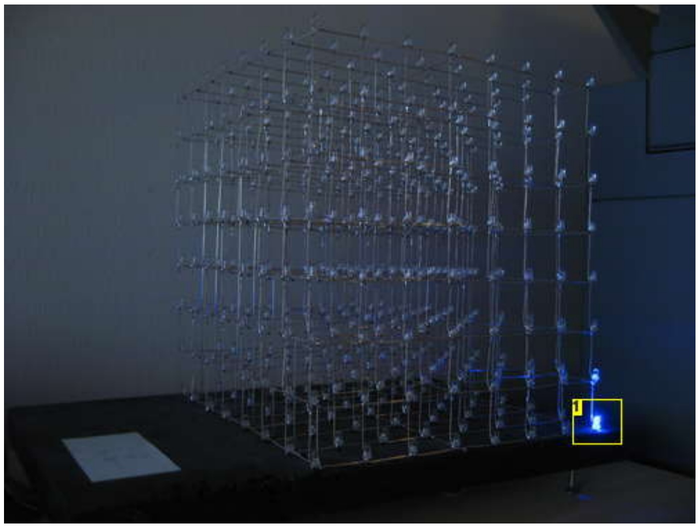


Image Notes
1. 0,7,0

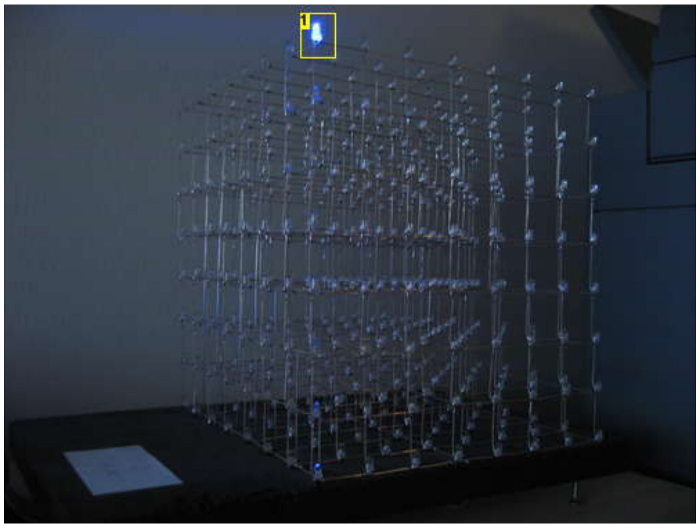


Image Notes
1. 0,0,7

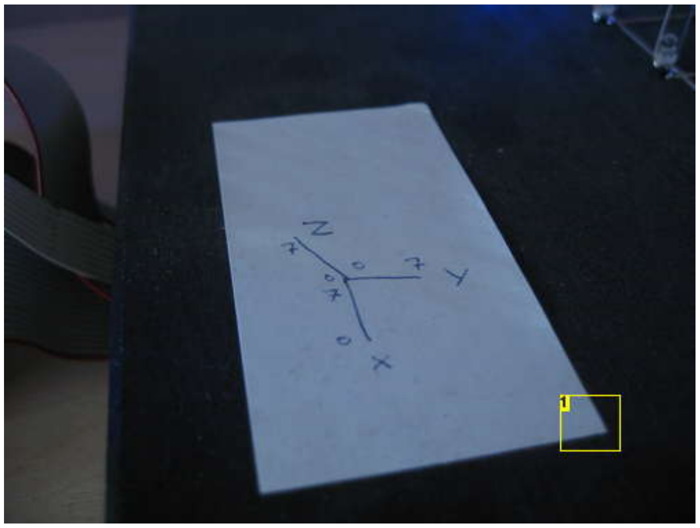


Image Notes
1. Made a little cheat note while programming ;)

Step 56: Software: Effect launcher

We wanted an easy way to run the effects in a fixed order or a random order. The solution was to create an effect launcher function.

launch_effect.c contains the function launch_effect (int effect).

Inside the function there is a switch() statement which calls the appropriate effect functions based on the number launch_effect() was called with.

In launch_effect.h EFFECTS_TOTAL is defined. We set it one number higher than the highest number inside the switch() statement.

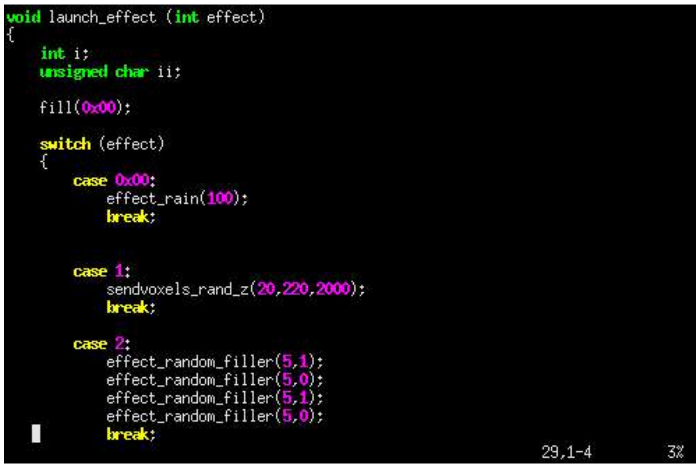
Launching the effects one by one is now a simple matter of just looping through the numbers and calling launch_effect(), like this:

```
while(1)
for (i=0; i < EFFECTS_TOTAL; i++)
{

launch_effect(i);

}

}
```

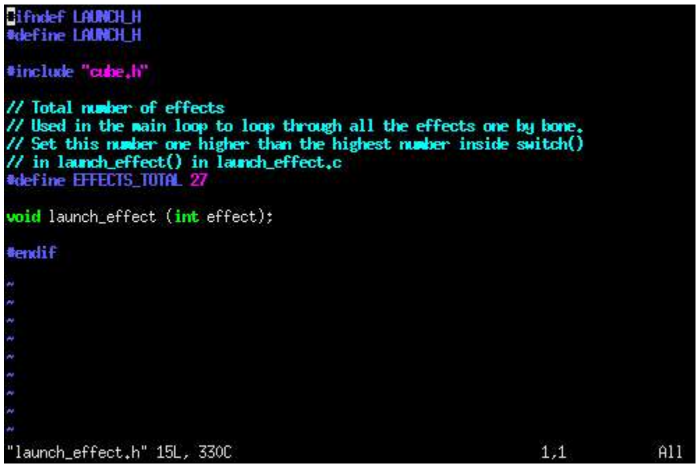
This code will loop through all the effects in incremental order forever.
If you want the cube to display effects in a random order, just use the following code:

```
while (1)
{
launch_effect(rand()%EFFECTS_TOTAL);

}
```

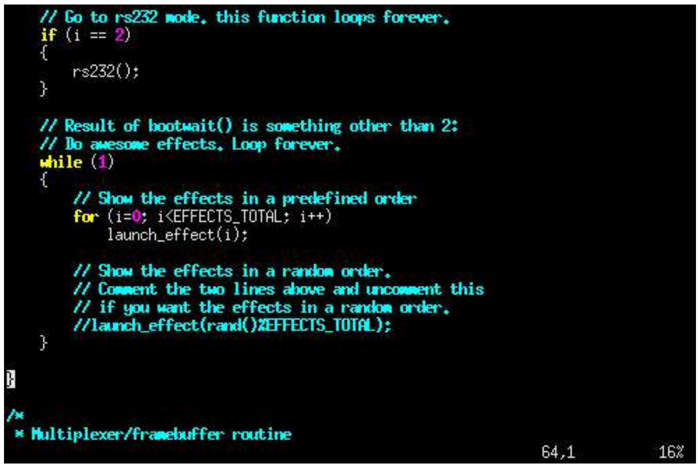
The %EFFECTS_TOTAL after rand() keeps the random value between 0 and EFFECTS_TOTAL-1.



```
void launch_effect (int effect)
{
    int i;
    unsigned char ii;

    fill(0x00);

    switch (effect)
    {
        case 0x00:
            effect_rain(100);
            break;


        case 1:
            sendvoxels_rand_z(20,220,2000);
            break;

        case 2:
            effect_random_filler(5,1);
            effect_random_filler(5,0);
            effect_random_filler(5,1);
            effect_random_filler(5,0);
            break;
                                                    29,1-4         3%
```



```
#ifndef LAUNCH_H
#define LAUNCH_H

#include "cube.h"

// Total number of effects
// Used in the main loop to loop through all the effects one by bone.
// Set this number one higher than the highest number inside switch()
// in launch_effect() in launch_effect.c
#define EFFECTS_TOTAL 27

void launch_effect (int effect);

#endif
~
~
~
~
~
~
~
~
~
"launch_effect.h" 15L, 330C                                  1,1          All
```



```
    // Go to rs232 mode. this function loops forever.
    if (i == 2)
    {
        rs232();
    }

    // Result of bootwait() is something other than 2:
    // Do awesome effects. Loop forever.
    while (1)
    {
        // Show the effects in a predefined order
        for (i=0; i<EFFECTS_TOTAL; i++)
            launch_effect(i);

        // Show the effects in a random order.
        // Comment the two lines above and uncomment this
        // if you want the effects in a random order.
        //launch_effect(rand()%EFFECTS_TOTAL);
    }

}

/*
 * Multiplexer/framebuffer routine
                                                    64,1          16%
```

Step 57: Software: Effect 1, rain

Lets start with one of the simplest effects.

In effect.c you will find the function effect_rain(int iterations).

This effect adds raindrops to the top layer of the cube, then lets them fall down to the bottom layer.

Most of the effects have a main for() loop that loops from i=0 to i < iterations.
effect_rain(int iterations) only takes one argument, which is the number of iterations.

Inside the iteration loop, the function does the following:

1) Create a random number between 0 and 3, lets call it n here.

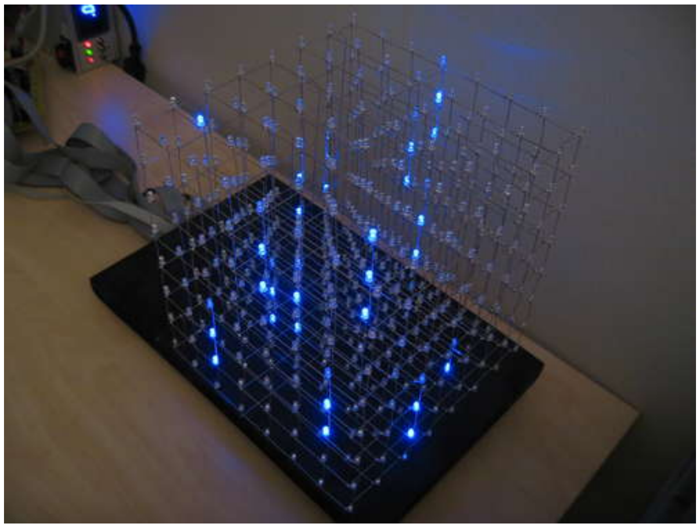
2) Loop a for() loop n number of times.

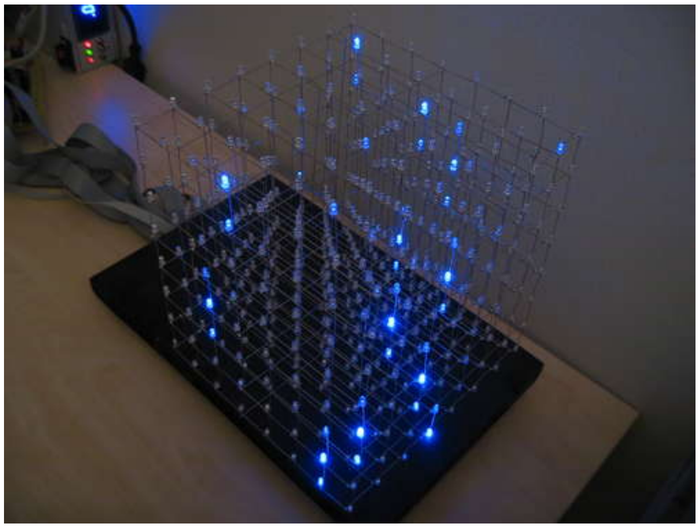
3) For each iteration of this loop, place a pixel on layer 7 (z=7) at random x and y coordinates.

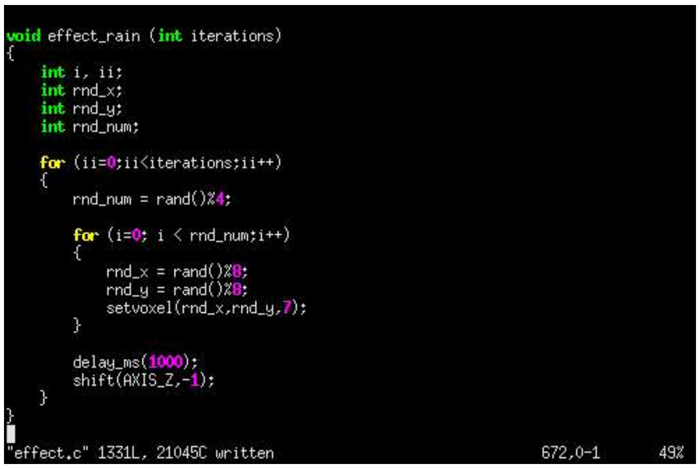
4) Delay for a while

5) Shift the contents of the entire cube along the Z axis by -1 positions. This shifts everything down one level.

This is a pretty simple effect, but it works!







```
void effect_rain (int iterations)
{
    int i, ii;
    int rnd_x;
    int rnd_y;
    int rnd_num;

    for (ii=0;ii<iterations;ii++)
    {
        rnd_num = rand()%4;

        for (i=0; i < rnd_num;i++)
        {
            rnd_x = rand()%8;
            rnd_y = rand()%8;
            setvoxel(rnd_x,rnd_y,7);
        }

        delay_ms(1000);
        shift(AXIS_Z,-1);
    }
}

"effect.c" 1331L, 21045C written                  672,0-1       49%
```

Step 58: Software: Effect 2, plane boing

Another simple effect, effect_planboing(int plane, int speed).

This effect draws a plane along the specified axis then moves it from position 0 to 7 on the axis and back again. This is very simple, but it really brings out the depth of the 3d LED cube :)

This function doesn't have an iteration loop. Instead it is called twice for each axis in launch_effect().

Here is what it does:

1) For()-loop i from 0 to 7.

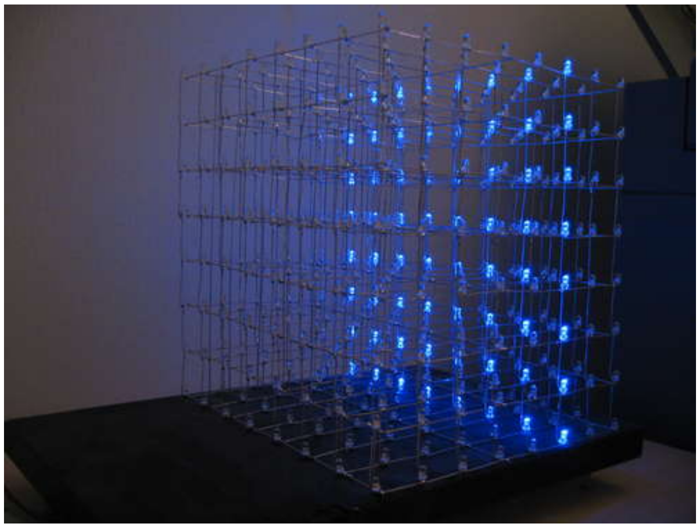
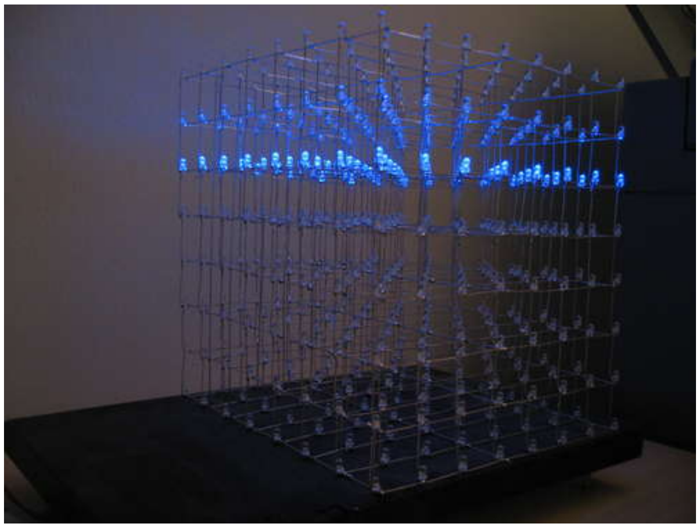
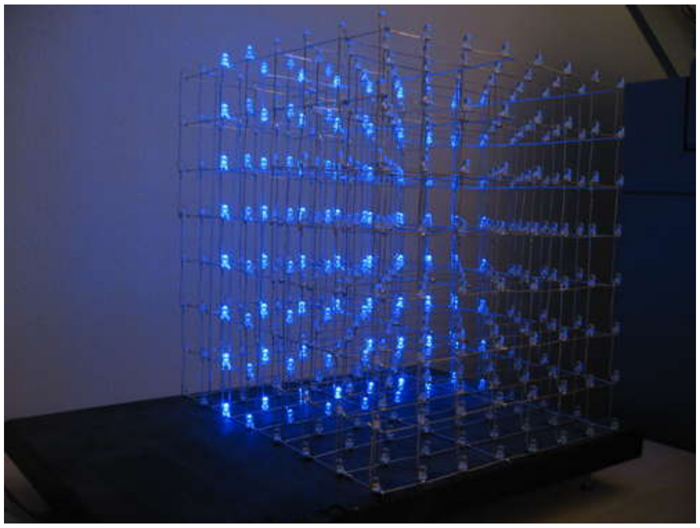
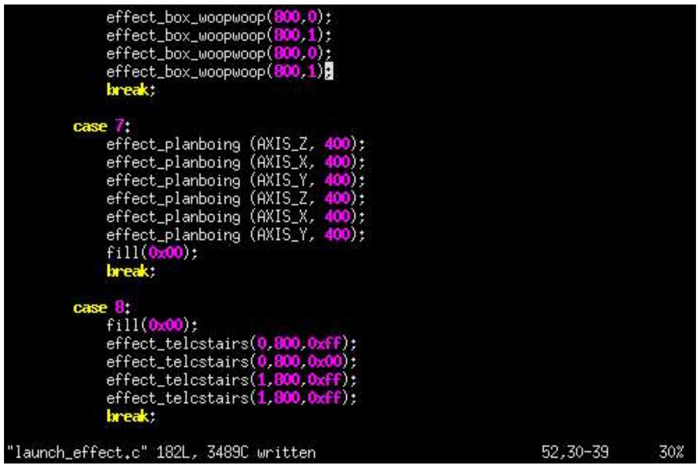
2) Clear the cube with fill(0x00);

3) Call setplane() to draw a plane along the desired axis at position i. The plane isn't actually drawn on the axis specified, it is drawn on the other two axis. If you specify AXIS_Z, a plane is drawn on axis X and Y. It's just easier to think of it that way. Instead of having constants named PLANE_XY, PLANE_YZ etc.

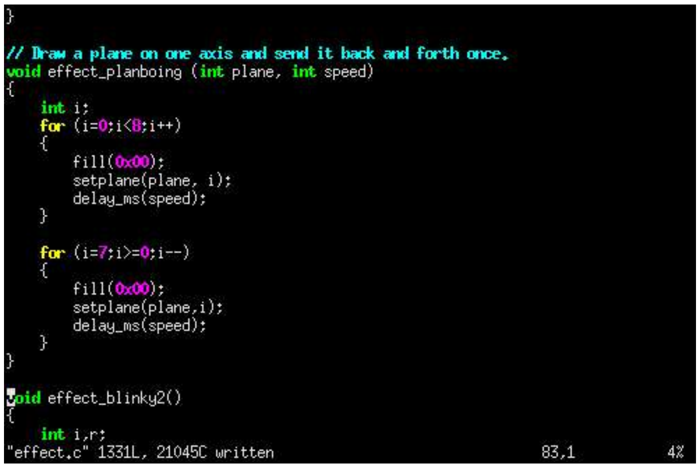
4) Delay for a while.

5) Repeat the same loop with i going from 7 to 0.

Very simple, but a very cool effect!

```
            effect_box_woopwoop(800,0);
            effect_box_woopwoop(800,1);
            effect_box_woopwoop(800,0);
            effect_box_woopwoop(800,1);
            break;

        case 7:
            effect_planboing (AXIS_Z, 400);
            effect_planboing (AXIS_X, 400);
            effect_planboing (AXIS_Y, 400);
            effect_planboing (AXIS_Z, 400);
            effect_planboing (AXIS_X, 400);
            effect_planboing (AXIS_Y, 400);
            fill(0x00);
            break;

        case 8:
            fill(0x00);
            effect_telcstairs(0,800,0xff);
            effect_telcstairs(0,800,0x00);
            effect_telcstairs(1,800,0xff);
            effect_telcstairs(1,800,0xff);
            break;

"launch_effect.c" 182L, 3489C written                          52,30-39      30%
```



```
}

// Draw a plane on one axis and send it back and forth once.
void effect_planboing (int plane, int speed)
{
    int i;
    for (i=0;i<8;i++)
    {
        fill(0x00);
        setplane(plane, i);
        delay_ms(speed);
    }

    for (i=7;i>=0;i--)
    {
        fill(0x00);
        setplane(plane,i);
        delay_ms(speed);
    }
}

void effect_blinky2()
{
    int i,r;
"effect.c" 1331L, 21045C written                               83,1          4%
```

Step 59: Software: Effect 3, sendvoxels random Z

This effect sends voxels up and down the Z axis, as the implies.

void sendvoxels_rand_z() takes three arguments. Iterations is the number of times a voxel is sent up or down. Delay is the speed of the movement (higher delay means lower speed). Wait is the delay between each voxel that is sent.

This is how it works:

1) The cube is cleared with fill(0x00);

2) Loop through all 64 positions along X/Y and randomly set a voxel at either Z=0 or Z=7.

3) Enter the main iteration loop

4) Select random coordinates for X and Y between 0 and 7. If the X and Y coordinates are identical to the previous coordinates, this iteration is skipped.

5) Check if the voxel at this X/Y coordinate is at Z=0 or Z=7, and send it to the opposite side using sendvoxel_z().

6) Delay for a while and save the coordinates of this iteration so we can check them against the random coordinates in the next iteration. It looked weird to move the same voxel twice in a row.

The actual movement of the voxels is done by another function, sendvoxel_z. The reason for this, is that a couple of other effects does the same thing only in different ways.

The function sendvoxel_z() takes four argument. X and Y coordinates. Z coordinate, this is the destination and can either be 0 or 7. Delay which controls the speed.

This is how it works:

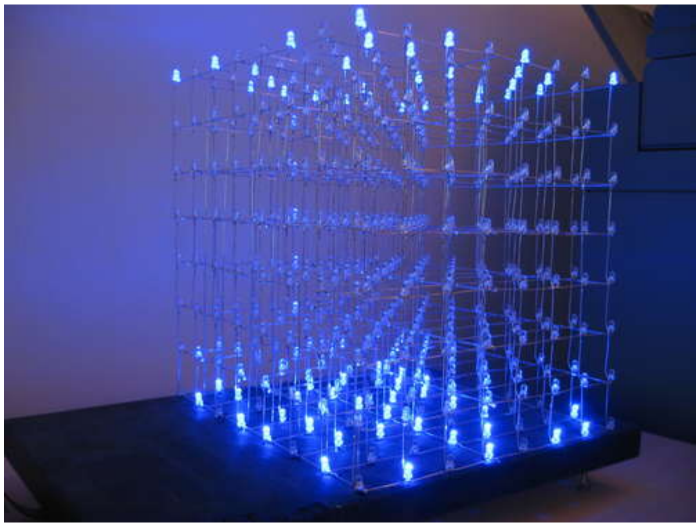
1) For()-loop i from 0 to 7.

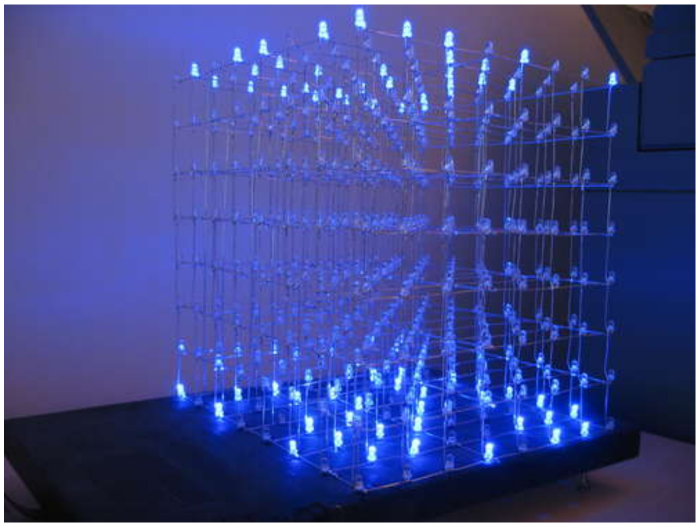
2) If the destination is 7, we set ii to 7-1, thus making ii the reverse of i. Clear the voxel at Z = ii+1. When moving down, ii+1 is the previous voxel.

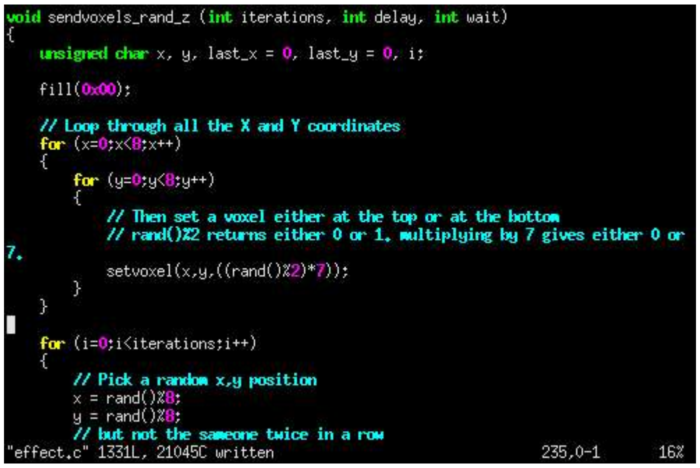
3) If the destination is 0, let ii be equal to i. Clear ii-1. When moving upwards, -1 is the previous voxel.

4) Set the voxel at z=ii.

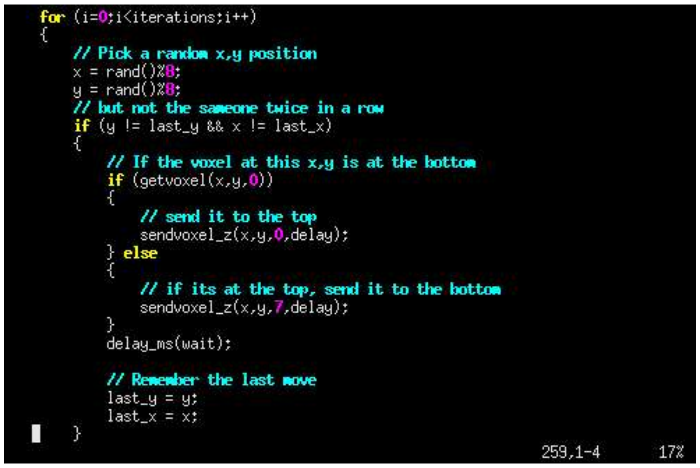
5) Wait for a while.



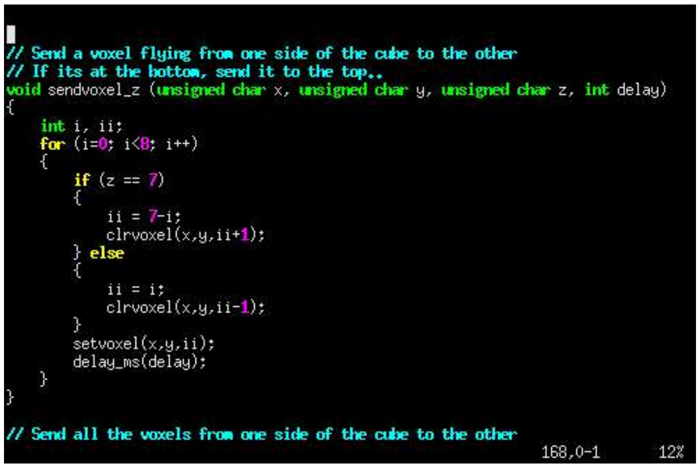




```
void sendvoxels_rand_z (int iterations, int delay, int wait)
{
    unsigned char x, y, last_x = 0, last_y = 0, i;

    fill(0x00);

    // Loop through all the X and Y coordinates
    for (x=0;x<8;x++)
    {
        for (y=0;y<8;y++)
        {
            // Then set a voxel either at the top or at the bottom
            // rand()%2 returns either 0 or 1. multiplying by 7 gives either 0 or
7.
            setvoxel(x,y,((rand()%2)*7));
        }
    }

    for (i=0;i<iterations;i++)
    {
        // Pick a random x,y position
        x = rand()%8;
        y = rand()%8;
        // but not the sameone twice in a row
"effect.c" 1331L, 21045C written                              235,0-1       16%
```



```
    for (i=0;i<iterations;i++)
    {
        // Pick a random x,y position
        x = rand()%8;
        y = rand()%8;
        // but not the sameone twice in a row
        if (y != last_y && x != last_x)
        {
            // If the voxel at this x,y is at the bottom
            if (getvoxel(x,y,0))
            {
                // send it to the top
                sendvoxel_z(x,y,0,delay);
            } else
            {
                // if its at the top, send it to the bottom
                sendvoxel_z(x,y,7,delay);
            }
            delay_ms(wait);

            // Remember the last move
            last_y = y;
            last_x = x;
        }
                                                              259,1-4       17%
```



```
// Send a voxel flying from one side of the cube to the other
// If its at the bottom, send it to the top..
void sendvoxel_z (unsigned char x, unsigned char y, unsigned char z, int delay)
{
    int i, ii;
    for (i=0; i<8; i++)
    {
        if (z == 7)
        {
            ii = 7-i;
            clrvoxel(x,y,ii+1);
        } else
        {
            ii = i;
            clrvoxel(x,y,ii-1);
        }
        setvoxel(x,y,ii);
        delay_ms(delay);
    }
}

// Send all the voxels from one side of the cube to the other
                                                              168,0-1       12%
```

Step 60: Software: Effect 4, box shrinkgrow and woopwoop

A wireframe box is a good geometric shape to show in a monochrome 8x8x8 LED cube. It gives a very nice 3d effect.

We made two box animation functions for the LED cube. Effect_box_shrink_grow() draws a wireframe box filling the entire cube, then shrinks it down to one voxel in one of 8 corners. We call this function one time for each of the 8 corners to create a nice effect. Effect_box_woopwoop() draws a box that starts as a 8x8x8 wireframe box filling the entire cube. It then shrinks down to a 2x2x2 box at the center of the cube. Or in reverse if grow is specified.

Here is how effect_box_shrink_grow() works.

It takes four arguments, number of iterations, rotation, flip and delay. Rotation specifies rotation around the Z axis at 90 degree intervals. Flip > 0 flips the cube upside-down.

To make the function as simple as possible, it just draws a box from 0,0,0 to any point along the diagonal between 0,0,0 and 7,7,7 then uses axis mirror functions from draw.c to rotate it.

1) Enter main iteration loop.

2) Enter a for() loop going from 0 to 15.

3) Set xyz to 7-i. This makes xyz the reverse of i. We want to shrink the box first, then grow. xyz is the point along the diagonal. We just used one variable since x, y and z are all equal along this diagonal.

4) When i = 7, the box has shrunk to a 1x1x1 box, and we can't shrink it any more. If i is greater than 7, xyz is set to i-8, which makes xyz travel from 0 to 7 when i travels from 8 to 15. We did this trick to avoid having two for loops, whith one going from 7-0 and one from 0-7.

5) Blank the cube and delay a little bit to make sure the blanking is rendered on the cube. Disable the interrupt routine. We do this because the mirror functions takes a little time. Without disabling interrupts, the wireframe box would flash briefly in the original rotation before being displayed rotated.

6) Draw the wireframe box in its original rotation. side of the box is always at 0,0,0 while the other travels along the diagonal.

7) Do the rotations. If flip is greather than 0, the cube is turned upside-down. rot takes a number from 0 to 3 where 0 is 0 degrees of rotation around Z and 3 is 270 degrees. To get 270 degrees we simply mirror around X and Y.

8) Enable interrupts to display the now rotated cube.

9) Delay for a while then clear the cube.

The other function involved in the wireframe box effect is effect_box_woopwoop(). The name woopwoop just sounded natural when we first saw the effect rendered on the cube ;)

The woopwoop function only does one iteration and takes two arguments, delay and grow. If grow is greater than 0, the box starts as a 2x2x2 box and grow to a 8x8x8 box.

Here is how it works:

1) Clear the cube by filling the buffer with 0x00;

2) For()-loop from 0 to 3.

4) Set ii to i. If grow is specified we set it to 3-i to reverse it.

5) Draw a wireframe box centered along the diagonal between 0,0,0 and 7,7,7. One corner of the box uses the coordinates 4+ii on all axes, moving from 4-7. The other corner uses 3-ii on all axes, moving from 3-0.

6) Delay for a while, then clear the cube.

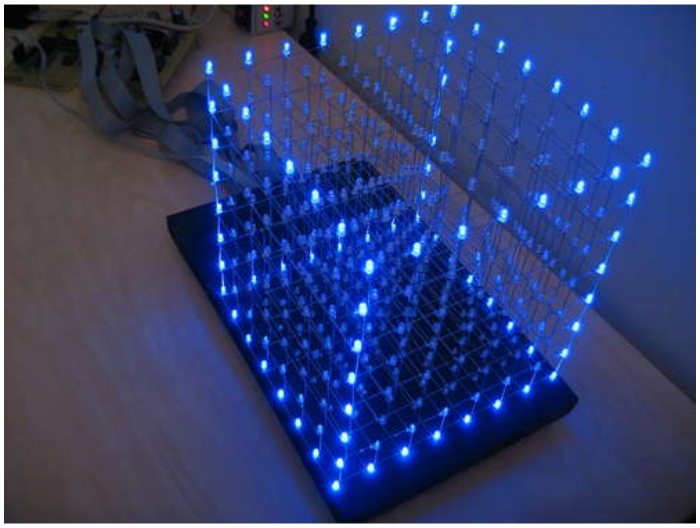
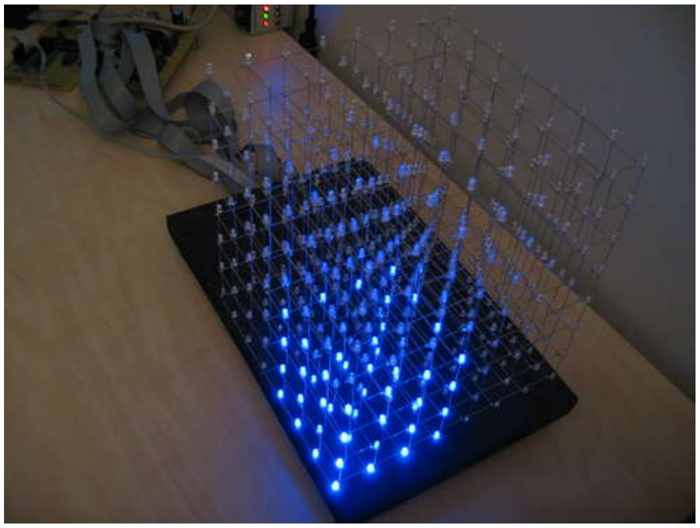
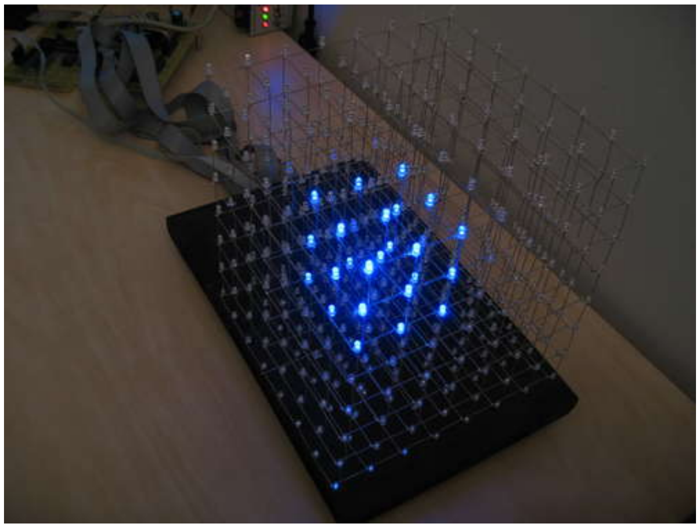
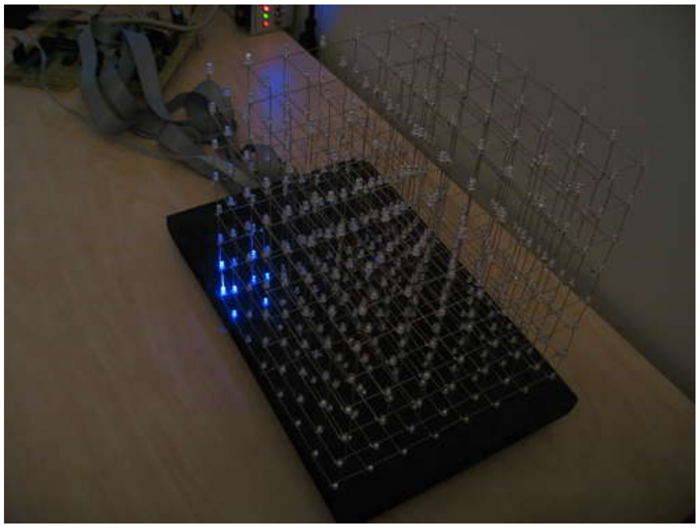
These two functions are used as one single effect in the effect launcher. First the shrink grow effect is called 8 times, one for each corner, then woopwoop is called four times, two shrink and grow cycles.

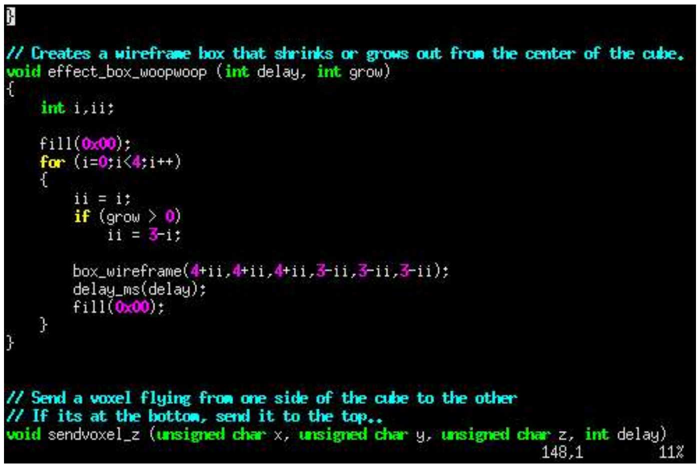
To launch the shrink grow function, we used a for loop with some neat bit manipulation tricks inside to avoid writing 8 lines of code.

The second argument of the shrink grow functions is the rotation, in 4 steps. We are counting from 0 to 7, so we can't simply feed i into the function. We use the modulo operator % to keep the number inside a range of 0-4. The modulo operator divides by the number specifies and returns the remainder.

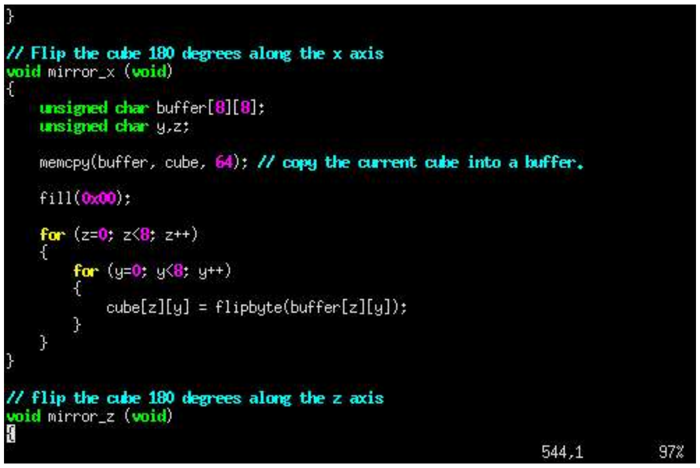
The third argument is the flip. When flip = 0, the cube is not flipped. > 0 flips. We use the bitwise AND operator to only read bit 3 of i.

Bitwise operators are an absolute must to know about when working with micro controllers, but that is outside the scope of this instructable. The guys over at AVR Freaks have posted some great information about this topic. You can read more at http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=37871

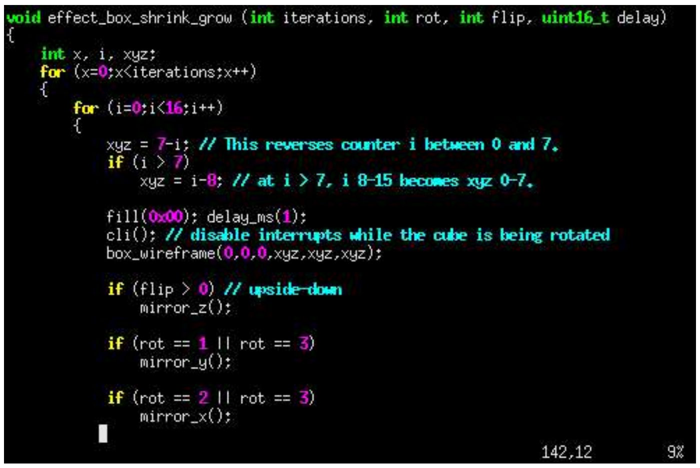







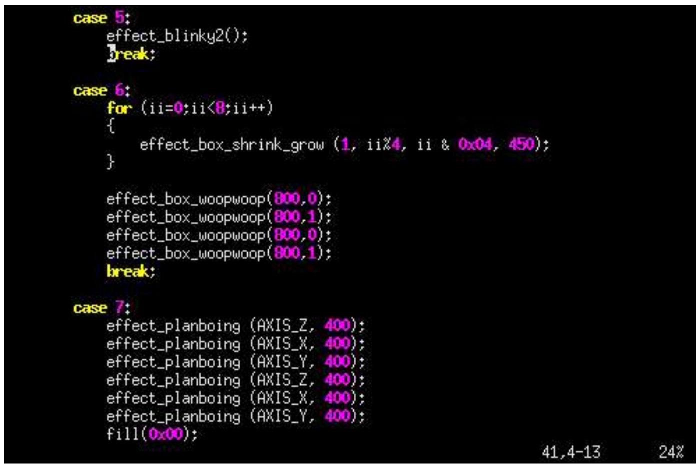
```
}

// Creates a wireframe box that shrinks or grows out from the center of the cube.
void effect_box_woopwoop (int delay, int grow)
{
    int i,ii;

    fill(0x00);
    for (i=0;i<4;i++)
    {
        ii = i;
        if (grow > 0)
            ii = 3-i;

        box_wireframe(4+ii,4+ii,4+ii,3-ii,3-ii,3-ii);
        delay_ms(delay);
        fill(0x00);
    }
}


// Send a voxel flying from one side of the cube to the other
// If its at the bottom, send it to the top..
void sendvoxel_z (unsigned char x, unsigned char y, unsigned char z, int delay)
                                                                148,1          11%
```



```
}

// Flip the cube 180 degrees along the x axis
void mirror_x (void)
{
    unsigned char buffer[8][8];
    unsigned char y,z;

    memcpy(buffer, cube, 64); // copy the current cube into a buffer.

    fill(0x00);

    for (z=0; z<8; z++)
    {
        for (y=0; y<8; y++)
        {
            cube[z][y] = flipbyte(buffer[z][y]);
        }
    }
}

// flip the cube 180 degrees along the z axis
void mirror_z (void)
{
                                                                544,1          97%
```



```
void effect_box_shrink_grow (int iterations, int rot, int flip, uint16_t delay)
{
    int x, i, xyz;
    for (x=0;x<iterations;x++)
    {
        for (i=0;i<16;i++)
        {
            xyz = 7-i; // This reverses counter i between 0 and 7.
            if (i > 7)
                xyz = i-8; // at i > 7, i 8-15 becomes xyz 0-7.

            fill(0x00); delay_ms(1);
            cli(); // disable interrupts while the cube is being rotated
            box_wireframe(0,0,0,xyz,xyz,xyz);

            if (flip > 0) // upside-down
                mirror_z();

            if (rot == 1 || rot == 3)
                mirror_y();

            if (rot == 2 || rot == 3)
                mirror_x();

                                                                142,12          9%
```



```
        case 5:
            effect_blinky2();
            break;

        case 6:
            for (ii=0;ii<8;ii++)
            {
                effect_box_shrink_grow (1, ii%4, ii & 0x04, 450);
            }

            effect_box_woopwoop(800,0);
            effect_box_woopwoop(800,1);
            effect_box_woopwoop(800,0);
            effect_box_woopwoop(800,1);
            break;

        case 7:
            effect_planboing (AXIS_Z, 400);
            effect_planboing (AXIS_X, 400);
            effect_planboing (AXIS_Y, 400);
            effect_planboing (AXIS_Z, 400);
            effect_planboing (AXIS_X, 400);
            effect_planboing (AXIS_Y, 400);
            fill(0x00);
                                                                41,4-13        24%
```

Step 61: Software: Effect 5, axis updown randsuspend

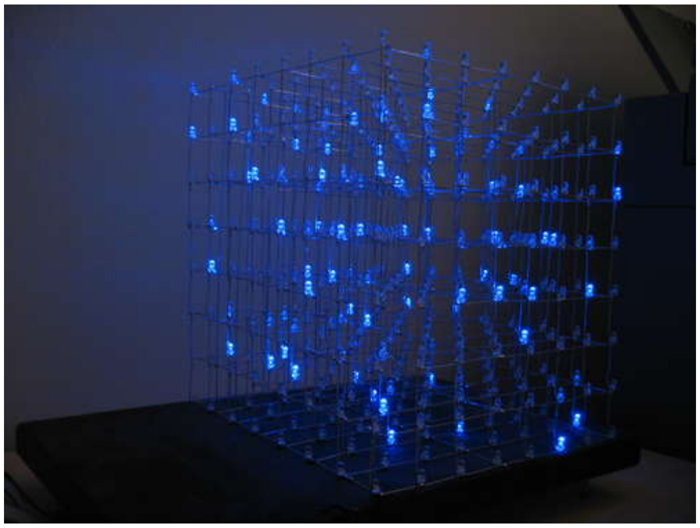
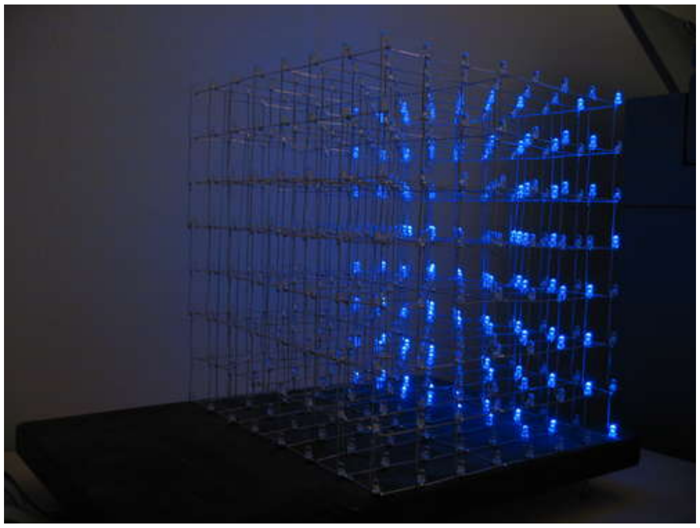
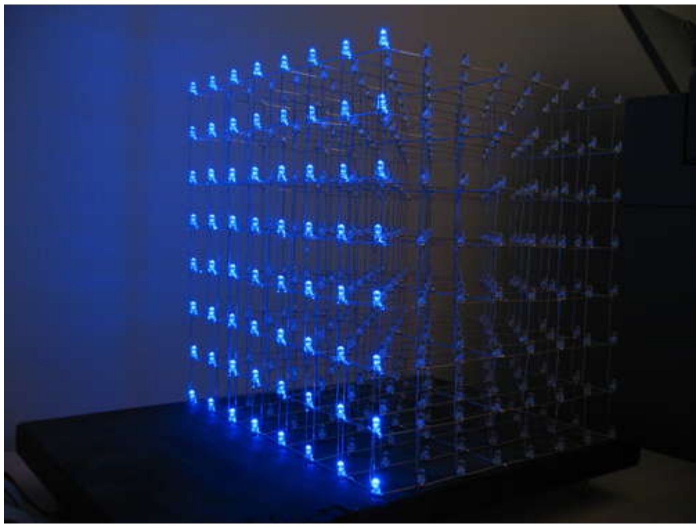
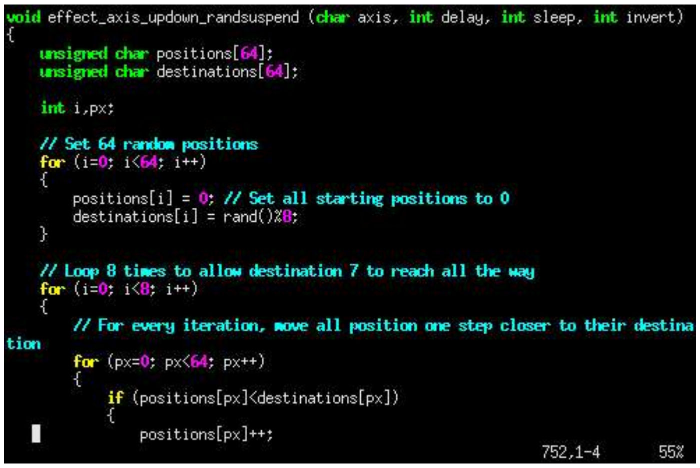
This is one of our favorite effects. The voxels randomly suspended in the cube gives a nice 3d depth, especially if you move your head while viewing the effect.

64 voxels start out on one of the side walls. Then they all get assigned a random midway destination between the side wall they started at and the wall on the opposite side.

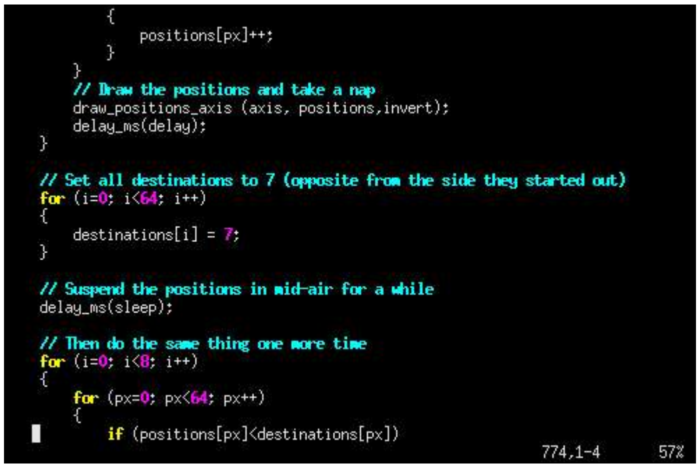
The function then loops 8 times moving each voxel closer to its midway destination. After 8 iterations, the voxels are suspended at different distances from where they started. The function then pauses for a while, thus the name axis_updown_randsuspend ;). It then loops 8 times again moving the voxels one step closer to their final destination on the opposite wall each time.

The actual voxel drawing is done in a separate function, draw_positions_axis() so it can be used in different effects. For example, the voxels could be suspended midway in a non-random pattern. We will leave it up to you to create that effect function! :D

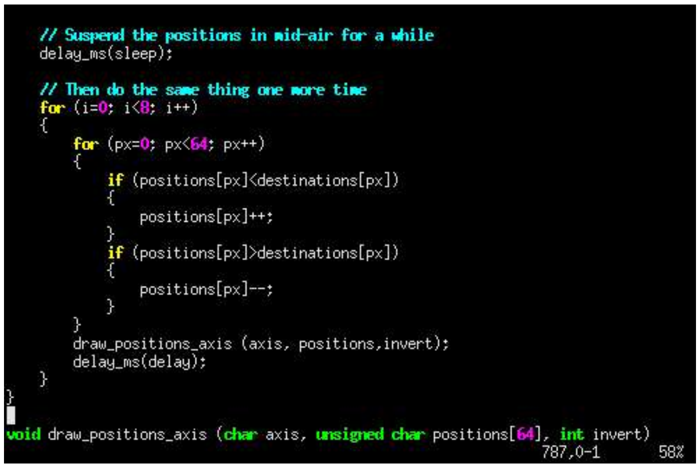
You may have noticed that the description for this effect was less specific. We encourage you to download the source code and read through the functions yourself. Keep the text above in mind when reading the code, and try to figure out what everything does.

```
void effect_axis_updown_randsuspend (char axis, int delay, int sleep, int invert)
{
    unsigned char positions[64];
    unsigned char destinations[64];

    int i,px;

    // Set 64 random positions
    for (i=0; i<64; i++)
    {
        positions[i] = 0; // Set all starting positions to 0
        destinations[i] = rand()%8;
    }

    // Loop 8 times to allow destination 7 to reach all the way
    for (i=0; i<8; i++)
    {
        // For every iteration, move all position one step closer to their destina
tion
        for (px=0; px<64; px++)
        {
            if (positions[px]<destinations[px])
            {
                positions[px]++;
                                                                  752,1-4       55%
```



```
            {
                positions[px]++;
            }
        }
        // Draw the positions and take a nap
        draw_positions_axis (axis, positions,invert);
        delay_ms(delay);
    }

    // Set all destinations to 7 (opposite from the side they started out)
    for (i=0; i<64; i++)
    {
        destinations[i] = 7;
    }

    // Suspend the positions in mid-air for a while
    delay_ms(sleep);

    // Then do the same thing one more time
    for (i=0; i<8; i++)
    {
        for (px=0; px<64; px++)
        {
            if (positions[px]<destinations[px])
                                                                  774,1-4       57%
```



```
    // Suspend the positions in mid-air for a while
    delay_ms(sleep);

    // Then do the same thing one more time
    for (i=0; i<8; i++)
    {
        for (px=0; px<64; px++)
        {
            if (positions[px]<destinations[px])
            {
                positions[px]++;
            }
            if (positions[px]>destinations[px])
            {
                positions[px]--;
            }
        }
        draw_positions_axis (axis, positions,invert);
        delay_ms(delay);
    }
}

void draw_positions_axis (char axis, unsigned char positions[64], int invert)
                                                                  787,0-1       58%
```

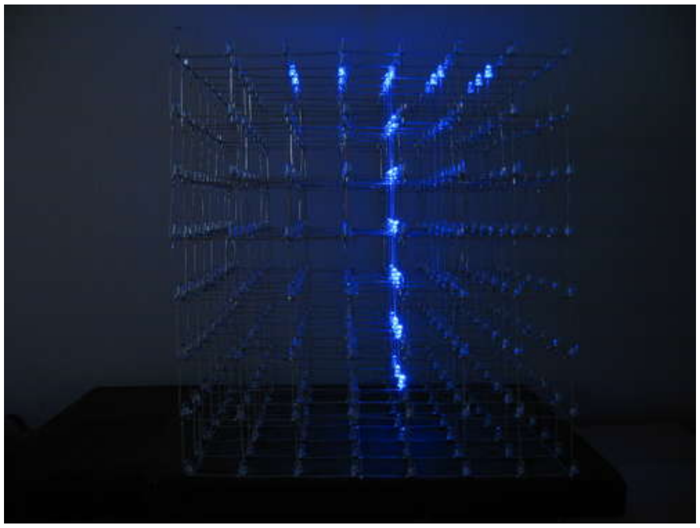
Step 62: Software: Effect 6, stringfly

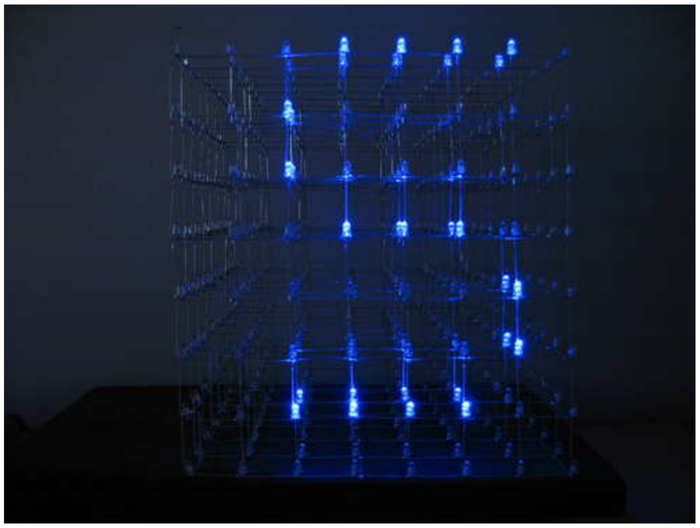
8x8 is about the smallest size required to render a meaningful text font, so we just had to do just that!

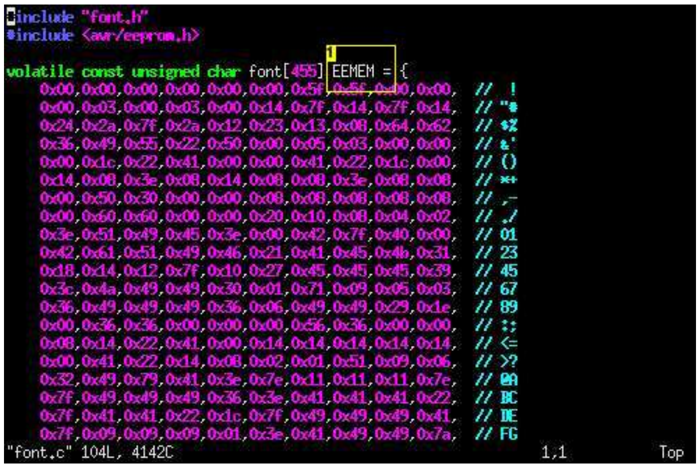
We loaded a 8x5 bitmap font that we had previously used with a graphical LCD display into EEPROM memory, and created some functions that took an ASCII char as an argument and returned a bitmap of the character.

The function stringfly2 takes any ASCII string and displays it as characters flying through the cube.

It starts by placing the character at the back of the cube, then uses the shift() function to shift the cube contents towards you, making the text fly.

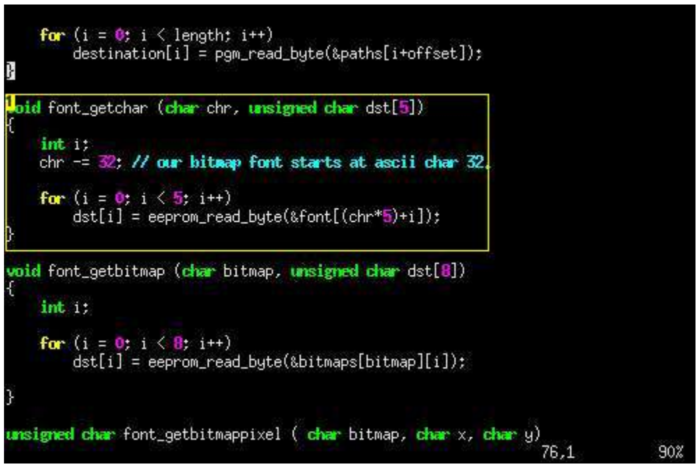






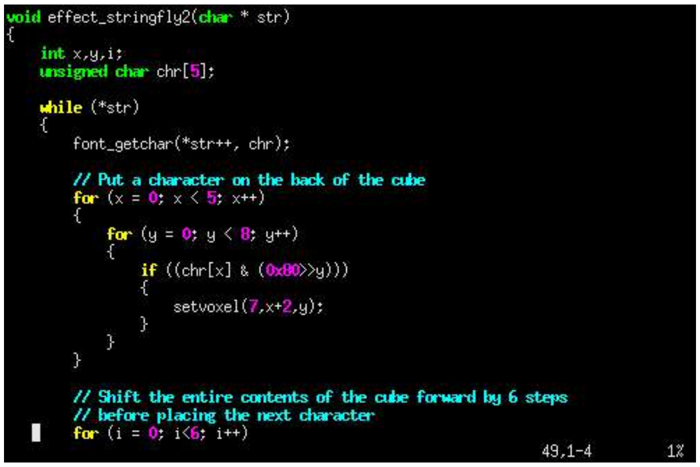
```
#include "font.h"
#include <avr/eeprom.h>

volatile const unsigned char font[455] EEMEM = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x5f,0x5f,0x00,0x00,  //  !
    0x00,0x03,0x00,0x03,0x00,0x14,0x7f,0x14,0x7f,0x14,  // "#
    0x24,0x2a,0x7f,0x2a,0x12,0x23,0x13,0x08,0x64,0x62,  // $%
    0x36,0x49,0x55,0x22,0x50,0x00,0x05,0x03,0x00,0x00,  // &'
    0x00,0x1c,0x22,0x41,0x00,0x00,0x41,0x22,0x1c,0x00,  // ()
    0x14,0x08,0x3e,0x08,0x14,0x08,0x08,0x3e,0x08,0x08,  // *+
    0x00,0x50,0x30,0x00,0x00,0x08,0x08,0x08,0x08,0x08,  // ,-
    0x00,0x60,0x60,0x00,0x00,0x20,0x10,0x08,0x04,0x02,  // ./
    0x3e,0x51,0x49,0x45,0x3e,0x00,0x42,0x7f,0x40,0x00,  // 01
    0x42,0x61,0x51,0x49,0x46,0x21,0x41,0x45,0x4b,0x31,  // 23
    0x18,0x14,0x12,0x7f,0x10,0x27,0x45,0x45,0x45,0x39,  // 45
    0x3c,0x4a,0x49,0x49,0x30,0x01,0x71,0x09,0x05,0x03,  // 67
    0x36,0x49,0x49,0x49,0x36,0x06,0x49,0x49,0x29,0x1e,  // 89
    0x00,0x36,0x36,0x00,0x00,0x00,0x56,0x36,0x00,0x00,  // :;
    0x08,0x14,0x22,0x41,0x00,0x14,0x14,0x14,0x14,0x14,  // <=
    0x00,0x41,0x22,0x14,0x08,0x02,0x01,0x51,0x09,0x06,  // >?
    0x32,0x49,0x79,0x41,0x3e,0x7e,0x11,0x11,0x11,0x7e,  // @A
    0x7f,0x49,0x49,0x49,0x36,0x3e,0x41,0x41,0x41,0x22,  // BC
    0x7f,0x41,0x41,0x22,0x1c,0x7f,0x49,0x49,0x49,0x41,  // DE
    0x7f,0x09,0x09,0x09,0x01,0x3e,0x41,0x49,0x49,0x7a,  // FG
"font.c" 104L, 4142C                                          1,1           Top
```

Image Notes
1. Font stored in EEPROM memory.



```
    for (i = 0; i < length; i++)
        destination[i] = pgm_read_byte(&paths[i+offset]);
}

void font_getchar (char chr, unsigned char dst[5])
{
    int i;
    chr -= 32; // our bitmap font starts at ascii char 32.

    for (i = 0; i < 5; i++)
        dst[i] = eeprom_read_byte(&font[(chr*5)+i]);
}

void font_getbitmap (char bitmap, unsigned char dst[8])
{
    int i;

    for (i = 0; i < 8; i++)
        dst[i] = eeprom_read_byte(&bitmaps[bitmap][i]);

}

unsigned char font_getbitmappixel ( char bitmap, char x, char y)
                                                              76,1          90%
```

Image Notes
1. Takes an ASCII character as input and returns a bitmap of the character.



```
void effect_stringfly2(char * str)
{
    int x,y,i;
    unsigned char chr[5];

    while (*str)
    {
        font_getchar(*str++, chr);

        // Put a character on the back of the cube
        for (x = 0; x < 5; x++)
        {
            for (y = 0; y < 8; y++)
            {
                if ((chr[x] & (0x80>>y)))
                {
                    setvoxel(7,x+2,y);
                }
            }
        }

        // Shift the entire contents of the cube forward by 6 steps
        // before placing the next character
        for (i = 0; i<6; i++)
                                                              49,1-4        1%
```

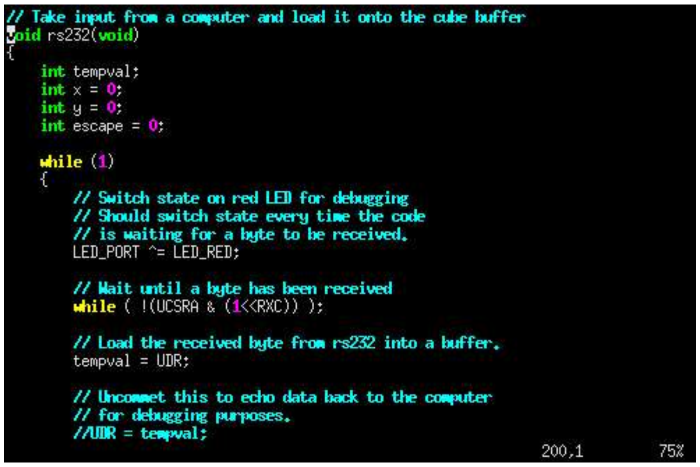
Step 63: Software: RS-232 input

To generate the most awesome effects, we use a desktop computer. Computers can do floating point calculations and stuff like that much quicker than a micro controller. And you don't have to re-program the micro controller for every effect you make, or every time you want to test or debug something.

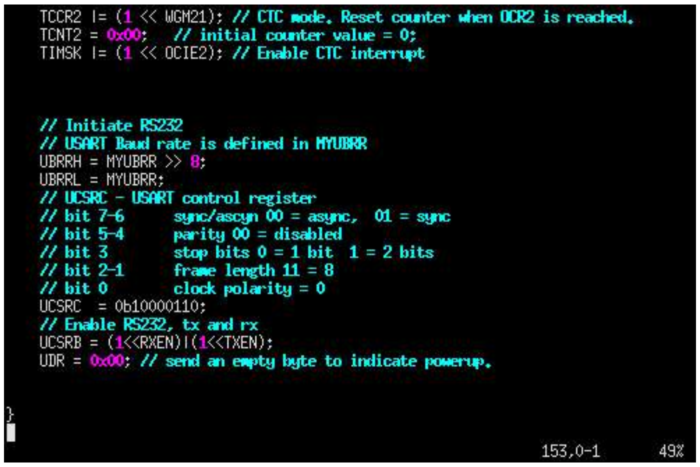
The USART interface in the ATmega is configured to work at 38400 baud with one stop bit and no parity. Each byte that is sent down the line has a start bit and a stop bit, so 10 bits is sent to transmit 8 bits. This gives us a bandwidth of 3840 bytes per second. The cube buffer is 64 bytes. Syncing bytes make up 2 bytes per cube frame. At 38400 baud we are able to send about 58 frames per second. More than enough for smooth animations.

0xff is used as an escape character, and puts the rs232 function into escape mode. If the next byte is 0x00, the coordinates for the buffer are restored to 0,0. If the next byte is 0xff, it is added to the buffer. To send 0xff, you simply send it twice.

The rs232 function just loops forever. A reset is needed to enter the cube's autonomous mode again.



```
// Take input from a computer and load it onto the cube buffer
void rs232(void)
{
    int tempval;
    int x = 0;
    int y = 0;
    int escape = 0;

    while (1)
    {
        // Switch state on red LED for debugging
        // Should switch state every time the code
        // is waiting for a byte to be received.
        LED_PORT ^= LED_RED;

        // Wait until a byte has been received
        while ( !(UCSRA & (1<<RXC)) );

        // Load the received byte from rs232 into a buffer.
        tempval = UDR;

        // Uncommet this to echo data back to the computer
        // for debugging purposes.
        //UDR = tempval;
                                                        200,1          75%
```



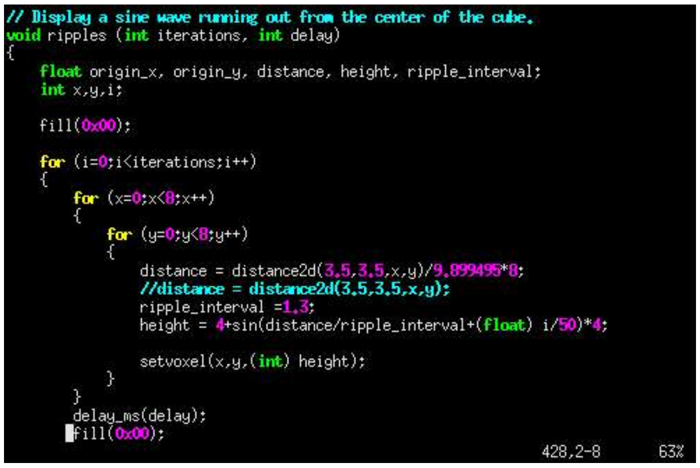
```
    TCCR2 |= (1 << WGM21); // CTC mode. Reset counter when OCR2 is reached.
    TCNT2 = 0x00;    // initial counter value = 0;
    TIMSK |= (1 << OCIE2); // Enable CTC interrupt



    // Initiate RS232
    // USART Baud rate is defined in MYUBRR
    UBRRH = MYUBRR >> 8;
    UBRRL = MYUBRR;
    // UCSRC - USART control register
    // bit 7-6      sync/ascyn 00 = async,  01 = sync
    // bit 5-4      parity 00 = disabled
    // bit 3        stop bits 0 = 1 bit  1 = 2 bits
    // bit 2-1      frame length 11 = 8
    // bit 0        clock polarity = 0
    UCSRC  = 0b10000110;
    // Enable RS232, tx and rx
    UCSRB = (1<<RXEN)|(1<<TXEN);
    UDR = 0x00; // send an empty byte to indicate powerup.


}

                                                        153,0-1        49%
```

Step 64: PC Software: Introduction

The cube just receives binary data via RS232. This data could easily be generated by a number of different programming languages, like python, perl or even php.

We chose to use C for the PC software, since the micro controller software is written in C. This way effects from the micro controller code can just be copy-pasted into the PC software.

Just like in the micro controller code, this code also does two things. Where the micro controller has an interrupt routine that draws the contents of cube[][] onto the LED cube, the PC software has a thread that continually sends data to the LED cube.



```
// Display a sine wave running out from the center of the cube.
void ripples (int iterations, int delay)
{
    float origin_x, origin_y, distance, height, ripple_interval;
    int x,y,i;

    fill(0x00);

    for (i=0;i<iterations;i++)
    {
        for (x=0;x<8;x++)
        {
            for (y=0;y<8;y++)
            {
                distance = distance2d(3.5,3.5,x,y)/9.899495*8;
                //distance = distance2d(3.5,3.5,x,y);
                ripple_interval =1.3;
                height = 4+sin(distance/ripple_interval+(float) i/50)*4;

                setvoxel(x,y,(int) height);
            }
        }
        delay_ms(delay);
        fill(0x00);
                                                    428,2-8        63%
```

File Downloads

cube_pc-v0.1.tar.gz (82 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'cube_pc-v0.1.tar.gz']

Step 65: PC Software: Cube updater thread

In cube.c we have a function called cube_push(). This takes the 64 byte array and sends it down the serial line to the LED cube.

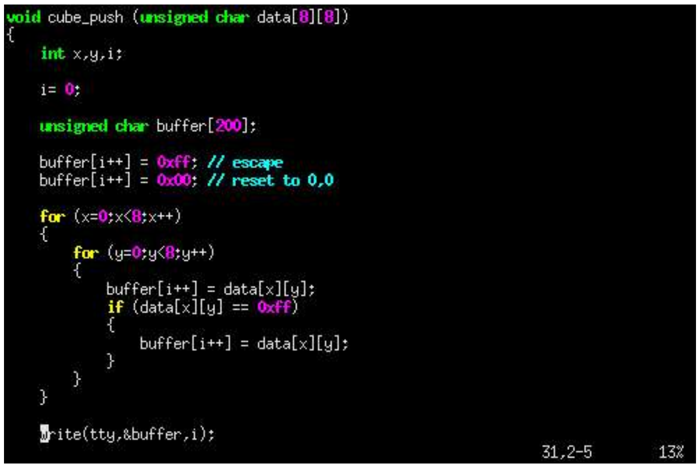
It also handles the formatting, sending every 0xff byte twice because 0xff is our escape character. 0xff and 0x00 is sent first to reset the LED cubes internal x and y counters.

In main.c we have the function cube_updater(). This function is launched as a separate thread using pthread_create(). The main thread and the cube updater thread shares the memory area rs232_cube[8][8]. The cube updater thread is just a while true loop that calls cube_push() over and over.

The first attempt at an updater thread turned out to create some flickering in the animations. After some debugging, we found out that frames were being transmitted before they were fully drawn by the effect functions. We generally do a fill(0x00), then some code to draw new pixels. If a frame is transmitted right after a fill(0x00), the cube will flash an empty frame for 1/60th ish of a second.

This wasn't a problem in the code running on the LED cube, since it has a refresh rate of over 1000 FPS, but at 60 FPS you can notice it.

To overcome this we create a double buffer and sync the two buffers at a point in time where the effect function has finished drawing the frame. Luckily all the effect functions use the delay_ms() function to pause between finished frames. We just put a memcpy() inside there to copy the cube buffer to the rs232 buffer. This works beautifully. No more flickering!



```
void cube_push (unsigned char data[8][8])
{
    int x,y,i;

    i= 0;

    unsigned char buffer[200];

    buffer[i++] = 0xff; // escape
    buffer[i++] = 0x00; // reset to 0,0

    for (x=0;x<8;x++)
    {
        for (y=0;y<8;y++)
        {
            buffer[i++] = data[x][y];
            if (data[x][y] == 0xff)
            {
                buffer[i++] = data[x][y];
            }
        }
    }

    write(tty,&buffer,i);
                                                    31,2-5         13%
```

Step 66: PC Software: Effect 1, ripples

This is the first effect we made for the PC software, and we think it turned out very nice.

While this may seem like a complicated effect, it's really not!

All the effect functions running on the micro controller mostly use if() statements to create effects. The effects on the PC software are built a little different. We use a lot of sin(), cos() and other math functions here. Most coordinates are calculated as floating point coordinates then typecast into integers before being drawn on the cube.

The effect you see in the video is actually just a sine wave eminating from the center of the cube, x=3.5, y=3.5.

Here is how it works:

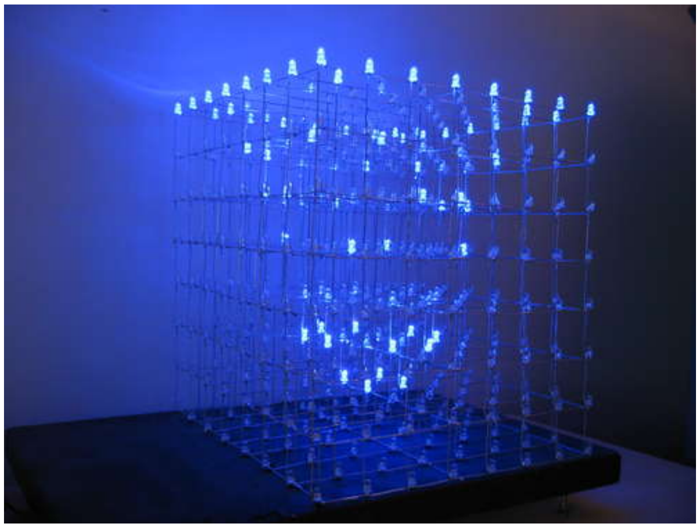
1) Loop through the iteration counter.

2) Loop through all 64 x and y coordinates.

3) Calculate the distance between the center of the cube and the x/y coordinate.

4) The z coordinate is calculated with sin() based on the distance from the center + the iteration counter. The result is that the sine wave moves out from the center as the iteration counter increases.

Look how easy that was!



Step 67: PC Software: Effect 2, sidewaves

This is basically the exact same function as the ripple function.

The only difference is the coordinates of the point used to calculate the distance to each x/y coordinate. We call this point the origin, since the wave emanates from this point.

The origin coordinate is calculated like this:

x = sin(iteration counter) y = cos(iteration counter)

The result is that these x and y coordinates moves around in a circle, resulting in a sin wave that comes in from the side.

We just wanted to show you how easy it is to completely alter an effect by tweaking some variables when working with math based effects!

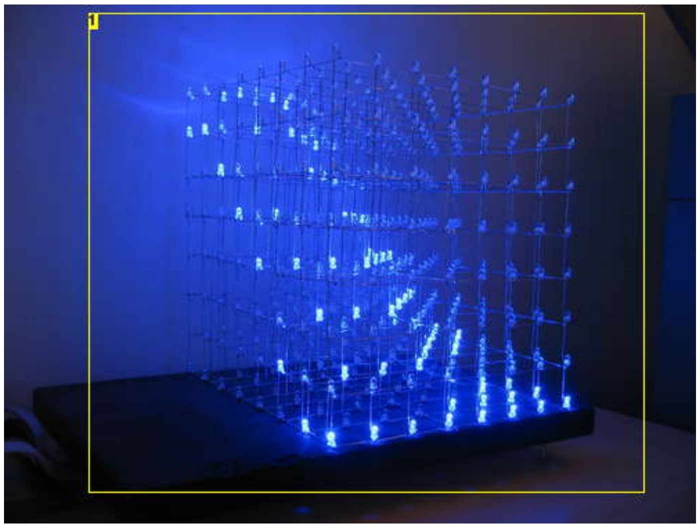


Image Notes
1. Beautiful math!

Step 68: PC Software: Effect 3, fireworks

This effect was quite fun to make.
To make this effect, we really had to sit down and think about how fireworks work, and which forces influence the firework particles.

We came up with a theoretical model of how fireworks work:

1) A rocket is shot up to a random position, origin_x, origin_y, origin_z.

2) The rocket explodes and throws burning particles out in random directions at random velocities.

3) The particles are slowed down by air resistance and pulled towards the ground by gravity.

With this model in mind we created a fireworks effect with a pretty convincing result. Here is how it works:

1) A random origin position is chosen. (within certain limits, x and y between 2 and 5 to keep the fireworks more or less in the center of the cube. z between 5 and 6. Fireworks exploding near the ground can be dangerous! :p)

2) The rocket, in this case a single voxel is moved up the Z axis at the x and y coordinates until it reaches origin_z.

3) An array of n particles is created. Each particle has an x, y and z coordinate as well as a velocity for each axis, dx, dy and dz.

4) We for() loop through 25 particle animation steps:

5) A slowrate is calculated, this is the air resistance. The slowrate is calculated using tan() which will return an exponentially increasing number, slowing the particles faster and faster.

6) A gravity variable is calculated. Also using tan(). The effect of gravity is also exponential. This probably isn't the mathematically correct way of calculating gravity's effect on an object, but it looks good.

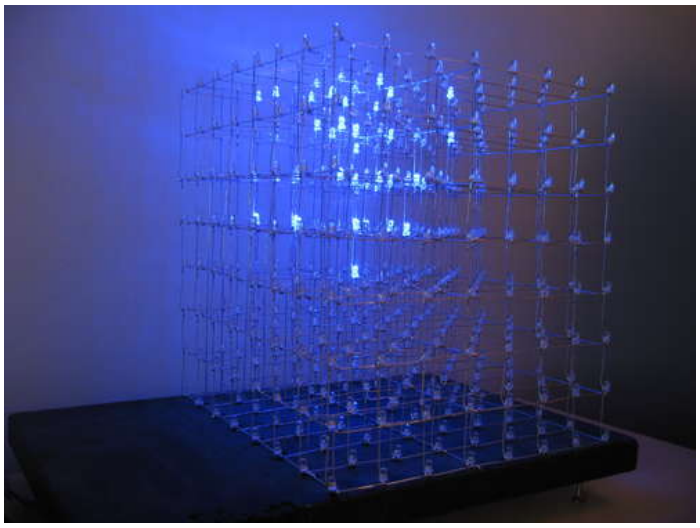
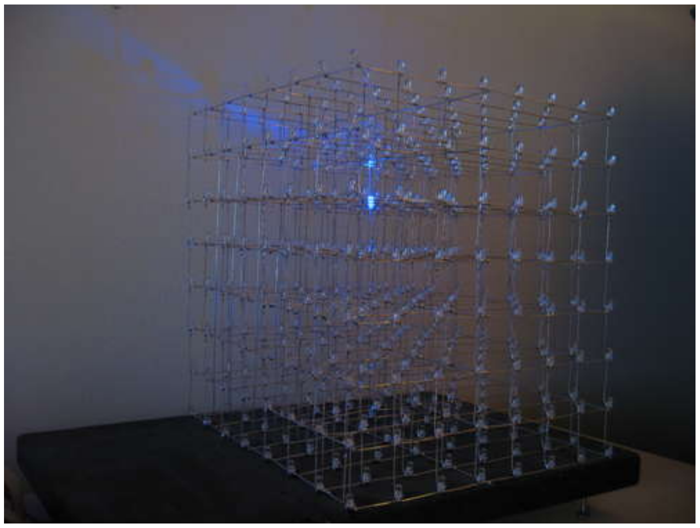
7) For each particle, the x y and z coordinates are incremented by their dx, dy and dz velocities divided by the slowrate. This will make the particles move slower and slower.

8) The z coordinate is decreased by the gravity variable.

9) The particle is drawn on the cube.

10) Delay for a while, then do the next iteration of the explosion animation.

We are quite pleased with the result.

Step 69: PC Software: Effect 4, Conway's Game of Life 3D

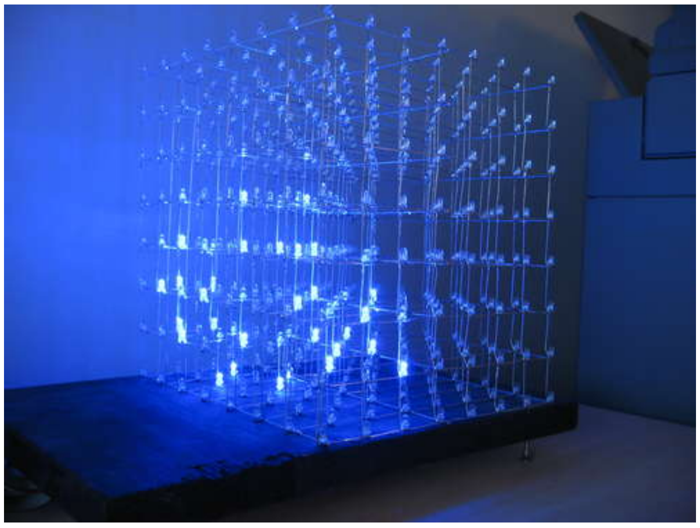
The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway. You can read more about this on Wikipedia , if you haven't heard about it before.

By popular demand, we have implemented Game of Life in 3D on the LED cube.
To make it work in 3d the rules have to be tweaked a little:

- A dead cell becomes alive if it has exactly 4 neighbors
- A live cell with 4 neighbors live
- A live cell with 3 or fewer neighbors die
- A live cell with 5 or more neighbors die

The program starts by placing 10 random voxels in one corner of the cube, then the game of life rules are applied and the iterations started.

In the second video, we run the animation faster and seed with 20 voxels.



Step 70: Run the cube on an Arduino

Since we published our last LED Cube instructable, we have gotten a lot of questions from people wondering if they could use an Arduino to control the cube.

This time, we are one step ahead of you on the "Can i use an arduino?" front :D

The IO requirements for an 8x8x8 LED cube is:

- Layer select: 8
- Data bus for latches: 8
- Address bus for latches: 3
- Output enable (OE) for latches: 1

Total: 21

The Arduino has 13 GPIO pins and 8 analog inputs, which can also be used as GPIO. This gives you a total of 21 IO lines, exactly the amount of IO needed to run the LED cube!

But why write about it when we could just show you?

We hooked the cube up to an Arduino and ported some of the software.

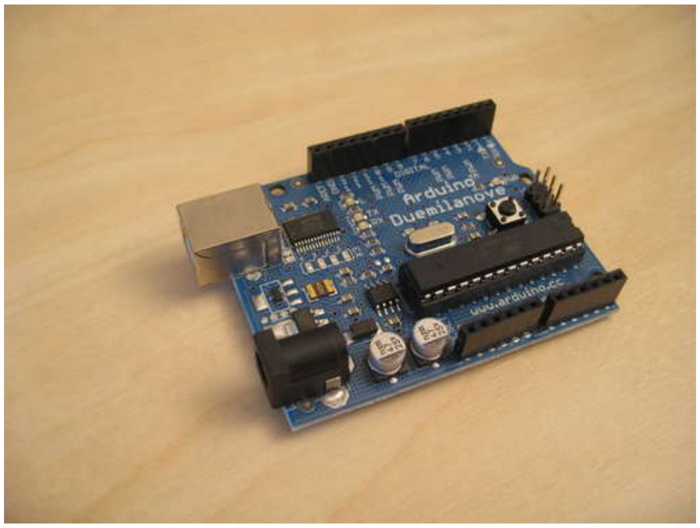
Since the multiplexer array and AVR board are separated by a ribbon cable, connecting the IO lines to an Arduino is a simple matter of connecting some breadboard wires. Luckily, we soldered in a female 0.1" pin header for the transistor lines when we were debugging the first set of transistors. Just remove the ATmega and connect wires from the Arduino to these pin headers.

We connected the cube like this: DATA bus: Digital pins 0-7. This corresponds to PORTD on the ATmega328 on the Arduino board, so we can use direct port access instead of Arduinos digitalWrite (which is slow). Address bus: Digital pins 8-10. This corresponds to PORTB bit 0-2. On this we HAVE to use direct port access. Arduinos digitalWrite wouldn't work with this, because you can't set multiple pins simultaneously. If the address pins are not set at the exact same time, the output of the 74HC138 would trigger the wrong latches. Output Enable: Digital pin 11. Layer transistors: Analog pins 0-5 and digital pins 12 and 13.

We had to go a bit outside the scope of the Arduino platform. The intention of Arduino is to use digitalWrite() for IO port access, to make the code portable and some other reasons. We had to sidestep that and access the ports directly. In addition to that, we had to use one of the timers for the interrupt routine.

The registers for the interrupt and timers are different on different AVR models, so the code may not be portable between different versions of the Arduino board.

The code for our quick Arduino hack is attached.



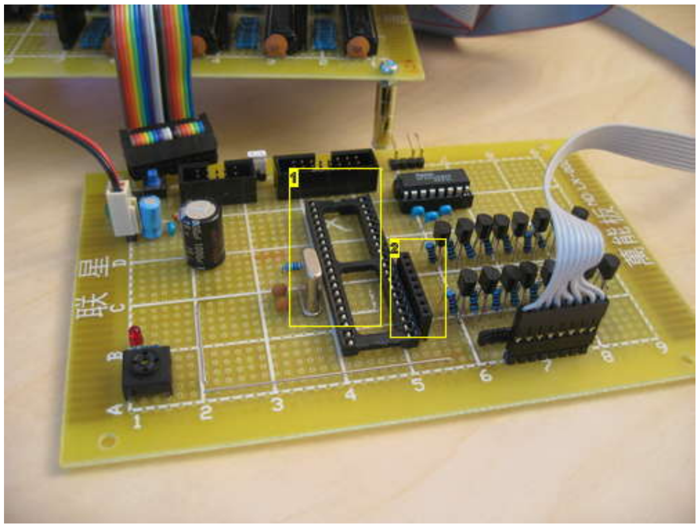
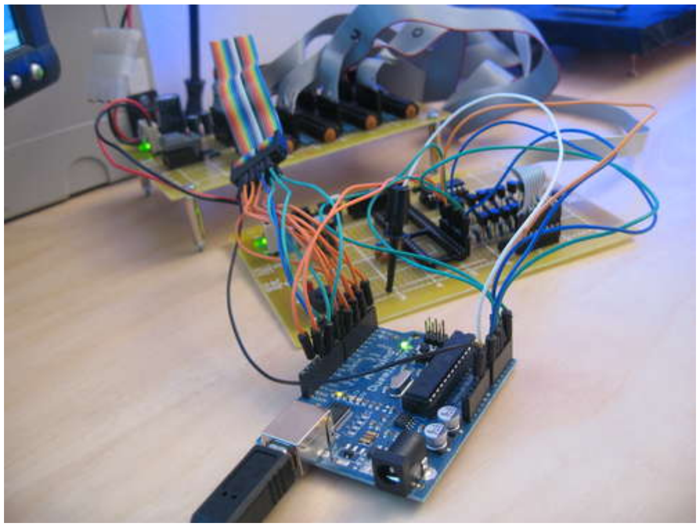
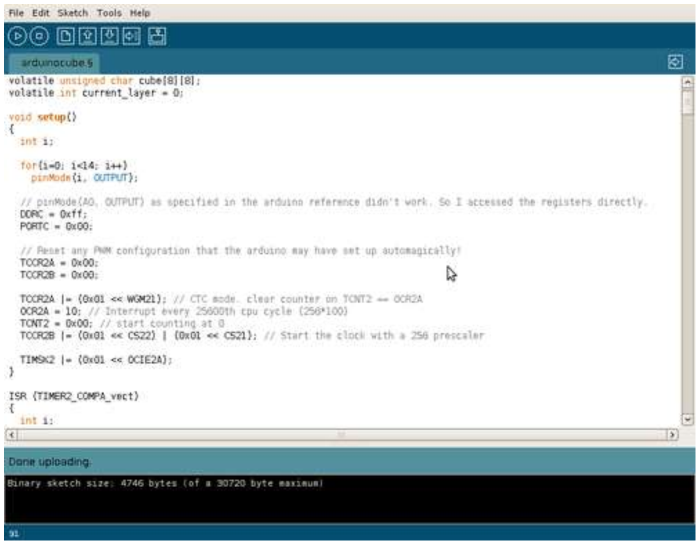


Image Notes
1. ATmega temporarily removed
2. Layer select lines can be connected to this header, without the ATmega interfering.





```
volatile unsigned char cube[8][8];
volatile int current_layer = 0;

void setup()
{
  int i;

  for(i=0; i<14; i++)
    pinMode(i, OUTPUT);

  // pinMode(A0, OUTPUT) as specified in the arduino reference didn't work. So I accessed the registers directly.
  DDRC = 0xff;
  PORTC = 0x00;

  // Reset any PWM configuration that the arduino may have set up automagically!
  TCCR2A = 0x00;
  TCCR2B = 0x00;

  TCCR2A |= (0x01 << WGM21); // CTC mode. clear counter on TCNT2 == OCR2A
  OCR2A = 10; // Interrupt every 25600th cpu cycle (256*100)
  TCNT2 = 0x00; // start counting at 0
  TCCR2B |= (0x01 << CS22) | (0x01 << CS21); // Start the clock with a 256 prescaler

  TIMSK2 |= (0x01 << OCIE2A);
}

ISR (TIMER2_COMPA_vect)
{
  int i;
```

```
Done uploading.
Binary sketch size: 4746 bytes (of a 30720 byte maximum)
91
```

File Downloads

arduinocube.pde (12 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'arduinocube.pde']

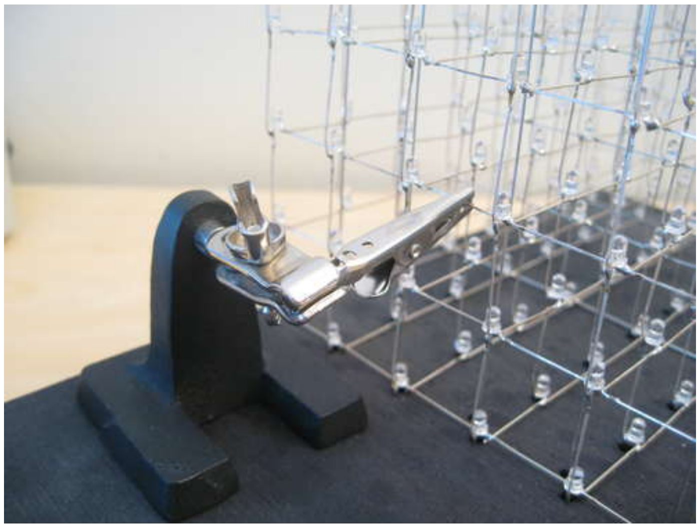
Step 71: Hardware debugging: Broken LEDs

Disaster strikes. A LED inside the cube is broken!

We had a couple of LEDs break actually. Luckily the hardest one to get to was only one layer inside the cube.

To remove the LED, just take a small pair of needle nose pliers and put some pressure on the legs, then give it a light touch with the soldering iron. The leg should pop right out. Do this for both legs, and it's out.

Inserting a new LED is the tricky part. It needs to be as symmetrical and nice as the rest of the LEDs. We used a helping hand to hold it in place while soldering. It went surprisingly well, and we can't even see which LEDs have been replaced.



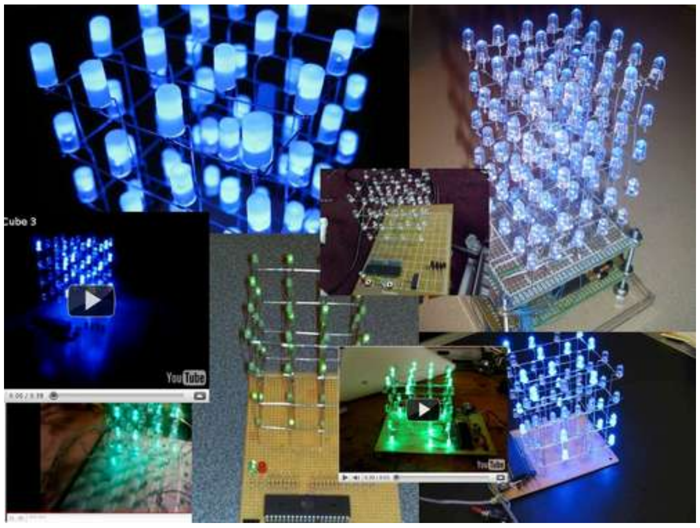
Step 72: Feedback

We love getting feedback on our projects! The 4x4x4 LED cube has received a ton of feedback, and many users have posted pictures and videos of their LED cubes.

If you follow this Instructable and make your own LED cube, please post pictures and video!

Oh, and don't forget to rate this Instructable if you liked it :)

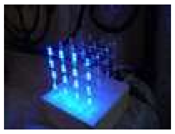
As a token of gratitude for all the great feedback, here is a collage of some of the feedback on our 4x4x4 LED cube instructable:



Related Instructables

Memory led cube by Radobot

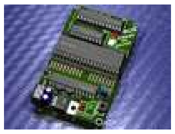
LED Cube 4x4x4 by chr


Led Cube 4x4x4 (video) by bajgik

LED Cube 3x3x3 with ATMEGA8 by G7Electronica.NET

Making a flexible ring of LED's by contrechoc

3D LED Cube by ScitechWA


How to fix dead atmega and attiny avr chips by manekinen

Road Following Robot"Tweety BOT" by <a href="http://www.in Singh"Techie"/?utn class="user">Abhis Singh"Techie"

Power Laces- the Auto lacing shoe

by **blakebevin** on July 3, 2010

Intro: Power Laces- the Auto lacing shoe

UPDATE: You can help bring **Power Laces to market! Click here** to Support the Future!

Also, check out **Power Laces: Version 2.0**

Why wait until 2015?

Inspired by 'Back to The Future II', this project is less 'Practical' than 'Proof of Concept', but hopefully it'll tide you over until Nike comes out with something more polished.

This was also the first time I worked with an Arduino microcontroller, and I wanted to get some experience with the little guy.

Operation is quite simple- step into the shoe and a force sensor reads the pressure of your foot and activates two servo motors, which apply tension to the laces, tightening the shoe. A touch switch reverses the servos.

Due to budget constraints, I only modified one shoe. Where did I put that darn sports almanac?!

And if you get a chance, vote for me in the Instructable USB contest!

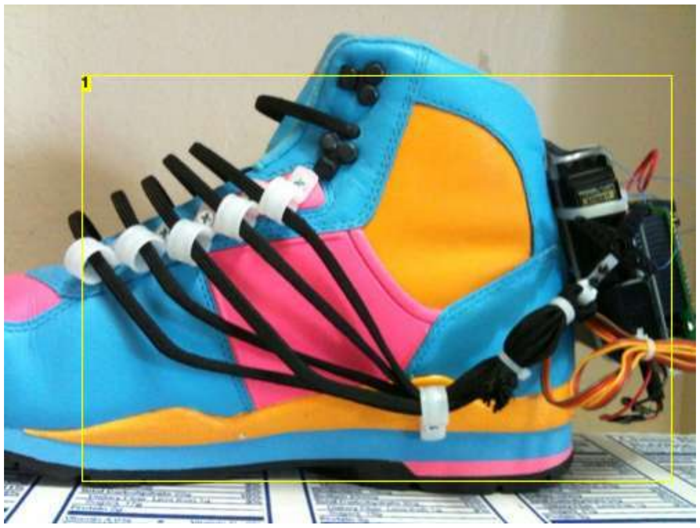


Image Notes
1. A slightly less flattering angle

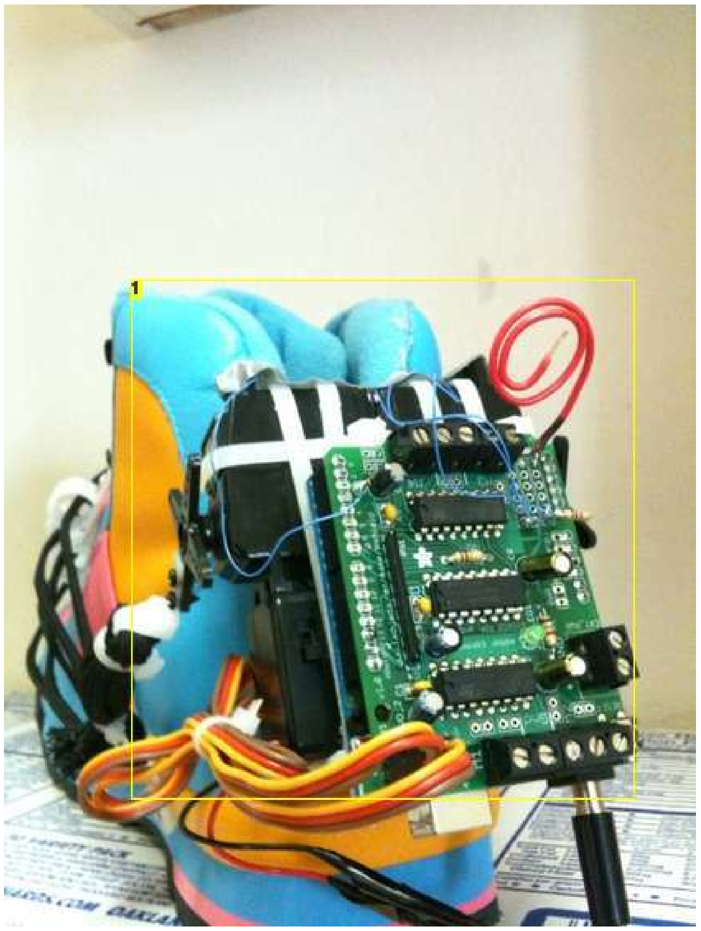


Image Notes
1. Just wait until everyone at the Airport sees your new kicks!

Image Notes
1. Got your hat and shoes, now don't forget all kids in the future wear their pants inside-out.

Image Notes
1. My lab is also a kitchen.
2. Time to go kick it at the Cafe 80s

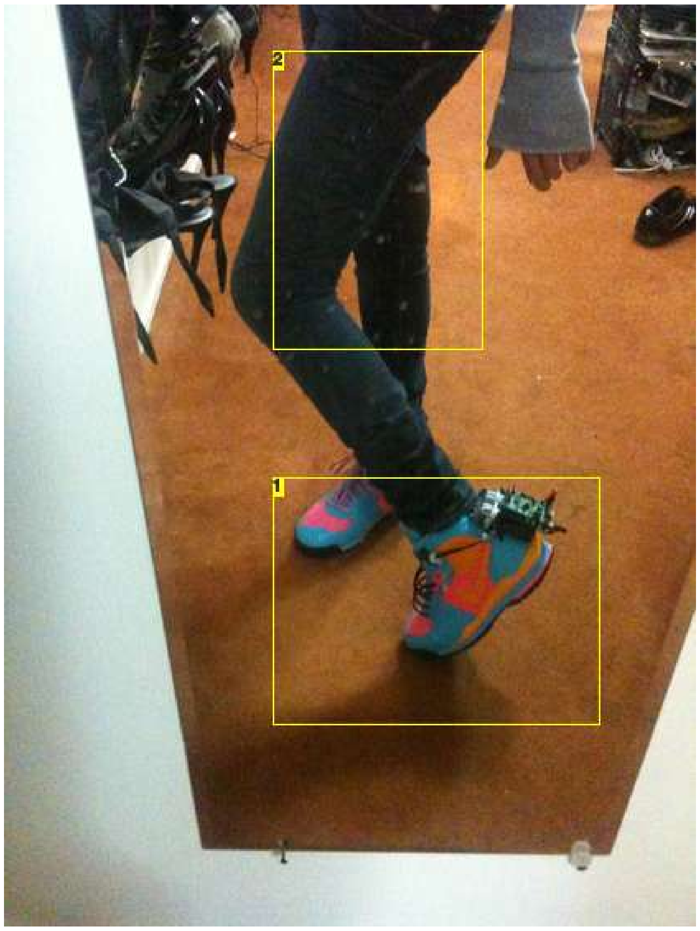


Image Notes
1. Surprisingly wearable!
2. I should get around to cleaning this mirror, eventually.

Step 1: Parts & Tools

Parts:

A shoe // a hightop with a lot of padding and undersole seemed easiest to work with and modify

Arduino // I used the Duemilanove

Motor shield // The kit from adafruit.com works great, and allows control of multiple types of motors

Force sensor // Also got this from adafruit.com

Servo Motors // again, also from adafruit.com. Gotta save on that shipping whenever you can!

Sheet metal // about 4" x 4", enough to keep it's shape but easily trimmed with shears

LED and a couple of resistors // I only had 1K's laying around, so that's what I used

9 Volt case, with built in battery clip and switch

Insulated copper wire // I used high and medium gauge

Plastic zip ties, various sizes // I went through a lot of these

Plastic 1/2" cable loops //used for organizing cables

1/8" braided nylon paracord // 10' should be enough

Tools:

Just the basics- soldering iron, screwdrivers, etc. A hot glue gun is handy, too.

Misc:

A USB A to B cable and a computer to load the Sketch to the Arduino.

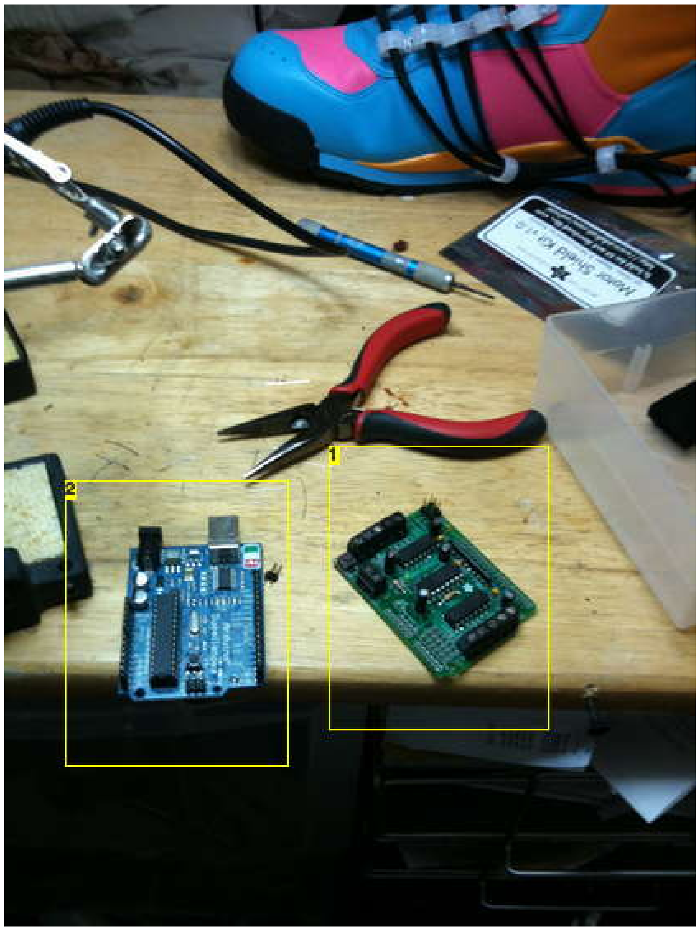


Image Notes
1. Completed Motor Shield
2. Arduino

Step 2: The Laces pt. 1

Cut six lengths of cord, each about 18" long. Remove the inner core and save for future projects.

Fit the zip ties into the holes for the shoelaces on one side. I fit in 5, leaving one space inbetween each zip tie.

Fit the shell of the cord over the zip tie. Don't trim the end, as shown in one of the pictures. Keep in long, as later the end of this will connect to the Servo.

I trimmed the "clipping part" off the zip tie, and then used a lighter to melt the cord down, giving it a cleaner appearance. If you apply some quick heat to a 1/4" area at the end here, it will stiffen and prevent the it from coming out (that's what she said.)

Repeat for the other four.

Screw on the plastic loops and thread through on the other end.

Save the 6th length of outershell cord for later in the project.

Image Notes
1. Pew pew

Image Notes
1. This inner core isn't needed.

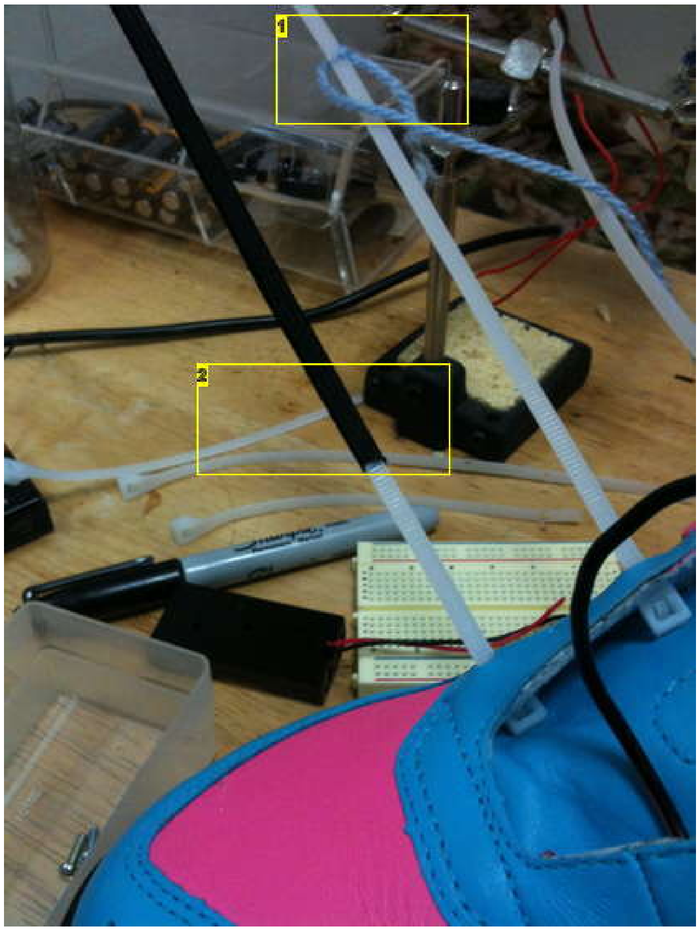


Image Notes
1. Ignore this- was just for protyping
2. Fit the shell onto the Zip tie.

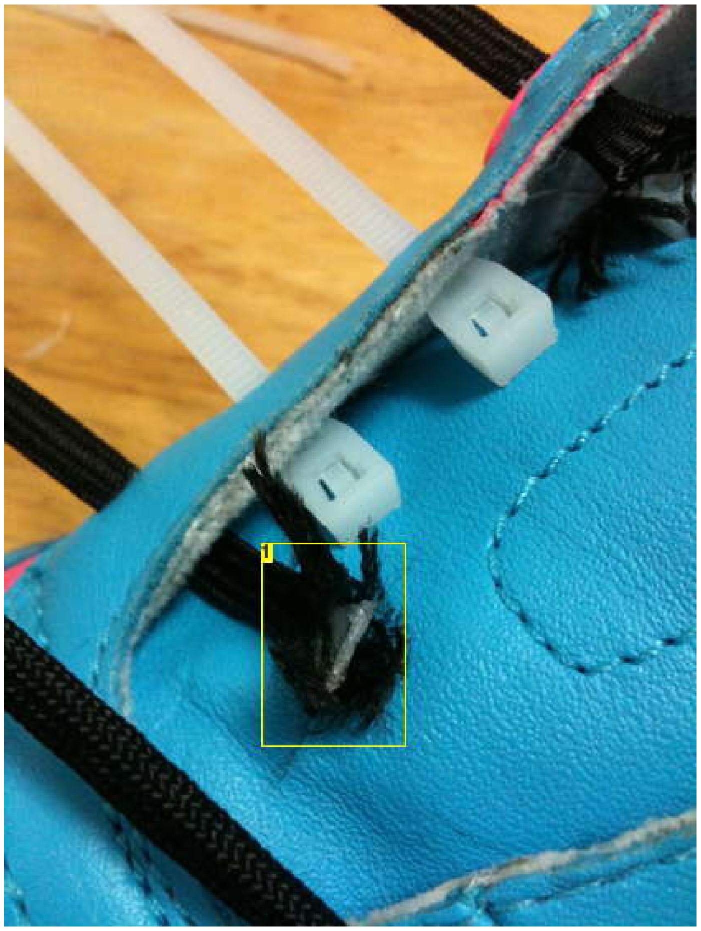
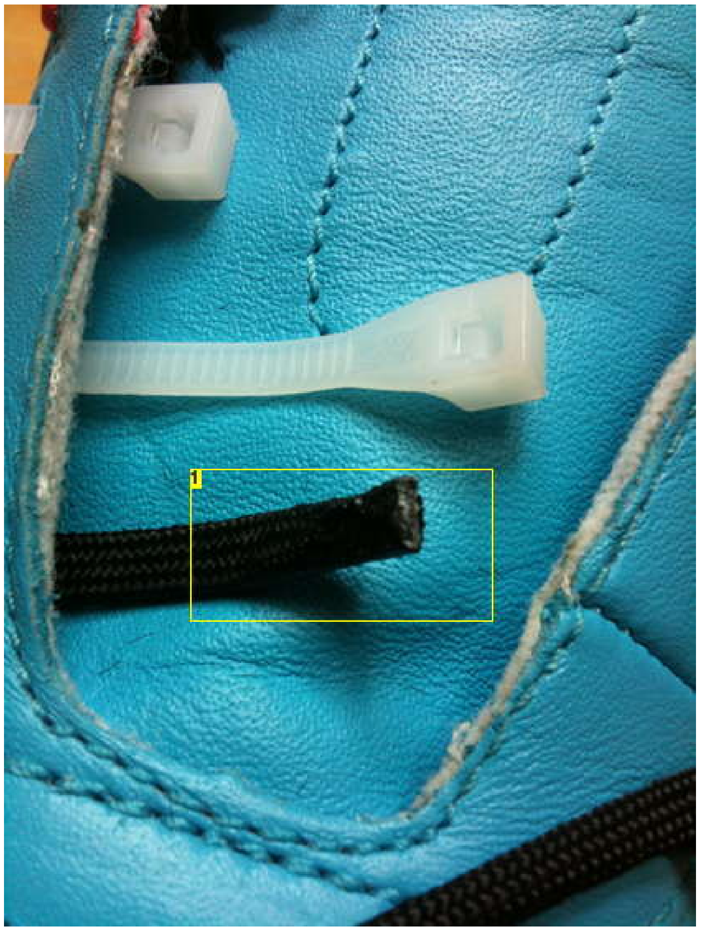


Image Notes
1. Trim right at the head, leaving a bit of a wedge.



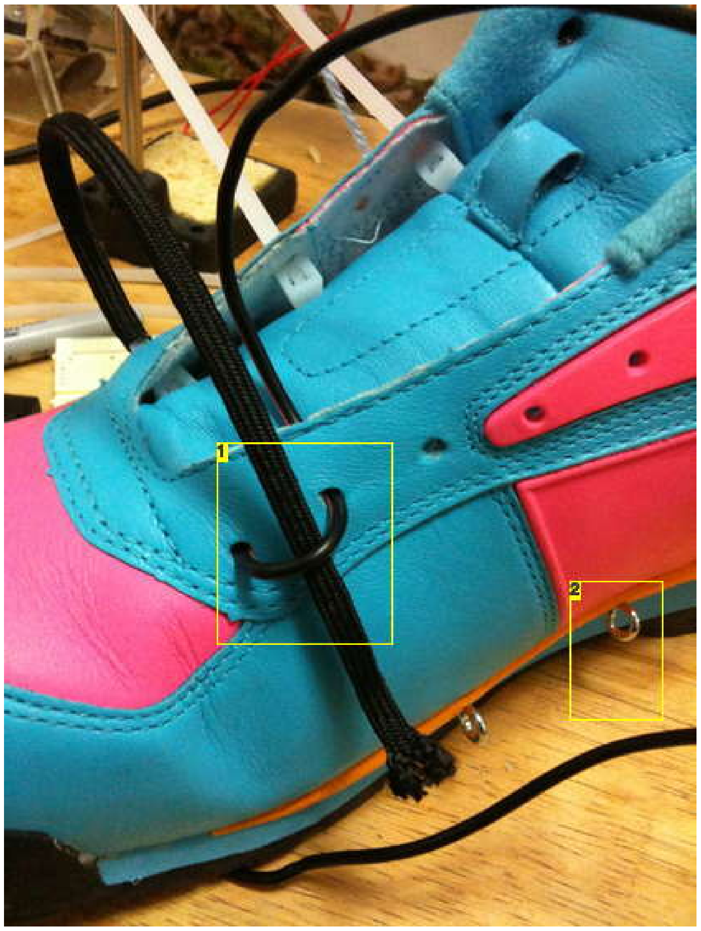


Image Notes
1. Melt end with a lighter, and heat about 1/4" near the end to lightly glaze- this should probably be done in a well ventilated area.

Image Notes
1. This wire was for prototyping to keep the zip tie in place. Later removed.
2. These were later replaced with larger plastic loops.

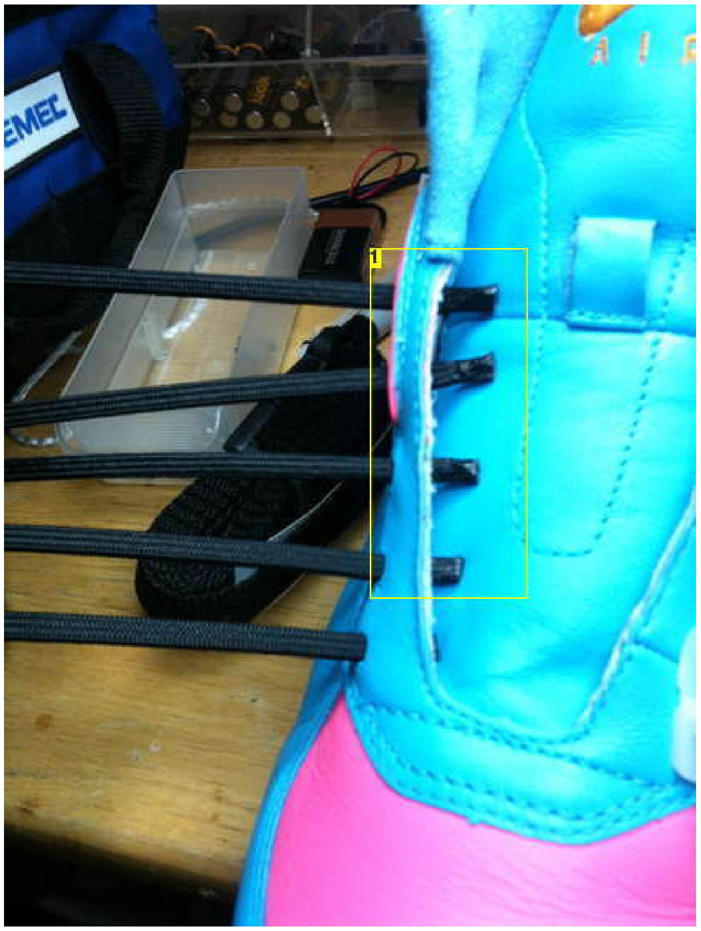


Image Notes
1. Repeat!

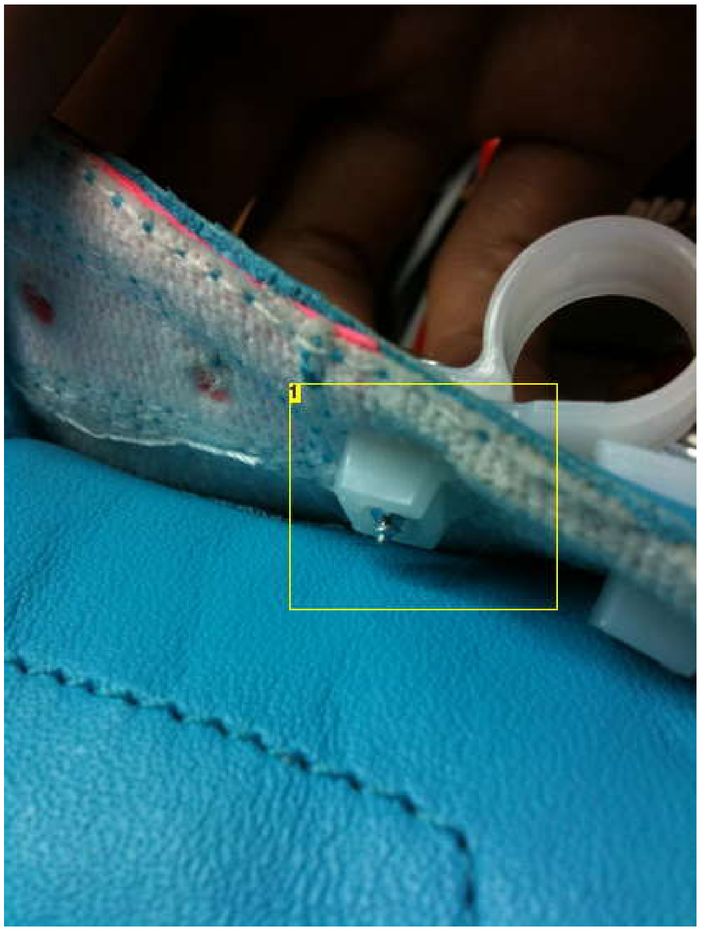


Image Notes
1. I ran out of screw nuts, so I reused the "heads" from the zip ties. I put a little hot glue on the screw tips afterward to prevent it from scratching anything.

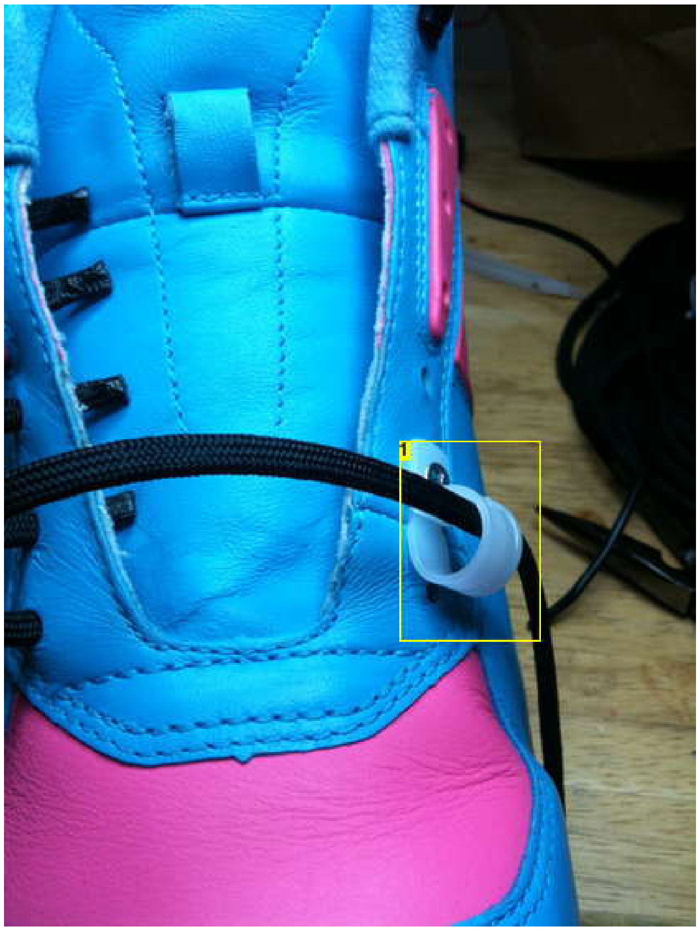


Image Notes
1. If the loop is too small, the laces won't "unlace" properly. 1/2" seems good enough so the untightened lace will "stand out" a bit on it's own.

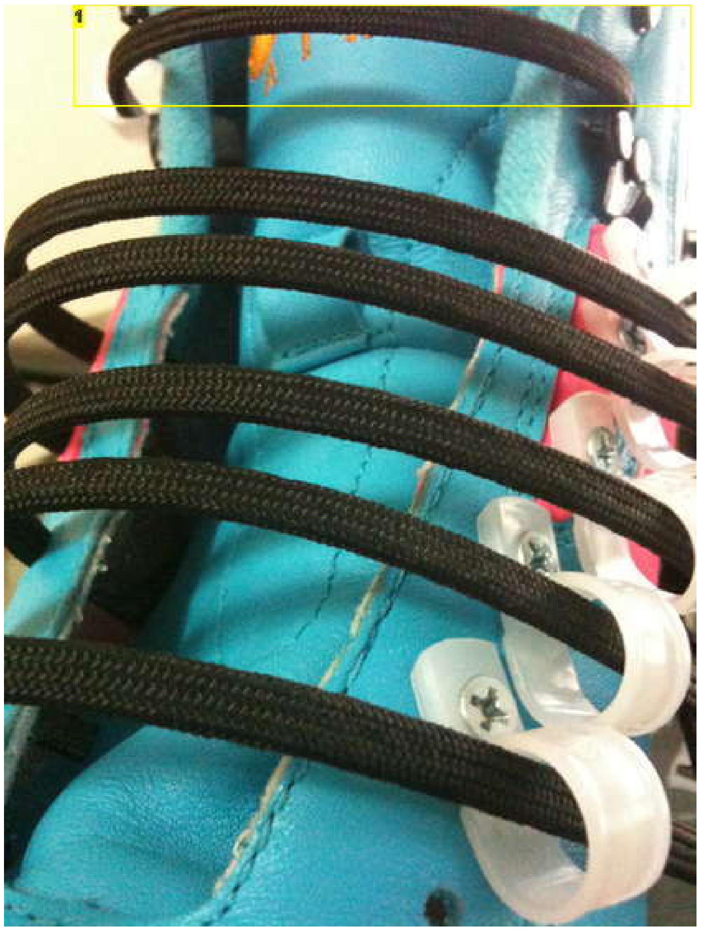


Image Notes
1. Ankle strap, added later in the project.

Step 3: The Laces pt. 2

The ankle strap is mounted pretty much the same way as the lower laces. The ankle strap runs the opposite way of the other straps, to put some counter-force on the servos when they put the laces under tension.

I put a small martini shaker into the shoe to act as a foot analog, so I could make strap adjustments and such. You want to be able to get your foot easily into the shoe, and visually the tightening laces look better if they have a lot of slack.

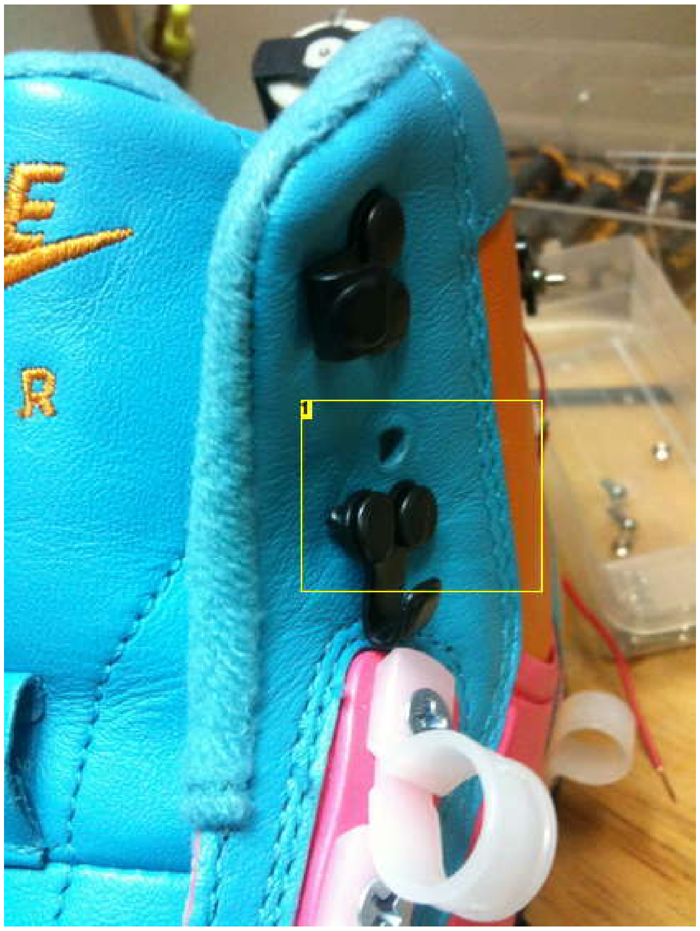


Image Notes
1. I didn't have to drill- this metal lace holder thing was loose and came right off.

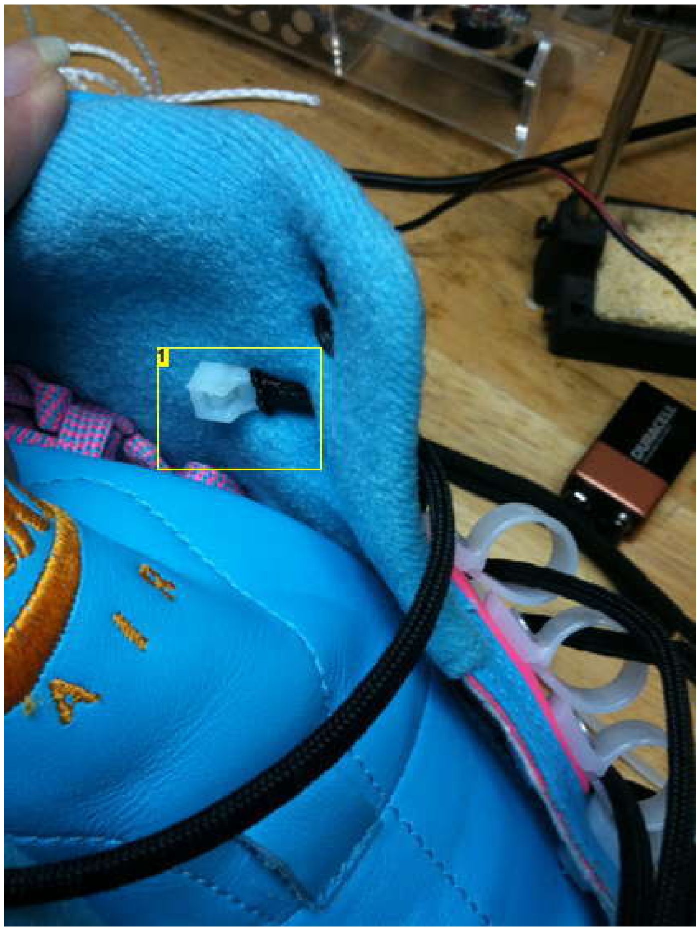


Image Notes
1. I still melted the nylon down a bit, but kept the head on this one attached. I then screwed it into place to secure it.

Image Notes
1. You want to make sure the ankle strap has enough give to let your foot in and out of the shoe easily.

Step 4: Servo Mounting Plate

Trim the sheet metal to fit on the back of the shoe. My pairs have a handy little rubber part at the back and I trimmed the metal to approximate the shape.

After sanding down any rough corners and edges, drill in some holes and use some flat bottomed screws to mount the plate to the shoe. The two screws on the side actually to go into the shoe and are secured with screw nuts, but they're wide enough apart that I don't feel them when I put the shoe on. The bottom screw just goes into the padding.

Leave a bit of a gap between the plate and the shoe for now, as later we're going to put some zip ties through to attach the servo motors.

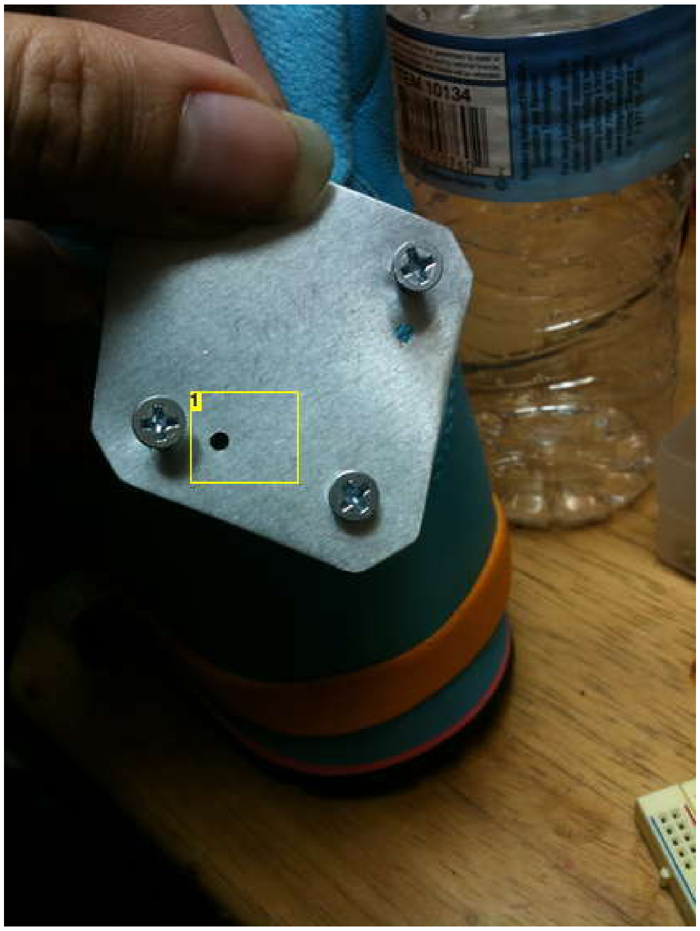


Image Notes
1. These holes were made for a geared motor I originally used. I later replaced it with servos for greater control, but the mounting holes on the servos weren't in the right place, hence the need for zip ties later.

Step 5: Construct the Motor Shield

The Motor Shield is an addon circuit board that enables the Arduino to control various motors, or Servos in this case.

The Shield comes as a kit from Adafruit.com, which has the building instructions along with software libraries that you will need to program and control the Servos.

Following the instructions on the site, solder the Motor Shield together and attach it to the Arduino via the header pins.

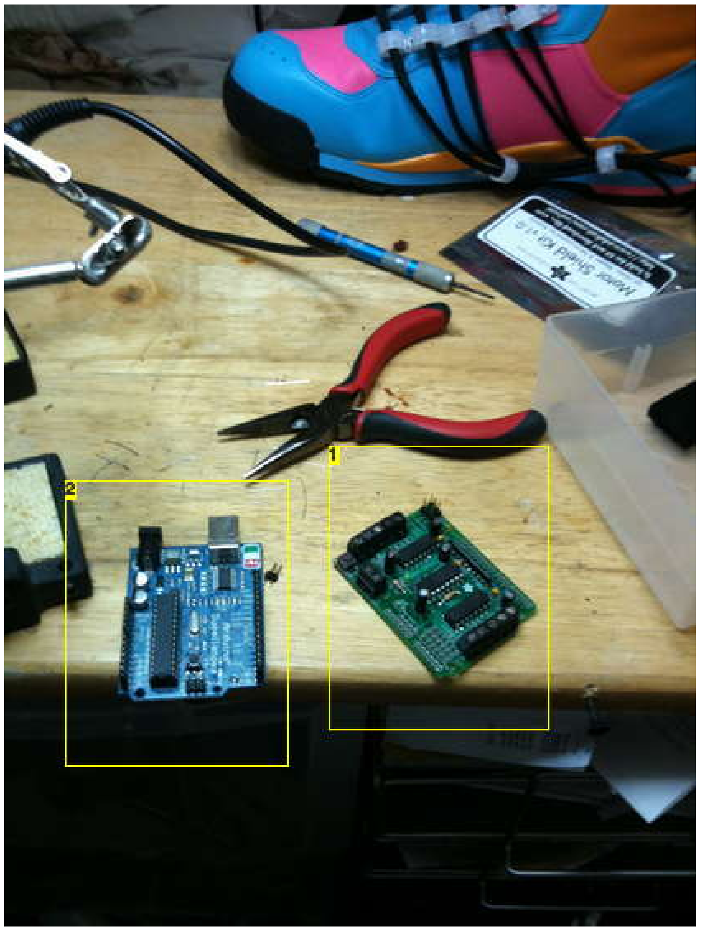


Image Notes
1. Completed Motor Shield
2. Arduino

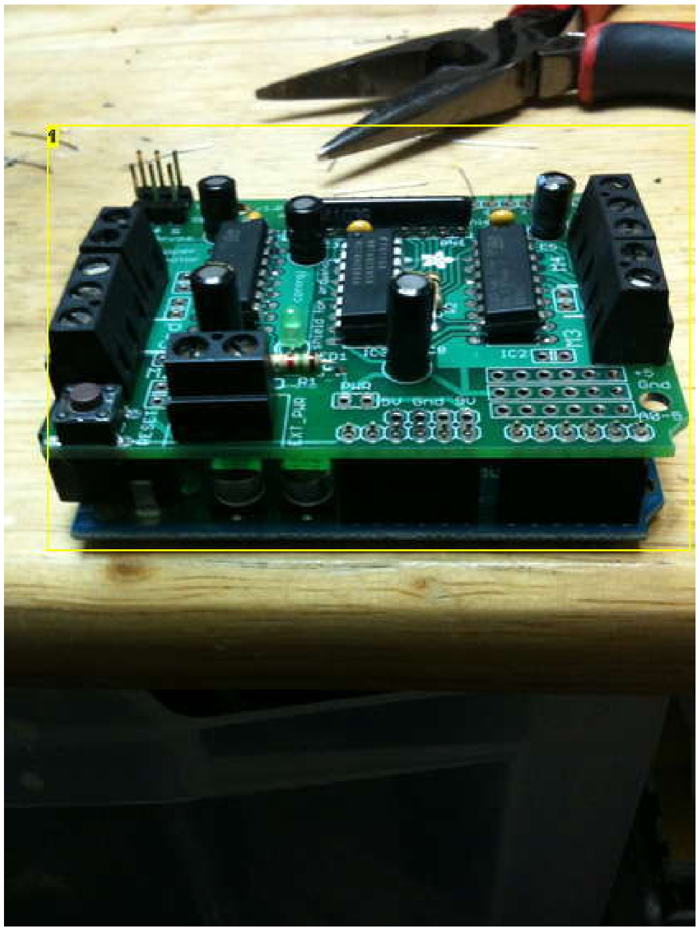


Image Notes
1. Motor shield mounted into place.

Step 6: Mount the Servos, Battery, and Arduino

Zip ties are used to mount the various electronic components.

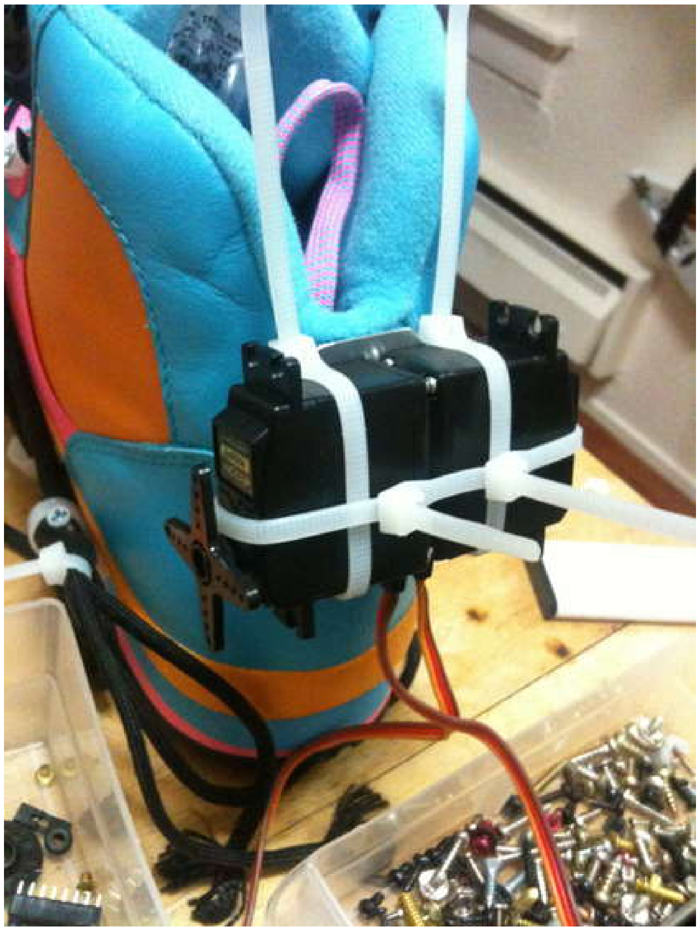
The motors were attached to the mounting plate first by using some rubber cement (optional) to keep them in place, and wrapping and tightening them as much as possible with the zip ties.

Note that most of the ties thread through the space between the mounting plate and the shoe. As my zip ties weren't long enough in some areas, I combined two together. After the servos were in place, I trimmed down the ties.

Beneath the motors, the battery case is mounted in much the same way, power switch outward.

Finally, the Arduino board can be attached. The holes line up with zip ties on both side, and is held in place by screws attached to them.

Before attaching the Motor shield, some modifications must be made.



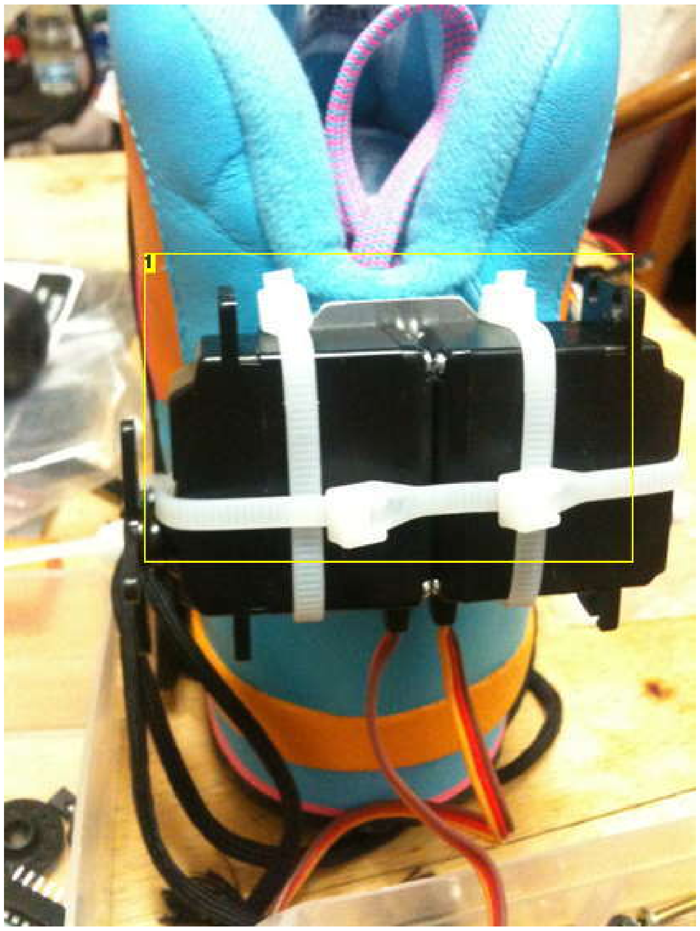


Image Notes
1. Clipped ties

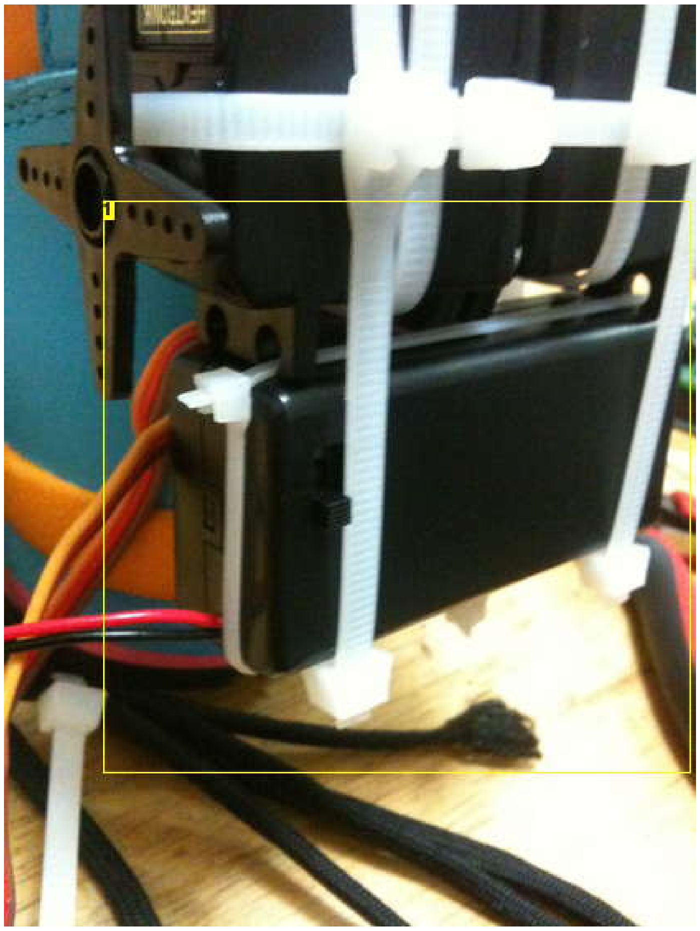


Image Notes
1. The battery case is held in place with a zip tie on both sides. It's tight enough to keep it from falling out, but I can still change the battery as needed.

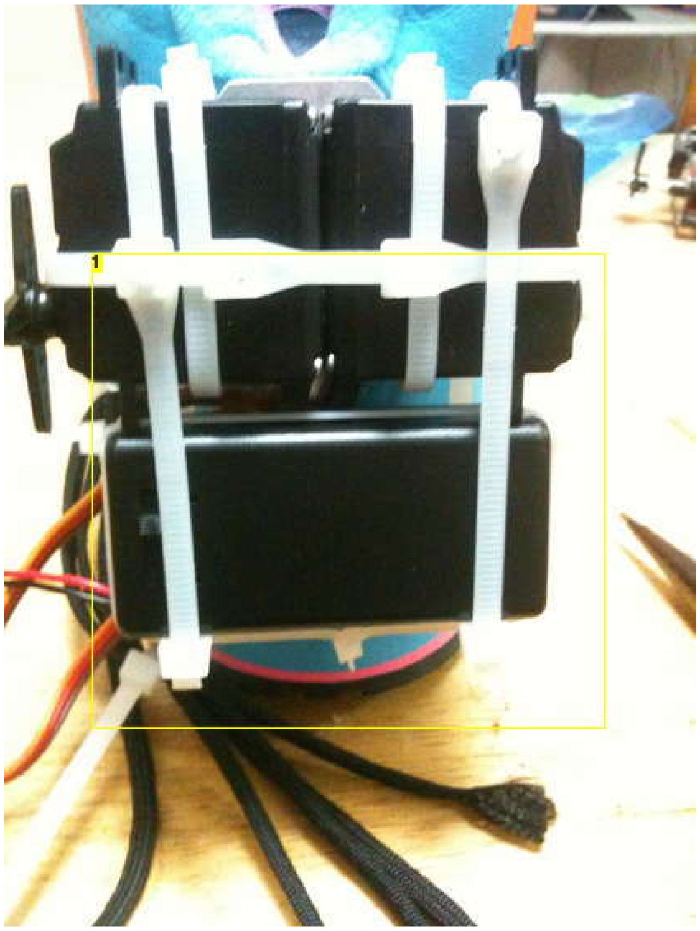


Image Notes
1. Just enough space here to mount the Arduino.

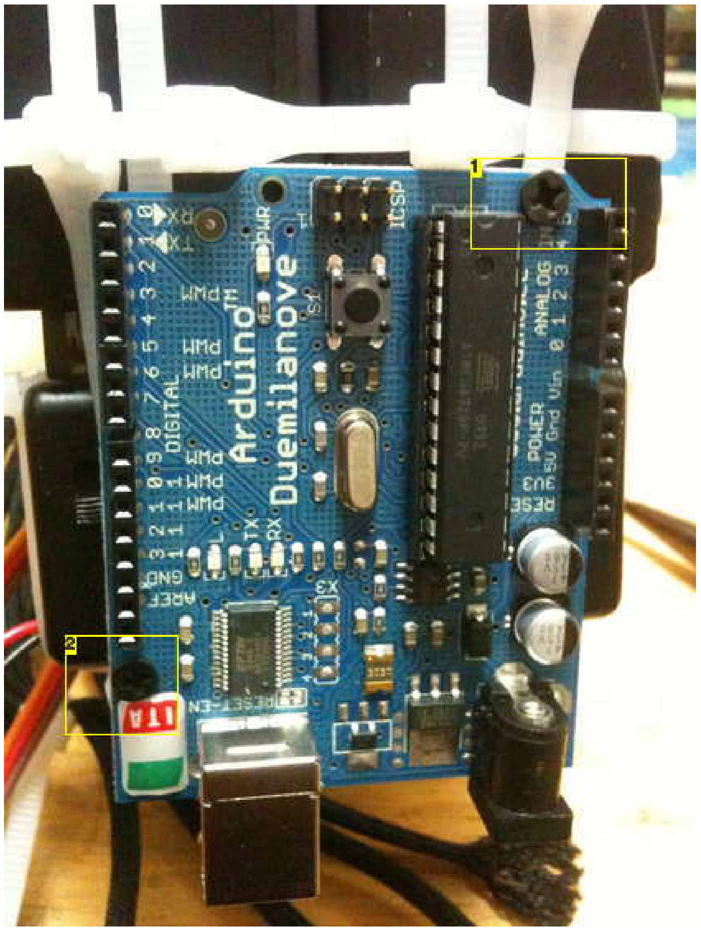


Image Notes
1. One screw
2. Two screw

Step 7: Adding some electronics to the motor shield

Is your soldering iron still warm? Time to add some of the bells and whistles.

As you can see in the pics, I tested the components and programing with a breadboard attached. Since I've done the heavy lifting you can skip this part, but it doesn't hurt to double check before everything is set in place.

After the program is uploaded (posted at the end of the instrutable) we can permanently mount the parts. First, we solder a resistor and to one pin of the LED and a length of wire to both pins.

That assembly is then pushed through an unused shoelace socket, and the wires are ran to the Arduino, using hot glue to keep everything in place and out of the way. **Make sure you know which of the wires goes to the positive pin of the LED!**

The Force sensor is mounted next. Soldering is not advised as the plastic may melt, so I wrapped some wire around the leads and hot glued them into place. The sensor was then glued and duct taped into the bottom of the shoe, right where my heel would rest.

The wires, also glued and taped into place, go up the back of the shoe and to the Arduino.

Finally, after we grab another resistor and a bit of medium gauge wire, we can begin soldering everything into place:

1. The positive wire of the LED goes to digital pin 2.

2. One force sensor wire (doesn't matter which) is soldered to +5v.

3. The other force sensor wire goes to Analog Pin 0.

4. Also connected to Ana. Pin 0 is a resistor. The other end of the resistor goes to Ground.

5. Also connect the negative LED pin to Ground.

6. A four inch spiral wrap of wire is soldered to Analog Pin 5. This is the touch switch- touching this wire firmly will cause the servo motors to move into the unlocked position.

7. Finally, plug in the servo motors, making sure to get the orientation right. My Arduino sketch assumes the left (looking from the back) servo is plugged into the right most servo pin, though this can easily be changed in the software.

The electronics are finished!

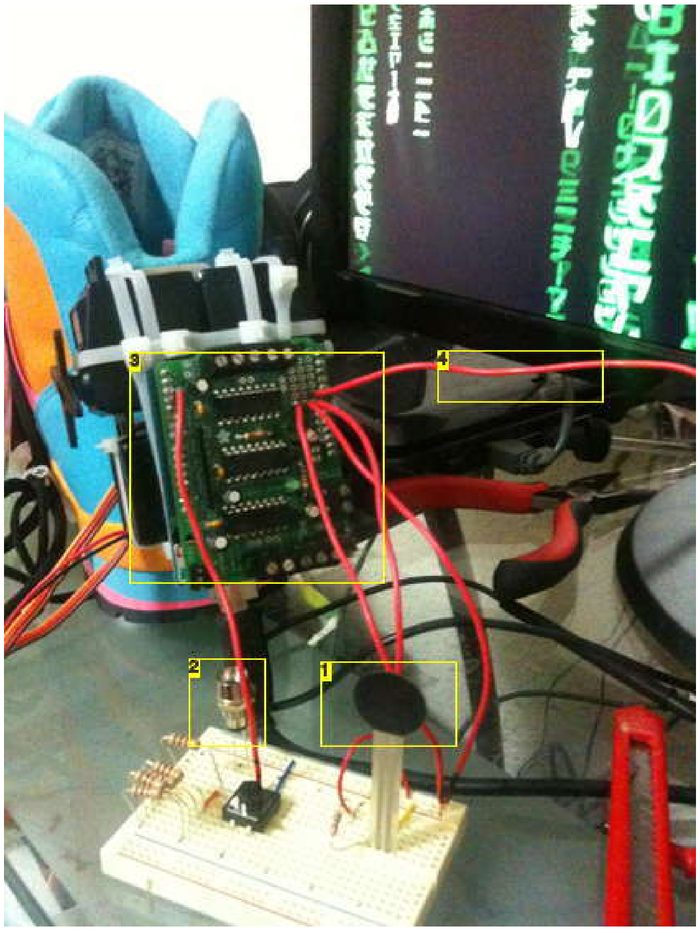


Image Notes
1. Force Sensor
2. LED (later removed the the metal shell)
3. Science!
4. Later wound up and turned into touch sensor

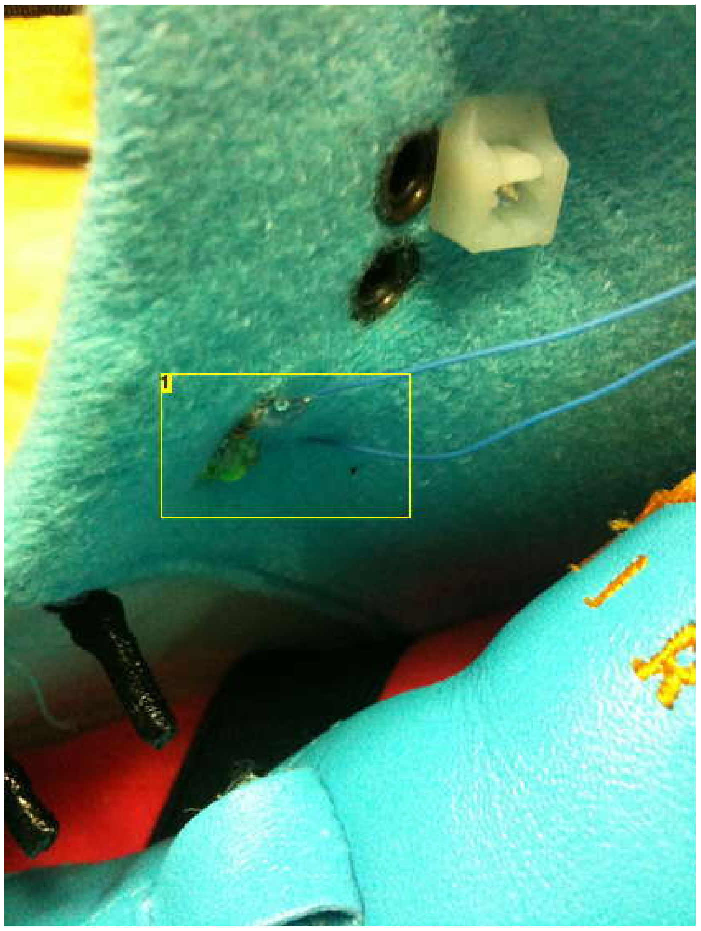


Image Notes
1. Added a little hot glue to protect the electronics.

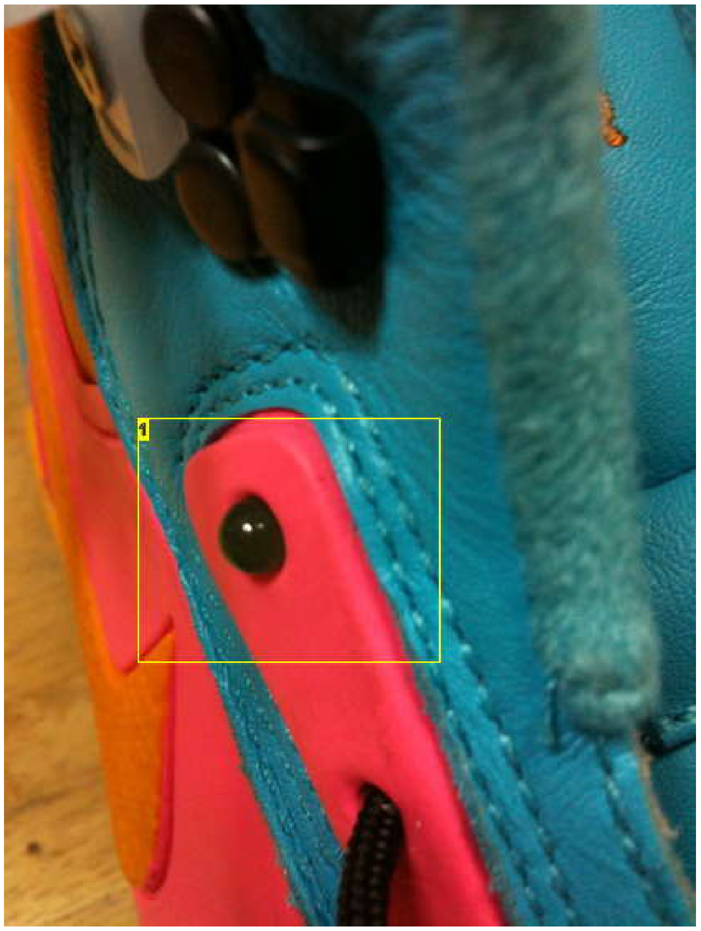


Image Notes
1. Lookin' subtle- nice contract to the rest of this contraption :)

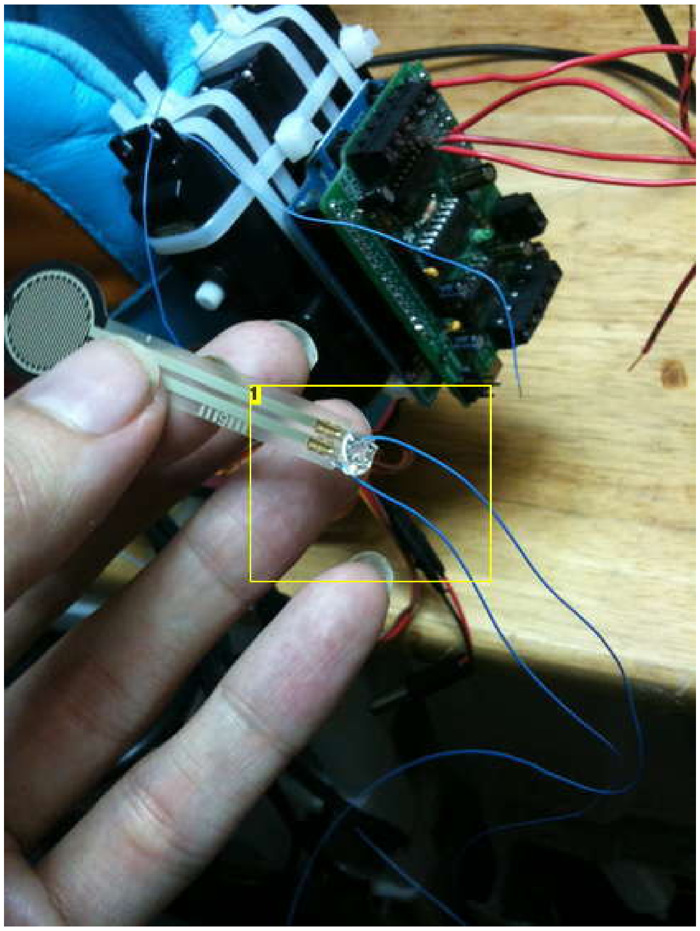


Image Notes
1. Force sensor with attached leads.

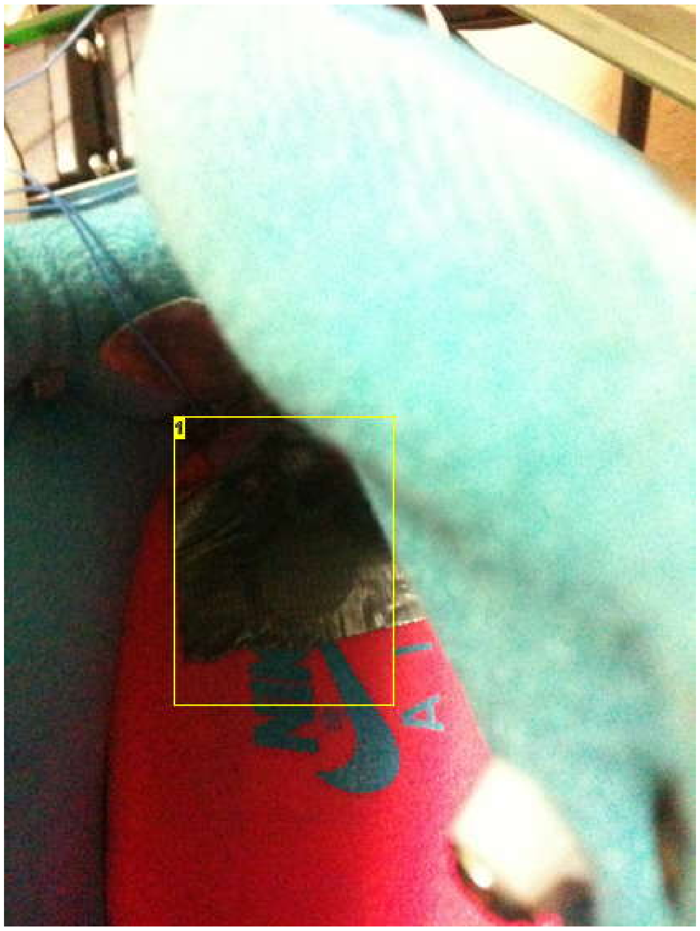


Image Notes
1. Duct Tape! (Covering force sensor)

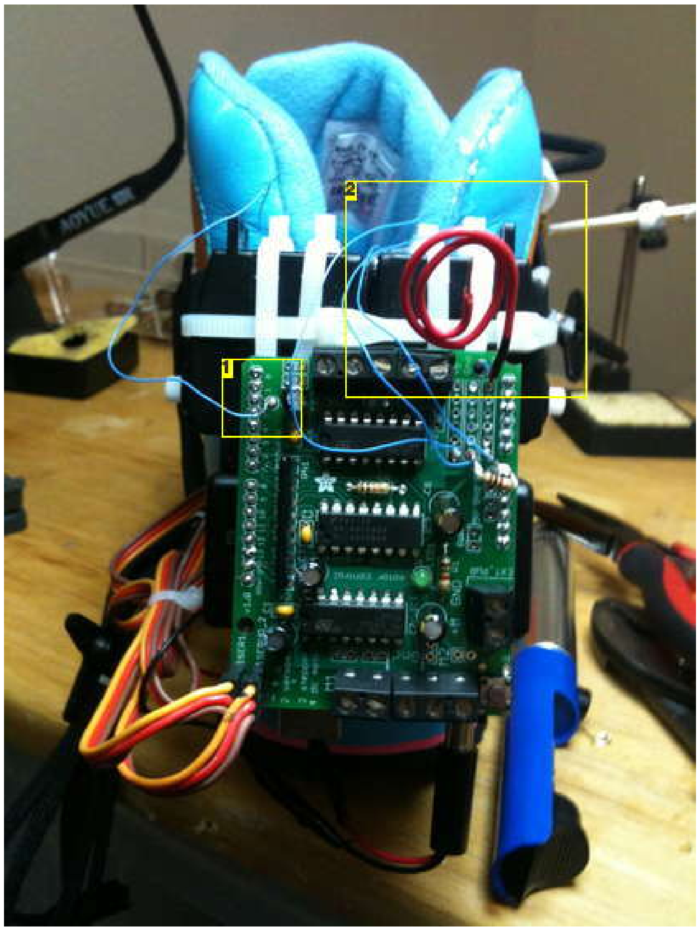


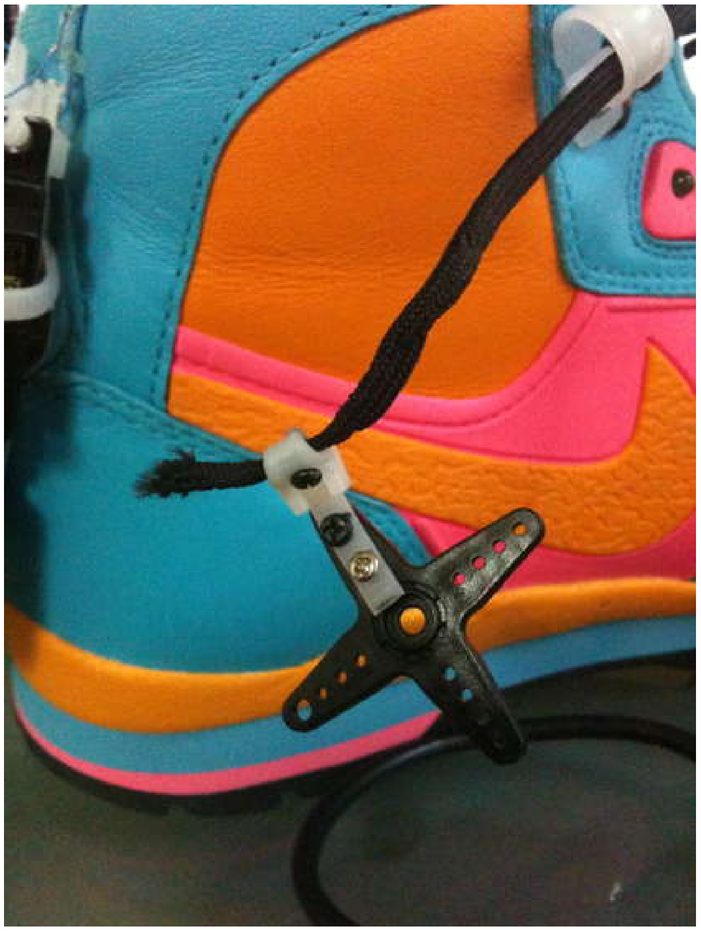
Image Notes
1. Positive LED wire to digital pin 2
2. I didn't have any working push-button switches, but I found that just touching the wire was enough to change the flow of current and could act as a switch by itself.

Step 8: Connect the Laces to the Servos

Now we just have to connect the laces to the servos and the hardware portion of the project is finished.

This usually requires a lot of little adjustments to get it right- you want enough slack to get your foot comfortably in the shoe, but when it laces up you want it to be visually appealing.

I used zip ties to secure the laces, and attached them to the servo arms, making sure the servos are in the "unlaced" position. The sketch will turn the servos 180 degrees from the starting point, tightening the laces.



Step 9: Upload the Arduino Sketch

Plug the USB cable into the Arduino and the computer, and open the .pde sketch with the Arduino IDE. Make sure you have download the motor shield libraries as detailed in it's instructions.

Click 'Upload' and within a few seconds it'll be transferred and the shoe is now ready for use!

Just a caveat, this is my first time working with the Arduino and programming in general, so I'm sure the sketch is nowhere near optimized and could use some fine tuning. Feel free to play around with it!

File Downloads

Power_Laces_6_28_550AM_1_0.pde (2 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Power_Laces_6_28_550AM_1_0.pde']

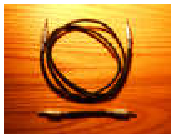
Related Instructables

DIY Elastic Shoelaces by Cheryl - SewCanDo

Paracord Shoelaces by Wasagi

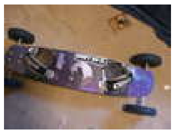
What to do with your shoelace knots! by thatskindacool

Make a better cord tie by Closer


Retro stereo patch cables from a USB cable by Winged Fist

Paracord Shoelaces with aglets by hirod3

Alternative Ways to Tie Your Shoes by lawizeg


Mountainboard Heelstraps by potato413

Plantduino Greenhouse

by **clovercreature** on June 12, 2011

Intro: Plantduino Greenhouse

UPDATE 7/9/11: The AC power fed relay has been replaced with a DC battery fed relay system as shown in step 10.

UPDATE: We have been selected as finalists in the microcontroller contest! Thank you for voting and rating. Thank you also for all the feedback on the safety of out relay system. We hope the new instructions are clear. We will be continually updating as we make progress on the new design.

Hello Everyone!

My name is Clover and I am in love with vascular plants and robots.

This summer I wanted to combine my two loves of plant science and engineering. Thus I am constructing my very own greenhouse in my backyard. I am an undergrad, and as any former student knows, this means I move around constantly, and I am not always around to take care of my vegetable garden. I love my plants but since I am moving back to school in July, and my family is unreliable, I need a way to make sure that they are taken care of. Enter Arduino!

I have constructed an automated watering and temperature system. This includes sensors that will turn the systems on only when needed. This is essential when the ever-changing New England weather demands some intelligence in watering and heating patterns.

This is my first project using an Arduino so I am using wonderful articles from MAKE and Instructables as very helpful templates. Already the Instructables, MAKE, and Ladyada blogs have been ridiculously helpful so, worry not biology nerds, you too can show the engineers just how awesome we are!

Step 1: Plant Science 101

One facet of this project is to grow my own vegetables and do some scientific experiments.
Warning! Science...
Sources: Much of this information/ images came from Northeastern Univesity Professor Donald Cheney's Plant Science lecture and the textbook Botany which is linked in the more information section.

Greenhouses
There are a lot of reasons that I am building a greenhouse. Greenhouses are a really cool way to grow larger and healthier plants faster and artificially extend the growing season. Greenhouses work by using a transparent airtight cover to trap in light and moisture to create a mini- ecosystem that is separate from the environment around the greenhouse. Heat is generated both from the sun’s rays that penetrate the greenhouse but do not escape as well as the trapped heat given off by the plants during their biological processes such as photosynthesis. This results in a very fascinating microclimate. This general idea of a layer of material trapping in heat and increasing the climate below is why sometimes global warming is called “the greenhouse effect” by people who like to oversimplify complicated climate phenomena.

Greenhouses can be made of glass or plastic. They create a controlled microclimate that makes experiments and procedures such as grafting or tissue cultures easier to perform.

Plant Anatomy and Physiology
Vascular Plants (plants that have stems and roots) develop mostly from seeds (a lot of nonvascular plants like mosses and ferns develop from spores which work a little differently). A seed consists of three types of tissues. The epidermal tissue is the outer layer for protection against the elements. This is usually embodied in the hard seed coat (Think the hard shell around sunflower seeds). The middle layer is called the ground tissue. The ground tissue is where photosynthesis takes place. The vascular tissue is in the very middle of the plant body and is where nutrients and water are conducted and stored. The roots are the first thing to grow out of the seed. The roots are made up of the same tissues except on the tips of the root is a number of epidermal cells called the root cap that are made to die and be ripped apart as the root burrows through the ground. The roots provide water and nutrients to the plant. Sometimes nutrients can be stored in the roots such as in the case of root vegetables like carrots or turnips.

The stem of the plant grows from what is called the coteledon in the seed. Longitudal growth is initiated by the apical meristem which is the primary growth bud. Lateral meristems are responsible for making the plant larger in diameter. There are two structures called the auxillary buds that grow on both sides of the main meristem. In case the axial bud is cut off for some reason the surrounding auxillary buds take over and grow the plant in a new direction. This is how pruners reroute tree growth by cutting off certain branches.

NOTE ABOUT WEEDING: This anatomical knowledge is very useful when you are weeding- particularly with grass. Grass is a special category of plants called a monocot. With grass the meristem is on the bottom of the grass blade which is why it grows even if it is cut. So do not just rip out the grass blade but make sure you get at the root system too.

Gardening: Plant Nutrients

Even though most plants grow well with just old fasion dirt and water there are a lot of other things that they need to grow and produce good fruits. These are broken up into two catagories: **Macronutrients** which take up more then .5% of a plant's dry mass and **Micronutrients** which are only present in trace amounts. Some macronutrients that are essential for all plants are Nitrogen, Magnesium, and Sulfur . Sulfur is in proteins and vitamins. Magnesium is in the chloraphyll which are involved in photosynthesis. All of these are found in the soil. Plants are also about 45% Carbon which comes from the CO_2 in the air and 45% Oxygen which comes from water and air. Some micronutrients that plants need are Boron, Chlorine, Manganese, Iron, Copper, and Zinc. All of those are naturally occuring in soil but the most important one is Iron. Iron deficiencies lead to a yellowing of the leaves. Fertilizers usually provide all the minerals needed for a plant to survive. The ratio of Nitrogen, Phosphorus, and Potassium are listed in that order on most fertilizer bags in ration form. For example: 10:20:10= ratio of N:P:K.

Photosynthesis
Photosynthesis is how plants turn sunlight into sugars. The chemical equation is $6CO_2 + 6H_2O$ --> sunlight--> $C_6H_{12}O_6 + 6O_2$. Photosynthesis happens in two stages: The dark and the light. First is the light stage where , as the name implies, the sunlight is needed. What happens is electrons are taken from the water molecule and excited in photosystem 2. Then the electrons are transported down what is called the electron transport chain. This chain is nothing more then a series of oxidation and reduction reactions that progressively bring the electrons down into a less excited state. Once the electrons hit photosystem 1 they are excited again and go down another smaller electron transport chain. While the electrons are moving from a more excited state to a less excited state they are also turning a substance called NADP+ (Nicotinamide adenine dinucleotide phosphate) into NADPH. This is used in the dark stage, also known as the calvin cycle. The Calvin Cycle is a series of modifications starting with a reaction with starting material RuBP (Ribulose bisphosphate) and Carbon dioxide using a series of enzymes and redox reactions. It is called a cycle because after the sugar is produced the starting material RuBP is again synthesized. For every round of the carbon cycle there is 1 sugar derivative output and 3ATP (the source of our life's energy). So it takes 6 rounds of the Calvin Cycle to create one glucose molecule. There are a lot of plants that utilize different variations of this cycle.

pH
Part of my experimentation is to come up with a low tec pH monitor for the soil. pH measures relative acidity by taking the log of the concentration of hydrogen molecules present. pH is measured on a 14 point scale with 1 being very acidic and 14 being very basic. Water is neutral or 7. I measured the acidity of my soil with an at home pH kit that I bough on amazon for $5. The acidity of my soil was about 6.5 which is perfect for the types of plants I am growing (strawberries, peas, basil, broccoli). If there is a problem with your plants and the fertilizer is fine I would suggest checking the pH.

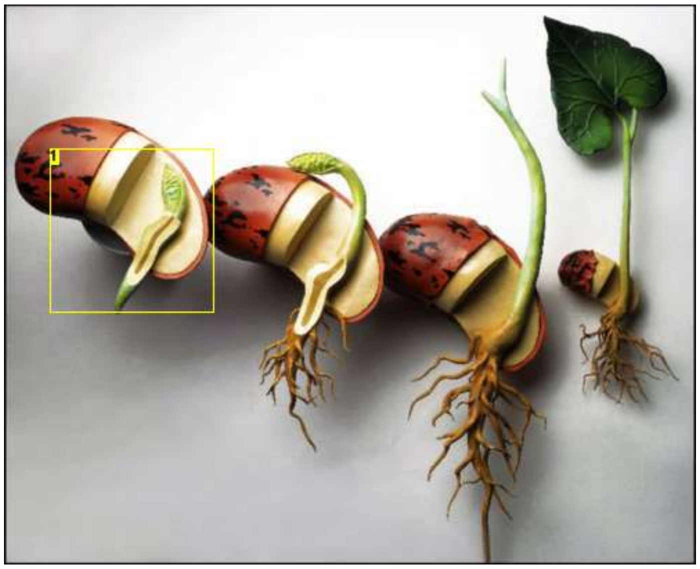


Image Notes
1. first thing to develop is the roots

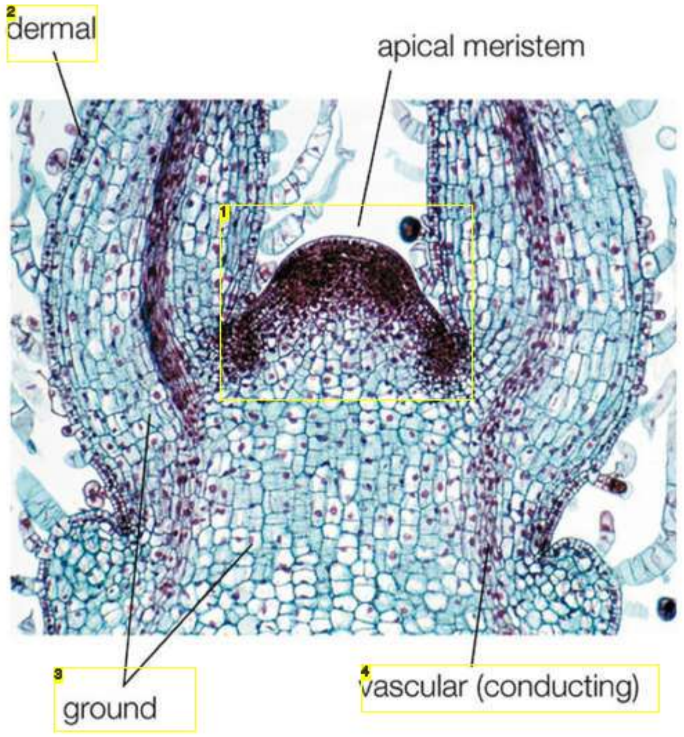
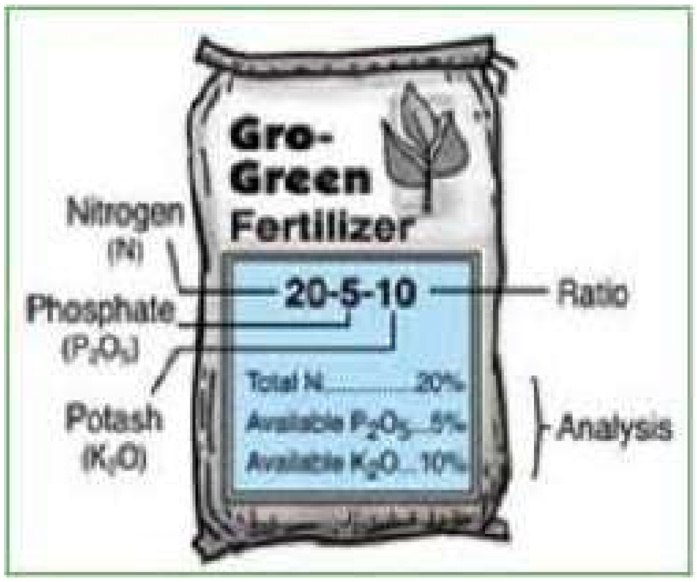
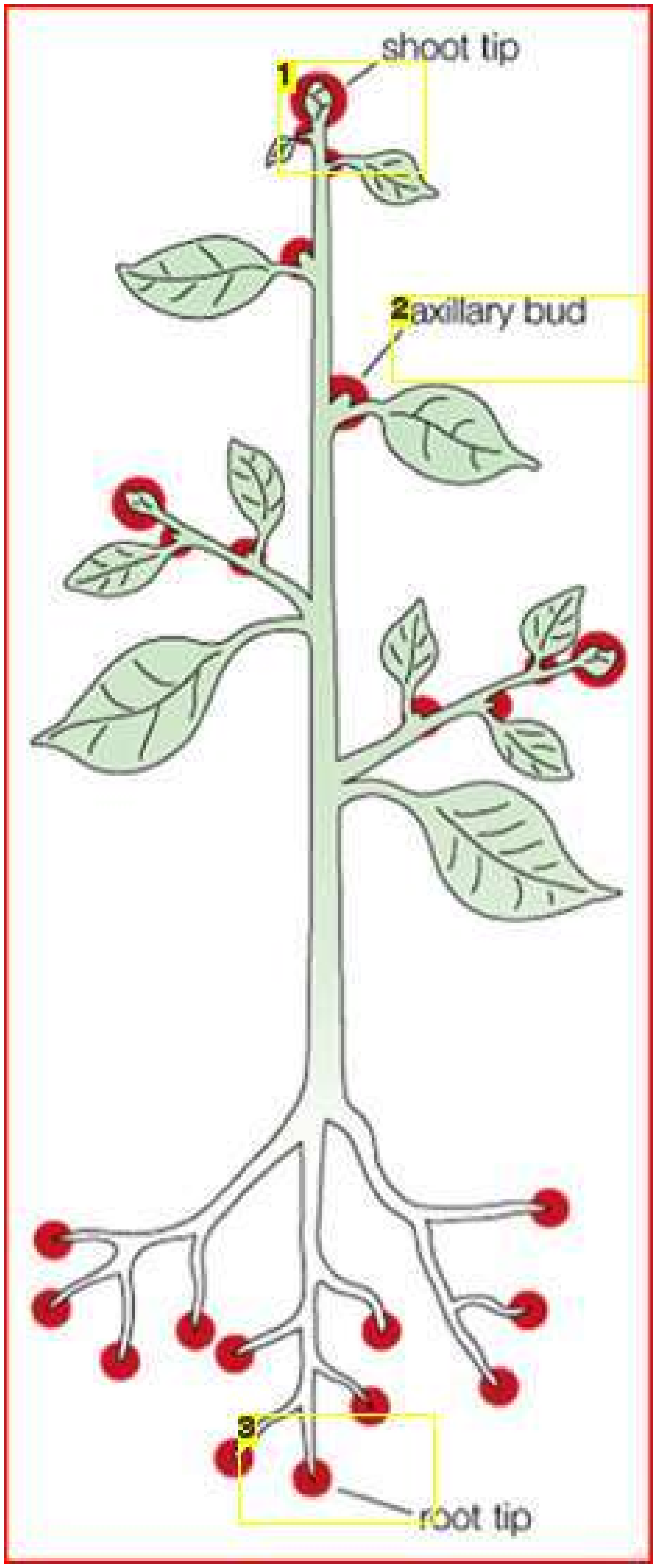


Image Notes
1. Where the growth happens
2. or epidermal tissue
3. where photosynthesis takes place
4. Where the water and nutrients are stored and moved around throughout the plant





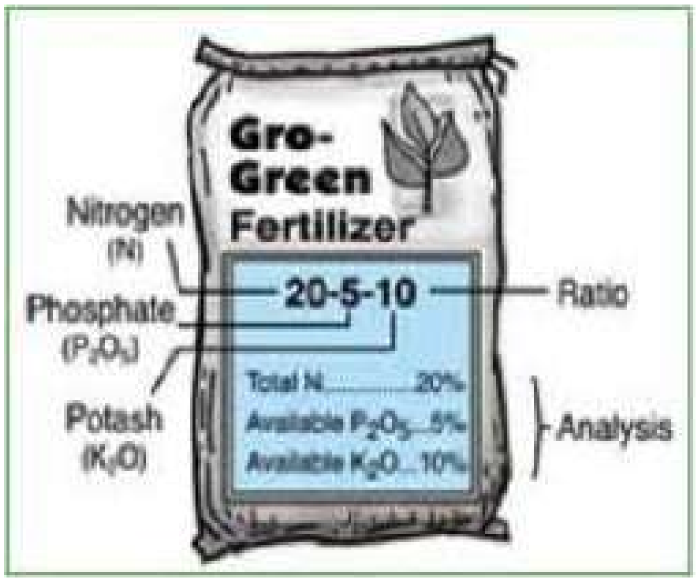
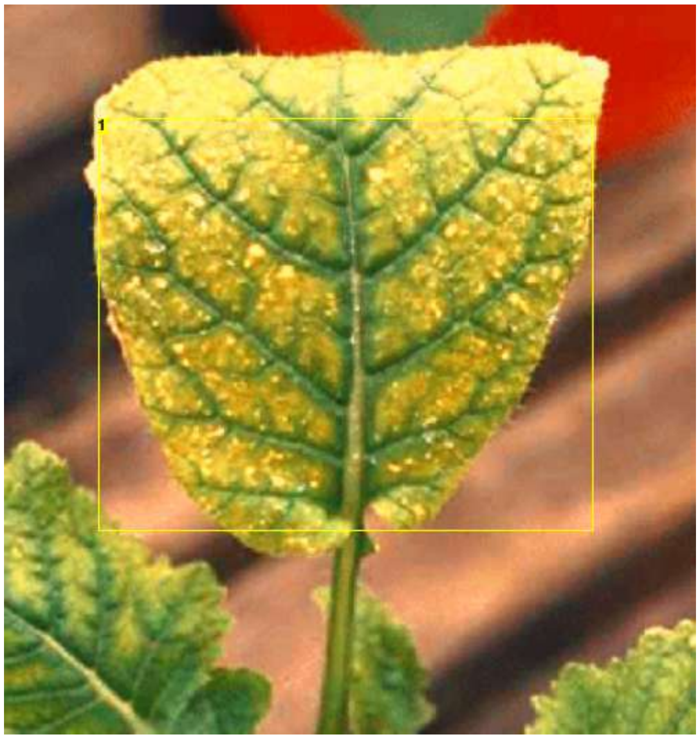


Image Notes
1. Apical Bud: Main longitudal growth
2. main latitudinal growth
3. Growth in the roots



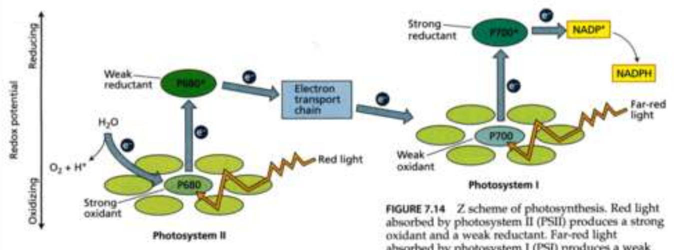
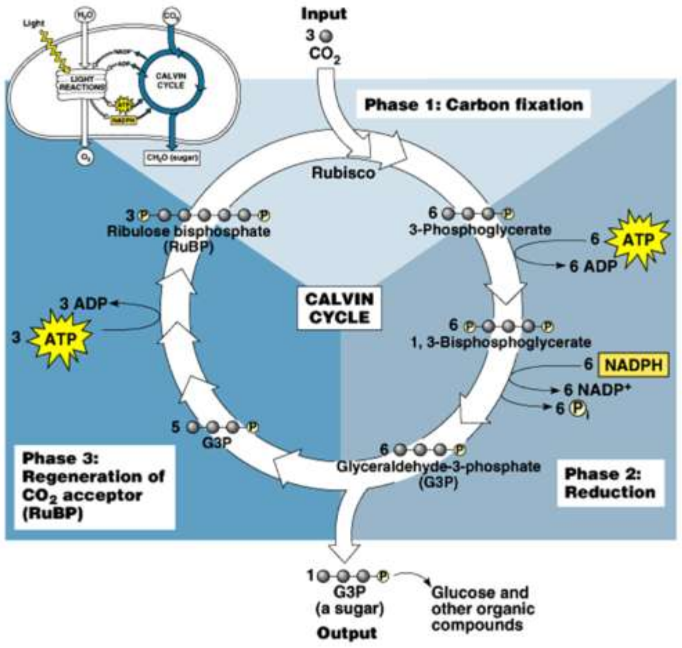


FIGURE 7.14 Z scheme of photosynthesis. Red light absorbed by photosystem II (PSII) produces a strong oxidant and a weak reductant. Far-red light absorbed by photosystem I (PSI) produces a weak oxidant and a strong reductant. The strong oxidant generated by PSII oxidizes water, while the strong reductant produced by PSI reduces NADP$^+$. This scheme is basic to an understanding of photosynthetic electron transport. P680 and P700 refer to the wavelengths of maximum absorption of the reaction center chlorophylls in PSII and PSI, respectively.

Image Notes
1. Iron Deficiency



Step 2: Build a Garden/ Plant Seeds

My garden is in a raised bed. Raised beds are when you plant your garden in a wooden frame that is above ground level. The nice thing about raised beds is that they are easier to maintain because they can block weeds. Also, you fill it with whatever soil you have instead of the soil that you are blessed to be living on. In the case of my New England house, the infamously horrible soil makes a raised bed the obvious choice.

Building a Raised Bed
To build a raised bed, dig out a patch of land the size you want your garden to be. Then build a wooden frame that fits the perimeter of your garden. A cool trick is to put some burlap or pebbles on the bottom-most layer of your garden. This way the grass won't grow back up through your garden. Putting down a bottom layer is not necessary and I did not do that in my garden. However, if you have the time, it will save you a lot of work later.

I have provided a link to a more step by step format for more specific details. I am doing this because the making of the garden was done the previous year before the greenhouse project. Here is a great website for these steps:
http://www.thisoldhouse.com/toh/how-to/intro/0,,1615067,00.html (this provides instructions for a slightly more elaborate garden than I built).

Soil
The mixture of soil that I use is a mixture of garden soil and peat moss. I also mix in some miracle grow potting soil. Dump a large lump of each soil into the garden and then mix it up with a large shovel or trowel. After it is mixed, spread it out over your garden evenly. This would be a good time to test the pH of your soil mixture. It should

be between 6.5-7.5. You can test this by purchasing a pH soil tester. I bought the Luster Leaf 1612 Rapitest pH Soil Tester from Amazon for about $5. It is not the most precise way to measure pH, but it gives you a nice range. Also, this test has a chart with what to add if your soil is too basic or acidic.

Seed Planting
Planting seeds is very easy. In all cases, read the back of the packet your seeds came in. There should be a chart with depth and spacing requirements for that particular plant. Absolutely follow those spacing requirements or you are going to have a lot of trouble down the road. Also, only plant one or two seeds per spot. I made the mistake of planting a bunch of strawberry seeds in one plot (the seeds were so small!!!) and I have about 10 tiny strawberry plants that can't get any bigger because they are entangled in each other and there are not enough resources for them all. Right after you plant them, make sure the soil is thoroughly moist. Watering is important and for most plants, should be daily. However, the plants are delicate, especially when they are developing stem systems, so water carefully.

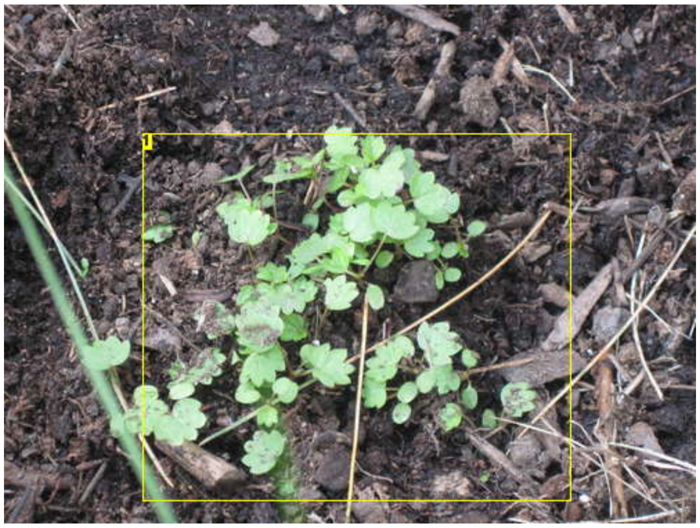


Image Notes
1. This is what not to do. I planted a whole bunch of strawberry seeds and now I have a whole bunch of strangled strawberry plants

Step 3: Build a Greenhouse: Step 1 materials

-two 10 foot long, ½ inch diameter PVC pipes

-three 40 inch long, ½ inch diameter PVC pipes

-Roughly twenty-five 6 inch long Zipties

-at least 9 by 12 feet painters clear painter's plastic tarp 3mm thick

-Waterproofing Tape

-Duct Tape

-Industrial VelCro

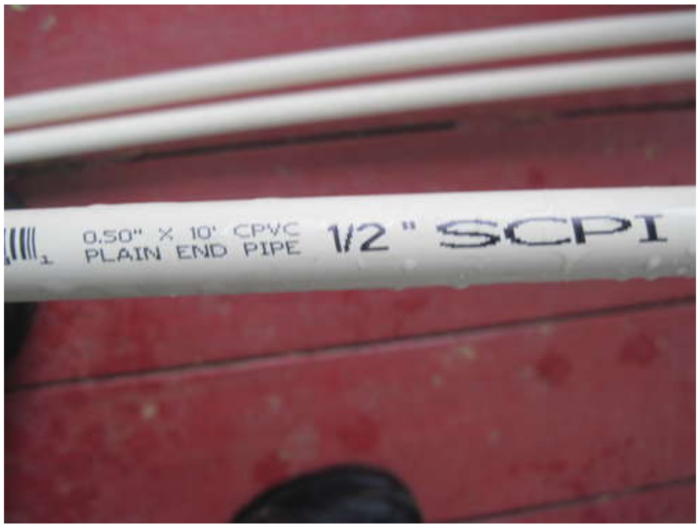
Tools:

Hacksaw

Scissors

Staple Gun

Measuring Tape



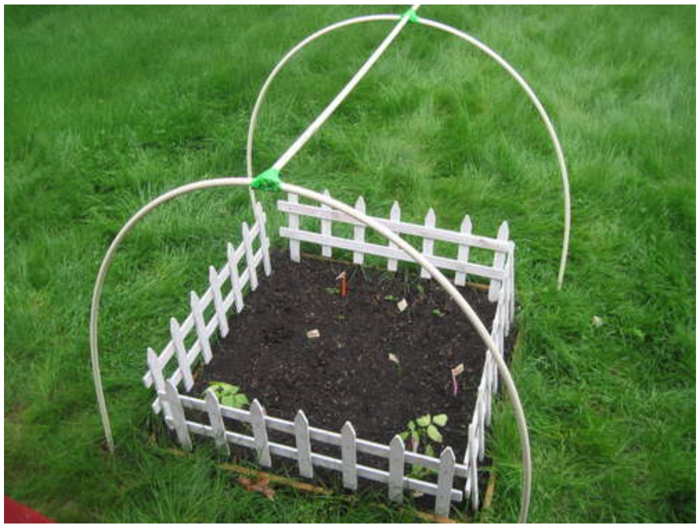
Step 4: Build a Greenhouse: Step 2 Build the Frame

1) First we measured my garden which is about 1 square meter (tiny I know). Then we used complex integration and approximation to measure the arc length of the frame. Just kidding. We just used a tape measure to approximate and then just used the 10 foot measurement that the PVC came in.

2) We used the hacksaw to cut the ends of the PVC pipe at an angle so that they could be easily stuck into the ground. To do this, start about 3 inches from the bottom of the pipe and cut away from yourself at an angle.

3) Stick both ends of the pipe into the ground at opposite sides so that it makes a nice arc. We placed one arc behind and one in front of the garden. We measured three inches to the left and then another three inches away from the garden corners.

4) Next we placed one 40 inch PVC across the top in the center of the arc. We duct taped the ends to the arc.



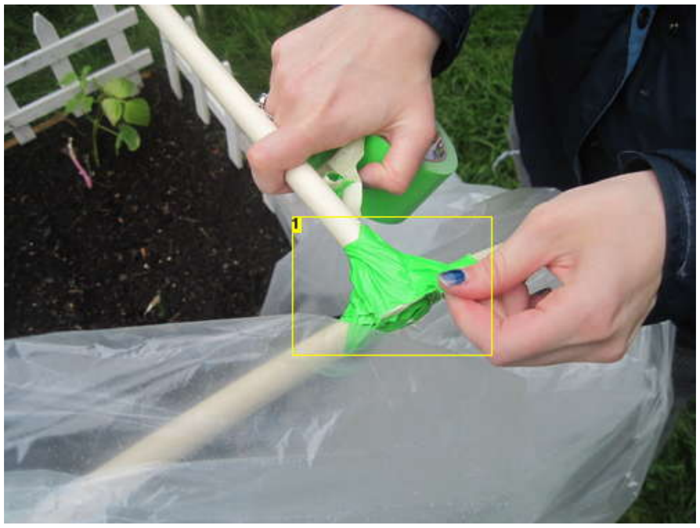


Image Notes
1. duct tape the joints of the PVC bar

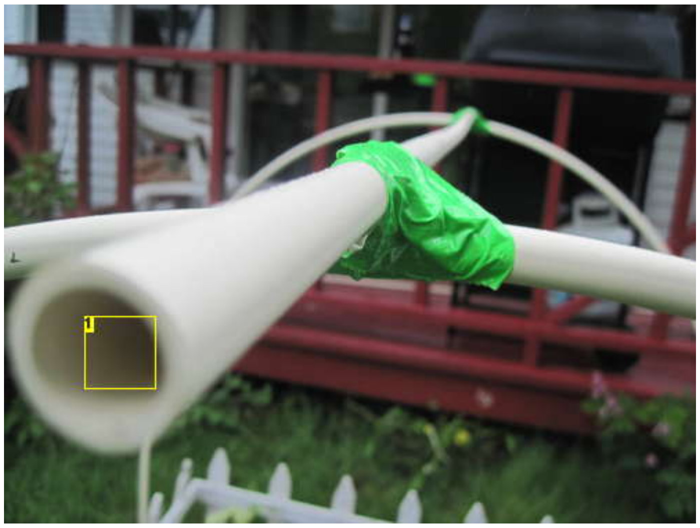
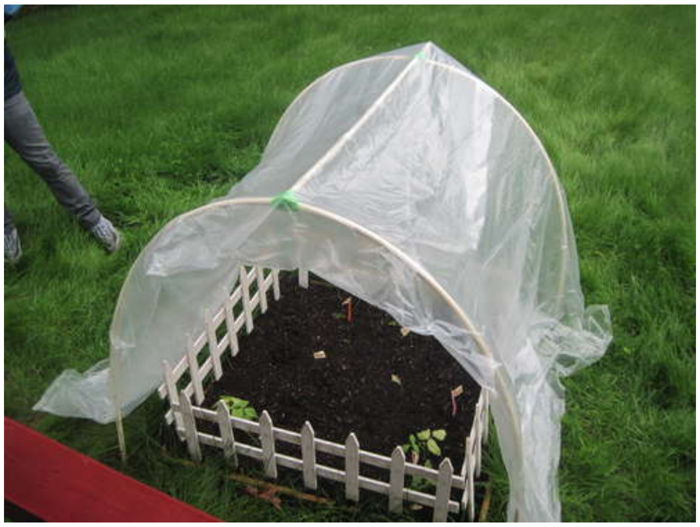
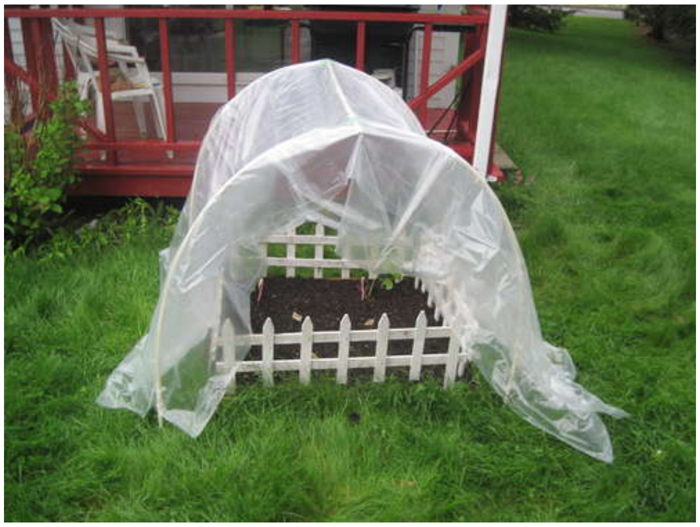


Image Notes
1. This overhang is not essential. It was just a cosmetic choice.

Step 5: Build a Greenhouse: Step 4 Lay the plastic

Next we cut a large portion of the plastic to cover the frame. In all honesty we gestimated the size. Once we draped it over the greenhouse, we secured it with zipties and cut off the excess on either side. We needed two people for the zipties because one person had to keep the side taught the entire time. We secured the plastic to wooden frame of the garden using the staple gun.





Step 6: Build the Greenhouse: Step 5 Add the back and the door

To make the end pieces we cut a small piece of the plastic and then draped it over the frame making sure it could touch the ground. Then we simply just ziptied it to the PVC frame, making sure the plastic was held firm.

As we ziptied the back wall we held the plastic as tight as we could. For the front, we left some slack so that the doors could close all the way.

Finally we cut a slit in the front to make the doorway and secure the tear with pieces of duct tape.

We lined the duct tape with the industrial Velcro so that the door can be opened and closed securely.

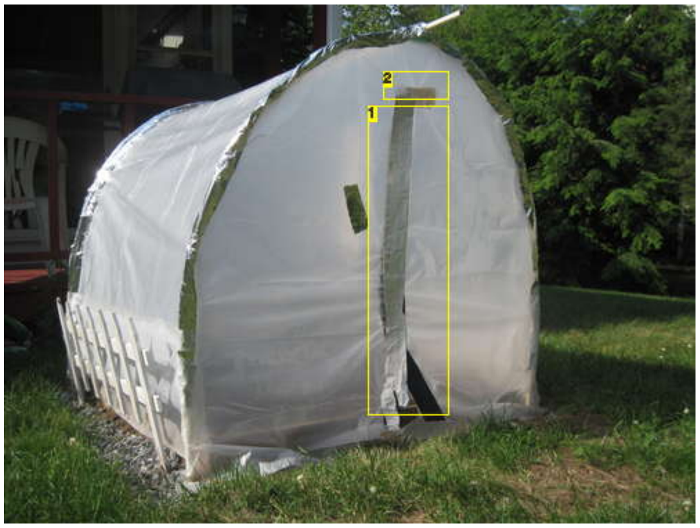


Image Notes
1. We made closed the flaps with industrial grade velcro
2. Make sure to put duct tape over the cut in the plastic so that your door won't continue to rip

Step 7: Build the Greenhouse: Step 7 Make it airtight/waterproof

To seal the sides of the greenhouse, wrap the waterproofing tape around the perimeter of the frame so that there are no openings in the plastic.

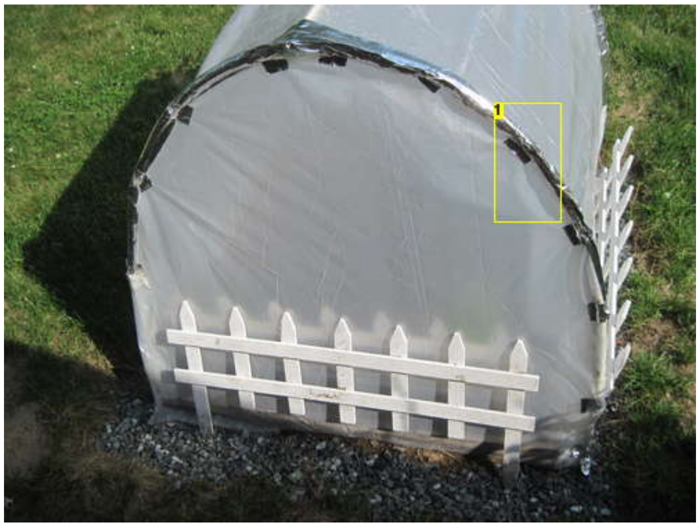


Image Notes
1. We also put this waterproof tape over the holes from the zipties.

Step 8: Build a Greenhouse: Step 8 Dig a Trench

Around the perimeter of the garden dig up the grass so that it does not encroach on the garden. I filled the trench with blue stone. Then we dug another small narrow trench so that we could put the wires that connect the sensors to the birdhouse underground. We needed to do that because when my dad mows the lawn any loose wire can be easily caught. We put the wires through PVC pipe to protect them from any sort of natural damage (insects, water, etc). We then buried the PVC pipe in the ground from the deck to the greenhouse (about a yard total).

Image Notes
1. The trench is about 10" wide and 5" deep. The specs don't matter as much as the barrier from the ever encroaching grass. It took two bags of bluestone to fill our trench.

Step 9: Watering System: Step 1 Materials

Parts:

Outlet box

Outlet (2 plugs for water and heat or four plugs for water, heat, lighting, and fan)

120v wall plug (can be hacked from pretty much anything. This one was taken from some computer speakers.)

14 gauge wire

5 volt relays (same number as number of individual outlet plugs)

22 gauge wire

solenoid valve

wall wart for solenoid valve (ours was 12 volts)

Screwdriver

Wire cutters/ Strippers

Glue Gun

For testing:

Arduino

Breadboard

(optional) small LED light

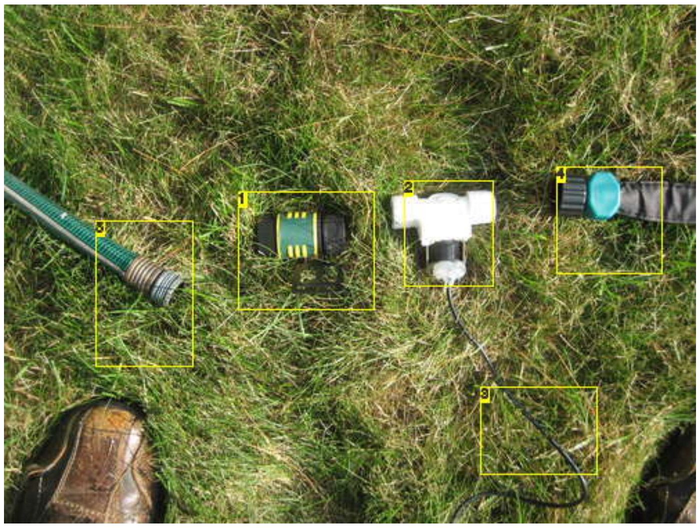


Image Notes
1. Hose connector
2. Valve- you may need to add plumbers tape to the threads so that the nozzles fit on as tight as possible.
3. cord to the Wall Wart
4. Sprinkler Connection
5. Hose connected to water source

Step 10: Watering System: Step 2 Build a Relay Box

Originally the valve and relay system was controlled by AC power from the wall. The breadboard layout below shows how to connect the valve and relay system using dc power from a battery. The battery is over 12v and the regulator is a 12 volt regulator. The solenoid represents the solenoid valve. The relay is activated by 5 volts. The 12 volts coming out of the regulator go to the arduino power and the relay. The blue wires coming from the relay attach to the arduino ground and a digital pin which control the relay state.

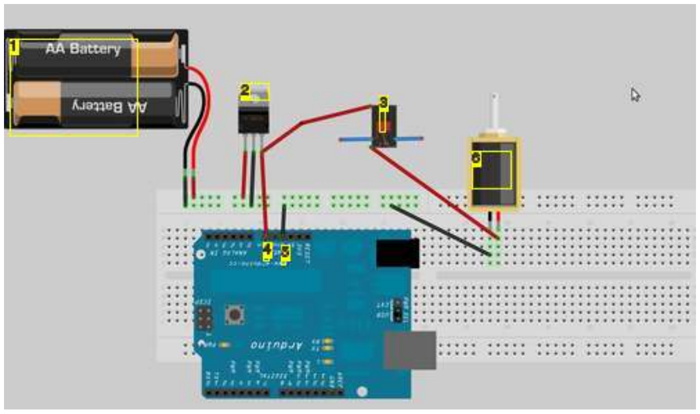


Image Notes
1. Battery over 12 volts
2. 12 Volt Regulator
3. relay
4. V in
5. Ground
6. Solenoid valve

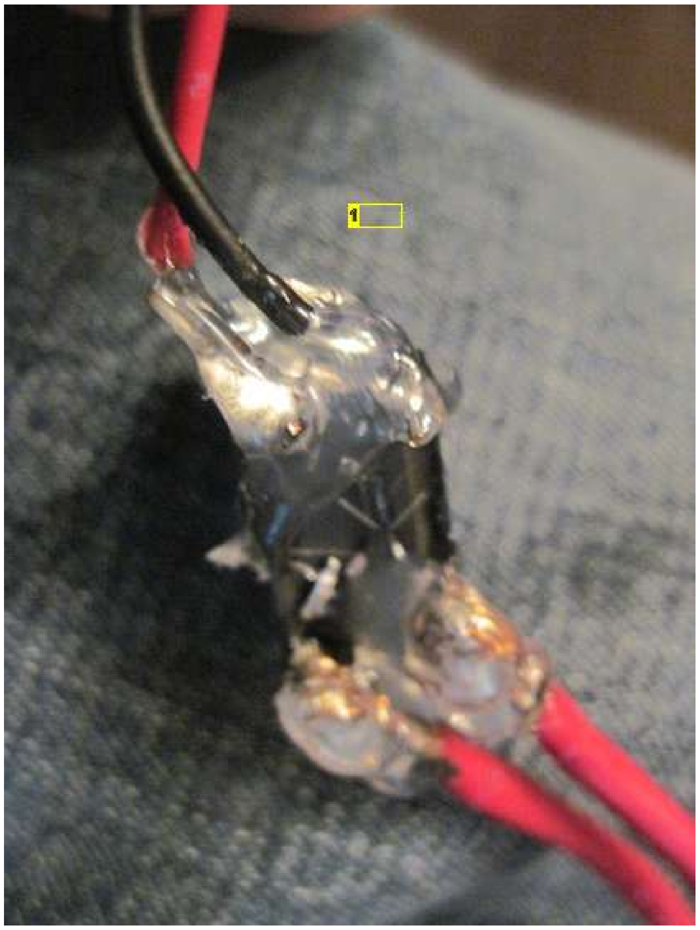


Image Notes
1. Hot glue every exposed connection. SERIOUSLY!

Step 11: Watering System: Step 3 Connect the Valve

You need to connect the valve to the relay box so that the valve will receive power and control the flow of the water. To do this you connect the valve to a 12 volt wall wart. The wall Wart can then be directly plugged in to the relay box.

1) cut the jack of the wall wart off with wire cutters
2) strip back the plastic to expose the red and black power and ground wires
3) Strip the black and red wire so that they can wrap through the holes in the valve solder terminals.
4) Solder the red wire to one of the terminals and the black wire to the other terminal.
5)Test to make sure it works
6) Hot glue the solder joints so that the electricity won't electrify things it's not supposed to.
7)Plug the wall wart into the relay box!

Step 12: Watering System: Step 4 moisture sensors

I first soldered a wire to each of two galvanized nails. This took a couple of tries but the secret is to scrape off some of the galvanized coating from where you want to solder with a knife. This helps the solder stick to the nail.

Next, hook the nails up to the breadboard and the breadboard to the arduino. Check out the pictures for a diagram. We connected the nails to analogue input 0 and used the pin 13 LED as an output.

Then be joyful and celebrate.

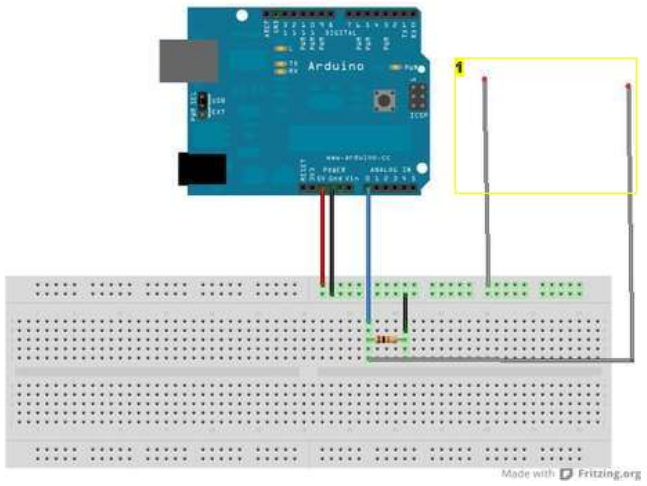
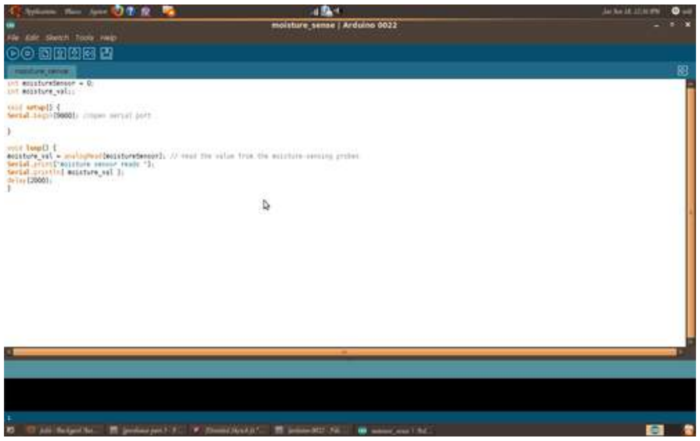


Image Notes
1. these are the nails

Step 13: Watering System: Step 5 Write the Code

```
int moistureSensor = 0;
int moisture_val;


void setup() {
Serial.begin(9600); //open serial port
pinMode (13, OUTPUT);
}

void loop() {
moisture_val = analogRead(moistureSensor); // read the value from the moisture-sensing probes
Serial.print("moisture sensor reads ");
Serial.println( moisture_val );

delay(1000);
if (moisture_val < 600){
 digitalWrite (13, HIGH); //soil is too dry- turn on LED
}else{
 digitalWrite (13, LOW); // soil is saturated- turn off LED
}

}
```

Test the nails using dry, perfect, and water saturated soils. You will want to calibrate your soils to your own watering habits and garden needs. These numbers are completely dependent on your own nail specs though.



```
int moistureSensor = 0;
int moisture_val;

void setup() {
Serial.begin(9600); //open serial port

}

void loop() {
moisture_val = analogRead(moistureSensor); // read the value from the moisture-sensing probes
Serial.print("moisture sensor reads ");
Serial.println( moisture_val );
delay(2000);
}
```

Step 14: Watering System: Step 6 Bring It All Together

We had to buy a nozzle that connected the solenoid valve to the hose. The irrigation drip tubing we bought at home depot attached perfectly to the other end of the valve. You can make your own irrigation tubing by buying some plastic or rubber hose and poking holes into it. Plug the 12V wall wart into the relay box. Make sure you use the socket that is connected to the right arduino ports. For our relay box the top outlet goes to the bottom set of wires and the bottom outlet goes to the top set of wires.

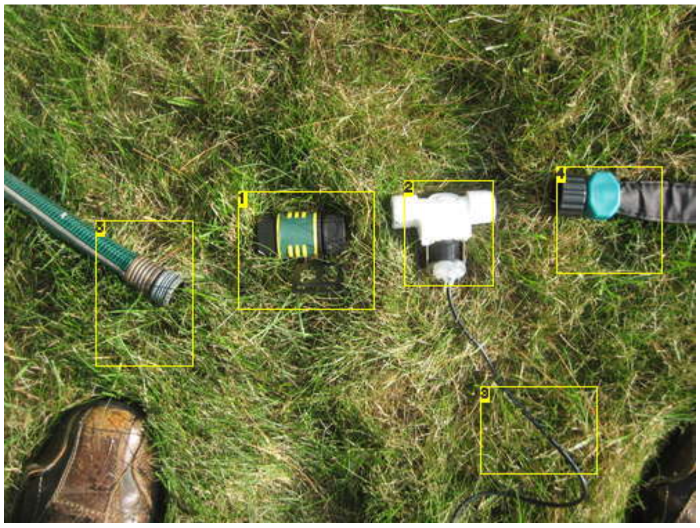


Image Notes
1. Hose connector
2. Valve- you may need to add plumbers tape to the threads so that the nozzles fit on as tight as possible.
3. cord to the Wall Wart
4. Sprinkler Connection
5. Hose connected to water source

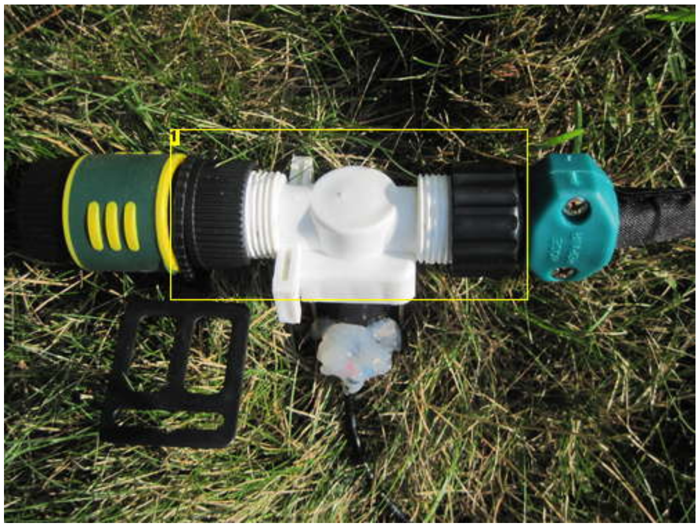


Image Notes
1. Attach components so that the arrow on the valve is pointing towards the sprinkler system.

Image Notes
1. we used zipties to attach this part to the deck so that the wall- wort can easily reach the relay box in the birdhouse

Image Notes
1. place the moisture sensors in a place that is representative for the entire garden

Image Notes

1. make sure to snake the irrigation tubing in the garden so that all the plants are watered equally

Step 15: Plantduino: Step 1 materials

1. perfboard

2. ribbon cable

3. 16 by 2 lcd

4. 10k potentiometer or patience and a resistor

5. female barrel jack

6. 7805 power regulator

7. (2) 10uF capacitors

8. Atmega 328 with arduino bootloader preloaded onto the chip

9. 28 pin dip socket

10. (7) 2-pin 5mm pitch screw terminals

11. (4) diodes

12. 22 gauge solid core wire

13. ftdi serial to usb break out board

14. 16 MHz crystal

15. (2) 22 pF (that’s picofarad not microfarad) capacitors

16. 100 nF capacitor

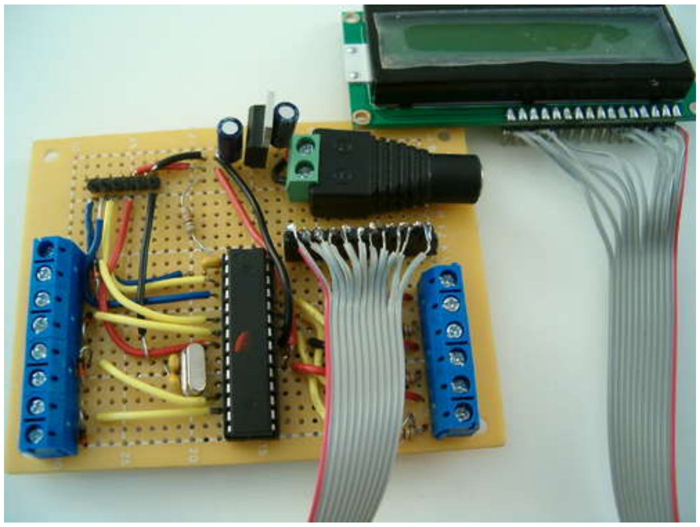
17. (4) 10k resistors

18. (11) female pin headers

19. (17) male pin headers

20. also, breadboards never hurt

(you will also need sensors but these are not technically part of the plantduino)



Step 16: Plantduino: Step 2 Schematics

We have broken down the schematics so you can check them out piece by piece. If you can't make out the images, go to revoltlab.com and check out the larger images. These schematics were made with Fritzing! Fritzing is a great opensource schematic/ pcb/ and breadboard layout program.

The text in each schematic is reproduced below:

Picture 1 (Analog sensor) Connect pins 23, 24, and 25 to screw terminals as shown. These pins are the analog pins used for the sensors (moisture, temperature, and light). R1 is 10k ohms. Do not connect all the sensors to the same screw terminal. Three terminals each with two sockets are needed to connect all three sensors.

Picture 2 (Crystal/ Timing) The crystal and capacitors connect to pins 9 and 10 as shown. These will help the microcontroller keep proper time. C1 and C2 are 22pF. The crystal is 16Mhz.

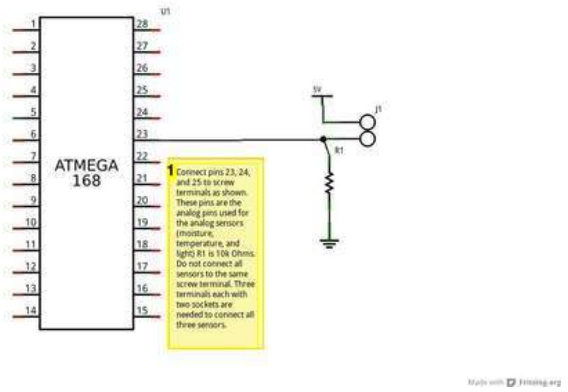
Picture 3 (LCD pins) The power and ground on the left of the LCD in this diagram supply power to the back light. DB7, DB6, DB5, and DB4 communicate with the microcontroller to display text. Vo is the contrast pin. You will have to experiment with different resistors to see which gives your LCD the best contrast. You can also use a 10k potentiometer if you wish. The LCD will be connected to arduino pins 7, 8, 9, 10, 11, and 12 which are shown here on the atmega pins 13 through 18.

Picture 4 (Power Regulation) The power plug is supplying 9 volts to the 7805 power regulator. 9 volts goes "IN". The ground of the 9 volt power plug goes to the middle "GND" pin of the regulator. "OUT" supplies 5 volts to the microcontroller. C1 and C2 both have their negative leads connected to ground. The power plug and microcontroller share the same ground. C1 has its positive lead connected to 9 volts. C2 has its positive lead connected to 5 volts. Both C1 and C2 are 10uF.

Don't forget to connect the power and ground on the other side of the chip!

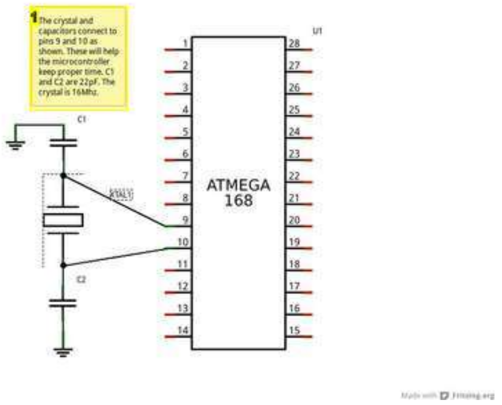
Picture 5 (Programming pins) The arrows above are male headers used for reprogramming the board. 8 is ground. 7 is 5 volts. 3 is TX. 2 is RX. 1 is reset. R1 is 10k ohms. C1 is 100nF.

Picture 6 (Relay pins) Connect pins 5, 6, 11, and 12 to their own screw terminals as shown with pin 5. These correspond with arduino pins 3, 4, 5, and 6 and will be used for controlling the relays.



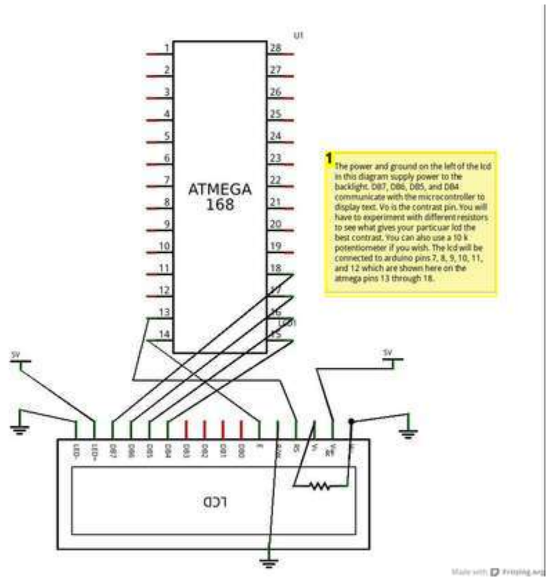
U1
ATMEGA 168
5V
J1
R1
1 Connect pins 23, 24, and 25 to screw terminals as shown. These pins are the analog pins used for the analog sensors (moisture, temperature, and light) R1 is 10k Ohms. Do not connect all sensors to the same screw terminal. Three terminals each with two sockets are needed to connect all three sensors.
Made with Fritzing.org

Image Notes
1. Connect pins 23, 24, and 25 to screw terminals as shown. These pins are the analog pins used for the sensors (moisture, temperature, and light). R1 is 10k ohms. Do not connect all the sensors to the same screw terminal. Three terminals each with two sockets are needed to connect all three sensors.



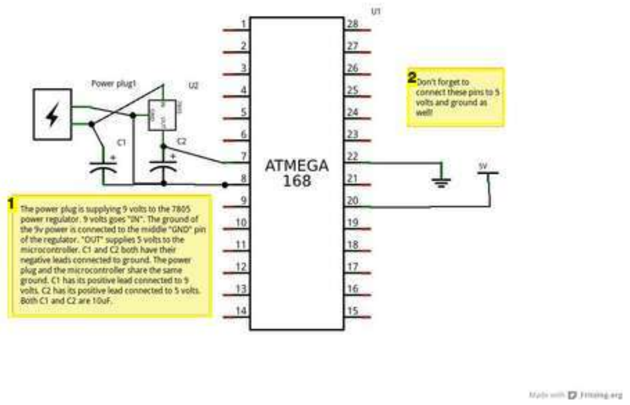
1 The crystal and capacitors connect to pins 9 and 10 as shown. These will help the microcontroller keep proper time. C1 and C2 are 22pF. The crystal is 16Mhz.
U1
C1
C2
XTAL1
ATMEGA 168
Made with Fritzing.org

Image Notes
1. The crystal and capacitors connect to pins 9 and 10 as shown. These will help the microcontroller keep proper time. C1 and C2 are 22pF. The crystal is 16Mhz.



U1
ATMEGA 168
1 The power and ground on the left of the lcd in this diagram supply power to the backlight. DB7, DB6, DB5, and DB4 communicate with the microcontroller to display text. Vo is the contrast pin. You will have to experiment with different resistors to see what gives your particuar lcd the best contrast. You can also use a 10 k potentiometer if you wish. The lcd will be connected to arduino pins 7, 8, 9, 10, 11, and 12 which are shown here on the atmega pins 13 through 18.
5V
LCD
R1
Made with Fritzing.org

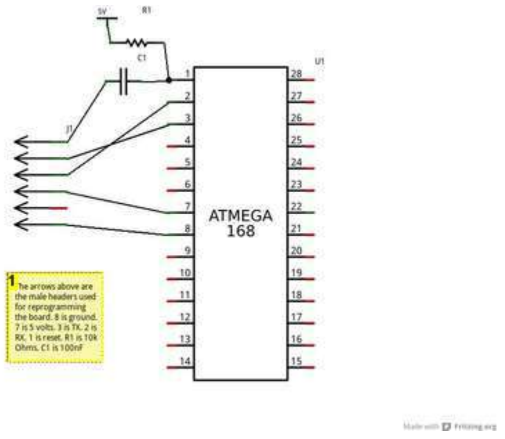
Image Notes
1. The power and ground on the left of the LCD in this diagram supply power to the back light. DB7, DB6, DB5, and DB4 communicate with the microcontroller to display text. Vo is the contrast pin. You will have to experiment with different



U1
Power plug1
U2
C1
C2
ATMEGA 168
2 Don't forget to connect these pins to 5 volts and ground as well!
5V
1 The power plug is supplying 9 volts to the 7805 power regulator. 9 volts goes "IN". The ground of the 9v power is connected to the middle "GND" pin of the regulator. "OUT" supplies 5 volts to the microcontroller. C1 and C2 both have their negative leads connected to ground. The power plug and the microcontroller share the same ground. C1 has its positive lead connected to 9 volts. C2 has its positive lead connected to 5 volts. Both C1 and C2 are 10uF.
Made with Fritzing.org

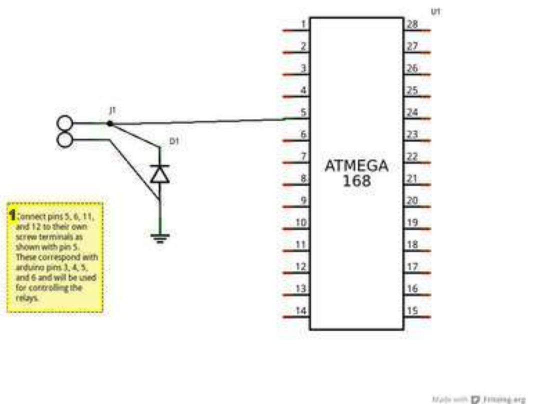
Image Notes
1. The power plug is supplying 9 volts to the 7805 power regulator. 9 volts goes "IN". The ground of the 9 volt power plug goes to the middle "GND" pin of the regulator. "OUT" supplies 5 volts to the microcontroller. C1 and C2 both have their negative leads connected to ground. The power plug and microcontroller share the same ground. C1 has its positive lead connected to 9 volts. C2 has its positive lead connected to 5 volts. Both C1 and C2 are 10uF.
2. Don't forget to connect the power and ground on the other side of the chip!

resistors to see which gives your LCD the best contrast. You can also use a 10k potentiometer if you wish. The LCD will be connected to arduino pins 7, 8, 9, 10, 11, and 12 which are shown here on the atmega pins 13 through 18.



ATMEGA 168
The arrows above are the male headers used for reprogramming the board. 8 is ground. 7 is 5 volts. 3 is TX. 2 is RX. 1 is reset. R1 is 10k Ohms. C1 is 100nF

Image Notes
1. The arrows above are male headers used for reprogramming the board. 8 is ground. 7 is 5 volts. 3 is TX. 2 is RX. 1 is reset. R1 is 10k ohms. C1 is 100nF.



ATMEGA 168
Connect pins 5, 6, 11, and 12 to their own screw terminals as shown with pin 5. These correspond with arduino pins 3, 4, 5, and 6 and will be used for controlling the relays.

Image Notes
1. Connect pins 5, 6, 11, and 12 to their own screw terminals as shown with pin 5. These correspond with arduino pins 3, 4, 5, and 6 and will be used for controlling the relays.

Step 17: Plantduino: Step 3 Assembly Tips and Tricks

For a full tutorial on making a generic arduino clone, check out the perfduino . The plantduino is an customized version of the perfduino.

Tip **#1** : Breadboard before you solder! Make sure your parts can work before you melt them to metal.

Tip **#2** : Go through the pictures and read the tags.

Tip **#3** : I chose a large perfboard for this project because of all the external connections. Make sure you leave room in your layout for the LCD.

Tip **#4** : Analog inputs are on the right. Digital outputs are on the left. I did this because the analog pins are all on the right of the atmega chip.

Tip **#5** : Start with the power supply. This will determine where you place other components.

Tip **#6** : While soldering the male headers to the LCD ribbon cable, keep them set in the female headers. This will prevent them twisting around in the heated plastic.

Tip **#7** : While every component is important, the 100nF capacitor and 10k resistor coming from the reset pin (pin 1) are absolutely vital! Your plantduino will not program without them!

Tip #8 : Connect the LCD wires on the back side of the board! It will save a lot of space and make it possible to examine your board for more than five seconds without losing normal human cognitive ability.

Tip #9 : This power jack will save you space. I got the large one for the sake of shipping costs.

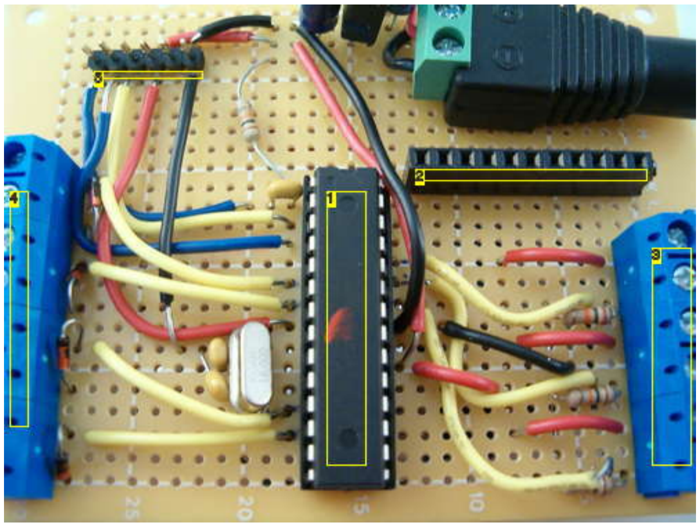


Image Notes
1. Atmega
2. LCD hookup pins
3. analog sensors
4. relay outputs
5. programming cables

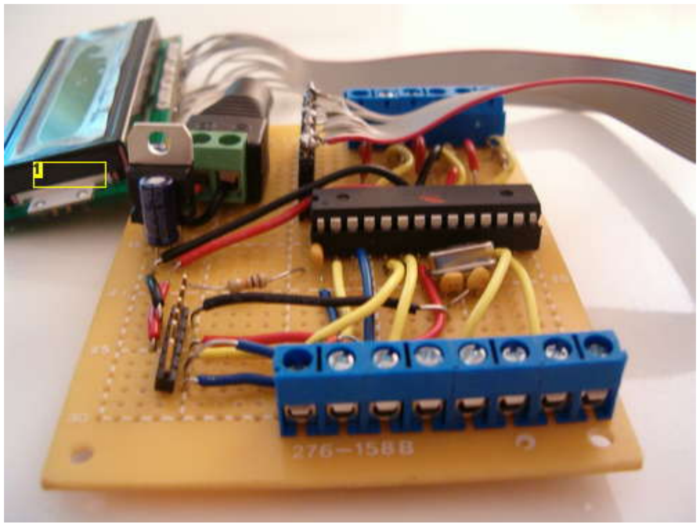
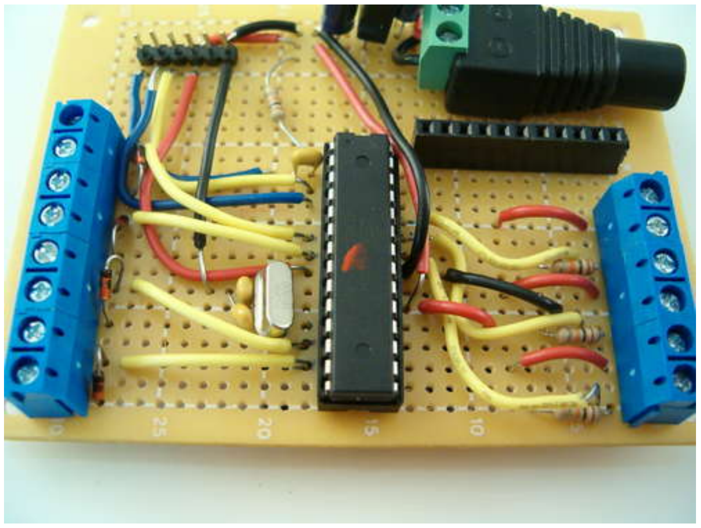


Image Notes
1. LCD



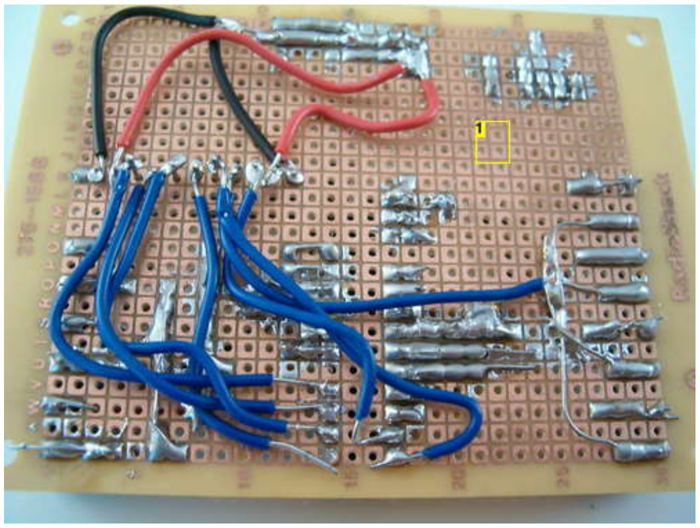
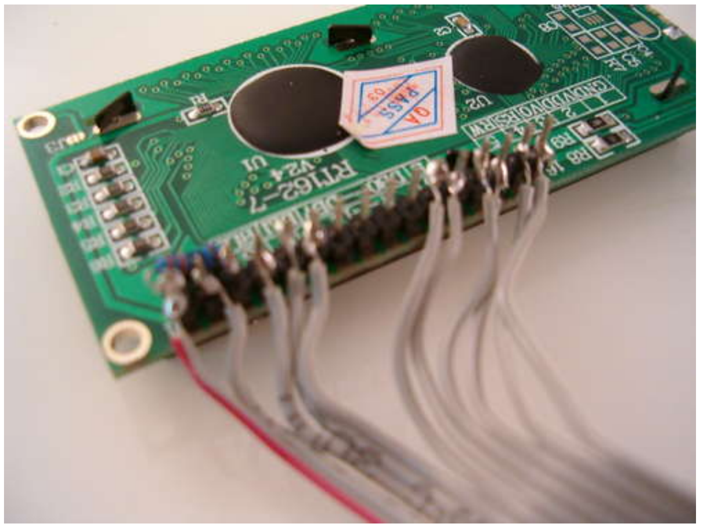


Image Notes
1. LCD wires are connected on the back of the board



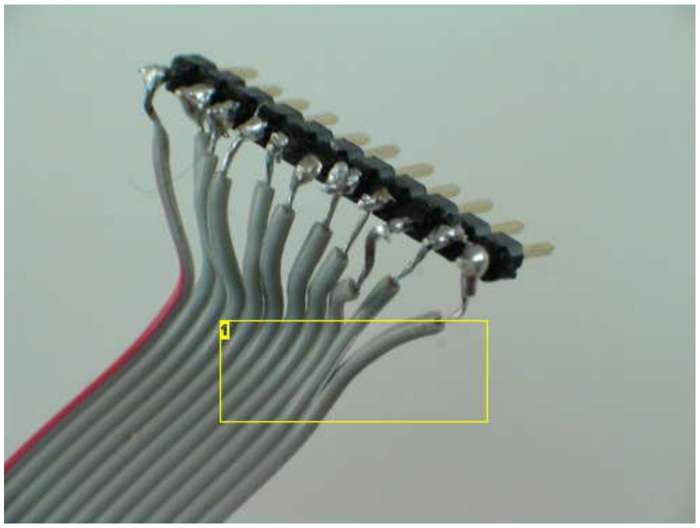


Image Notes
1. 11 wires come from the LCD

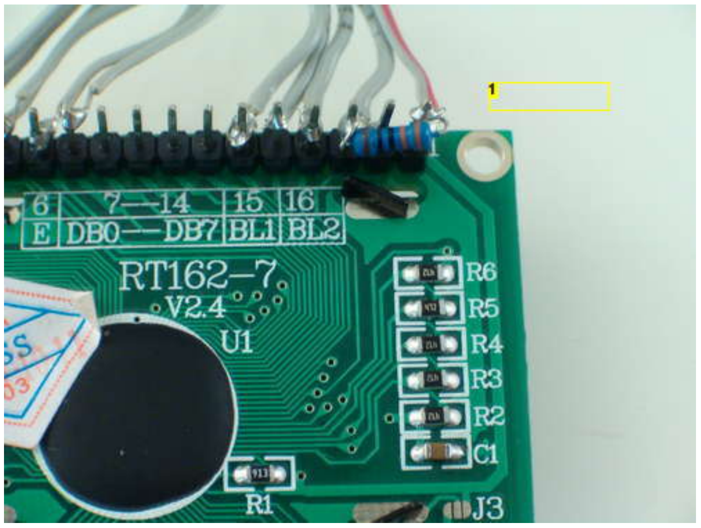


Image Notes
1. The resistor here connects to pins 1 and 3 of the LCD and controls the contrast. You can use a 10 k potentiometer instead if you don't want to bother testing resistors to find what value works with your screen

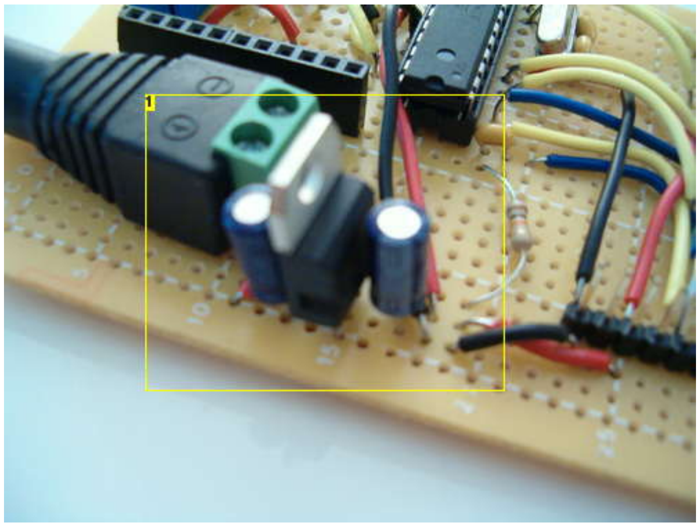


Image Notes
1. Power regulation.

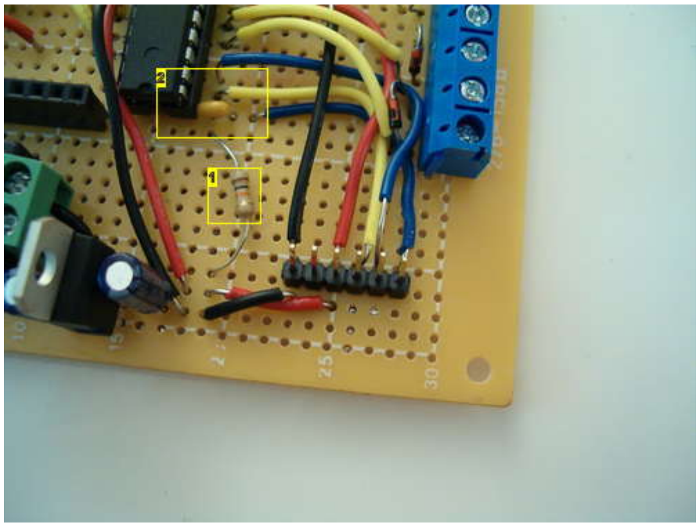


Image Notes
1. 10 k resistor goes from reset pin to 5 volts
2. DO not forget this 100nF capacitor! you need it to allow programming communication.

Step 18: Birdhouse: Creation and Installation

We bought a birdhouse from Michael's Craft Store to put all of the electronics in. You can use whatever enclosure you feel works for your home or environment. Here is what we did.
Materials:
1 wooden house cd holder- unpainted- Micheal's Craft Store
Oil paint
Paintbrushes
Drill
Screwdriver
Nails (any size)
Screw (any size)
Hacksaw
Plywood Sheet (16 inches by 14 inches)- from Micheal's craft store
Hinges- bought from home depot
Magnetic door clasps- bought from home depot
Clear Plastic Portfolio envelope

Steps:
1) Paint the Birdhouse: We used oil paints because they are harder for the rain to wash off. Be warned, they take a very long time to dry.
2) Cut the doors: Using a hacksaw I cut the doors out of a piece of plywood.
3) Attach the hinges- I bought a pair of really small hinges at home depot. The screwed on side of the birdhouse and then on the side of the doors.
4) Glue on the Magnetic Door Clasps- The magnetic clasps keep the doors shut. However, the magnets are very strong. You can increase and decrease the force of the magnet by limiting the exposure the magnet and the clasp have. We just hot glued these two pieces on.
5) Weatherproof the birdhouse- We used the plastic from a clear plastic portfolio envelope to coat the windows and other exposed areas of the birdhouse.
6) Cut holes- We used a drill to drill two holes into the bottom of the birdhouse and two in the second story floor. The holes on the bottom floor of the birdhouse are for the wires from the nails and the thermistor in the greenhouse that are connected to the Garduino microcontroller. The wires connecting the relay box to the micro-controller go through there too.
7) Nail the birdhouse to a sturdy spot- We nailed the birdhouse to the support beam on my deck. We nailed the bottom and the back to make sure it won't blow away.
8) Put the relay box into the birdhouse. We put it into the bottom floor. We strung the power cord through one of the windows of the house.
9) Put the microcontroller into the box. We put it in the top floor.

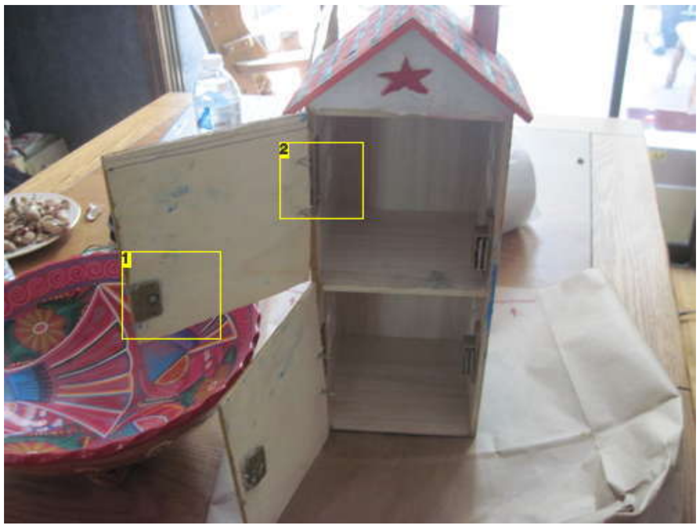


Image Notes
1. Magnetic door clasps
2. Hinge

Image Notes
1. Hinge

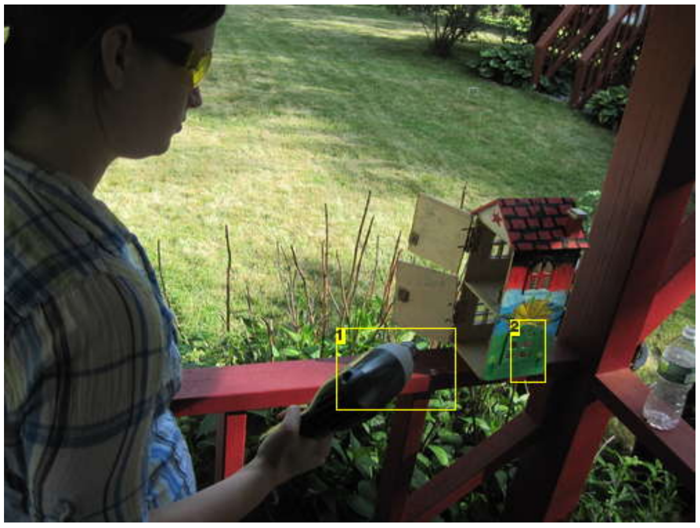
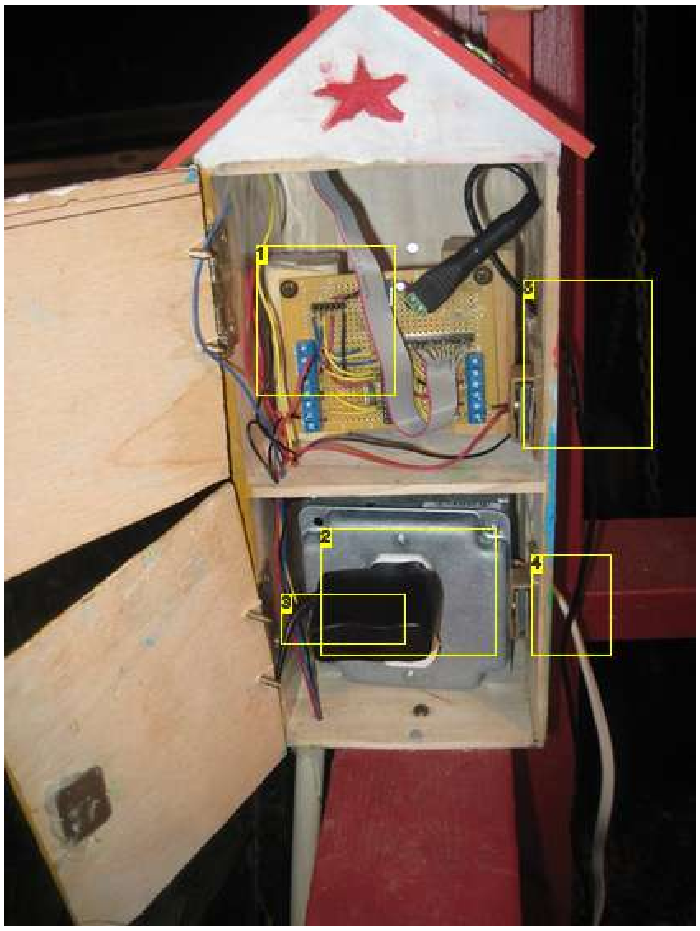


Image Notes
1. Securely Attach it to house
2. These windows were cut out so that the plug from the relay bow could reach the extension cord

Step 19: Creating the Birdhouse Motherboard



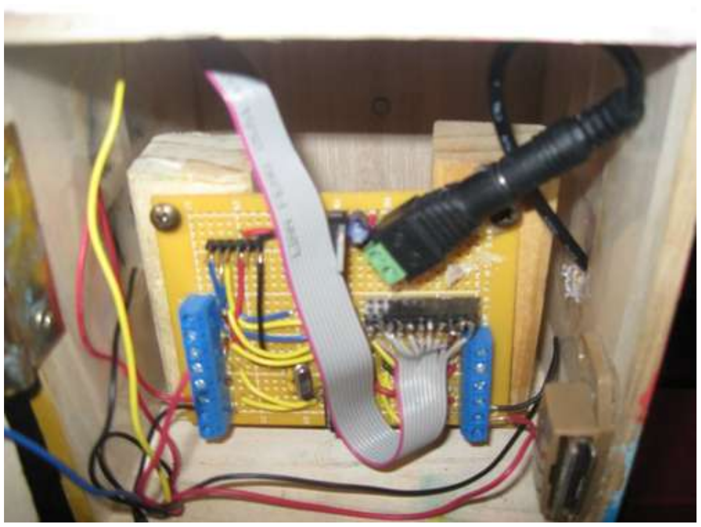


Image Notes
1. We nailed the board to pieces of wood that are offset from the back for easy access to the screw terminals
2. Relay box goes into the bottom shelf of the birdhouse
3. Wallwart that is connected to the solenoid valve (make sure it is plugged into the right outlet!)
4. cord from relay box going to outdoor extension cord
5. wall-wart from the arduino going to the outdoor extension cord

Image Notes
1. LCD screen displaying temperature and watering status of the greenhouse, including Celsius and F readings.

Image Notes
1. Transparent plastic protecting the screen from water damage
2. snake cords through the windows

Image Notes
1. make sure to snake the irrigation tubing in the garden so that all the plants are watered equally

Image Notes
1. oil paint not running even in the rain!

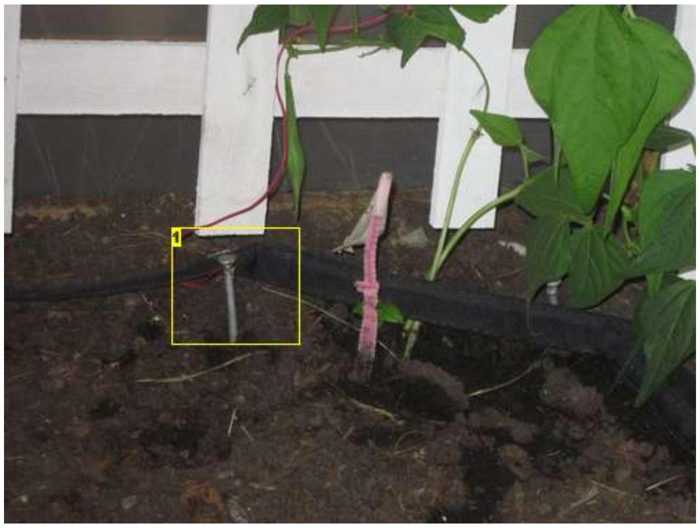


Image Notes
1. place the moisture sensors in a place that is representative for the entire garden

Step 20: Video

http://youtu.be/hdojUHjg35g

Step 21: Final Thoughts/ Additional Reading

Adding Light and Temperature regulation:

This link shows how to include photoresistors and 10K thermistors as analog inputs. Using these tools you can regulate the light and temperature of the greenhouse.

For more information:

On watering systems:
Garduino project in Make Magazine

On plant biology/ botany
My plant biology textbook website

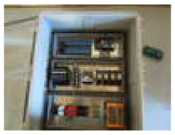
On electronics
Sparkfun
Make Magazine

On Arduino:
Arduino playground and forums
Adafruit

Related Instructables

Autonomous Greenhouse Factory by pablopeillard

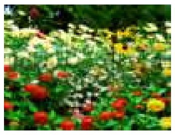
Solar-powered wireless tweeting birdhouse (video) by smfrayne

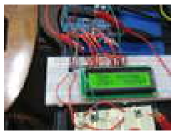

Greenhouse Climate Control System Preview (Photos) by LancePenney

The Arduino Weather Station / Thermostat by sspence

Arduino Asteroid Game by SanticN4N

Telemetry with solar cell by bthjehiel


Garduino- Automated Gardening System by dls02010


Arduino Based Temperature Monitor (video) by kunal_djscoe

The EyeWriter 2.0

by **thesystemis** on December 1, 2010

Intro: The EyeWriter 2.0

The EyeWriter is a low-cost eye-tracking apparatus + custom software that allows graffiti writers and artists with paralysis resulting from Amyotrophic Lateral Sclerosis to draw using only their eyes.

The original design, as shown here, featured a pair of glasses as the basis for the eyewriter design:

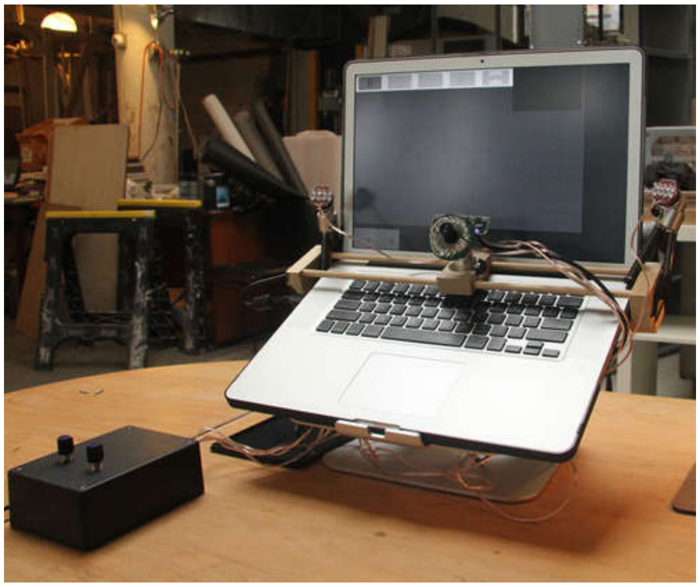
Since that first video, we've been hacking on and developing the project, and we have a new design, which we've called "eyewriter 2.0" which improves the accuracy of the device, and allow for people who's heads are moving slightly to also use an eye tracker. The original eyewriter, designed for a paralyzed Graffiti artist TEMPT1, is designed to be worn on a completely motionless head. The 2.0 design, which uses a camera and LED system mounted away from the head, can be used by people whose heads are moving slightly, such as MS patients, and people who wear glasses, etc.

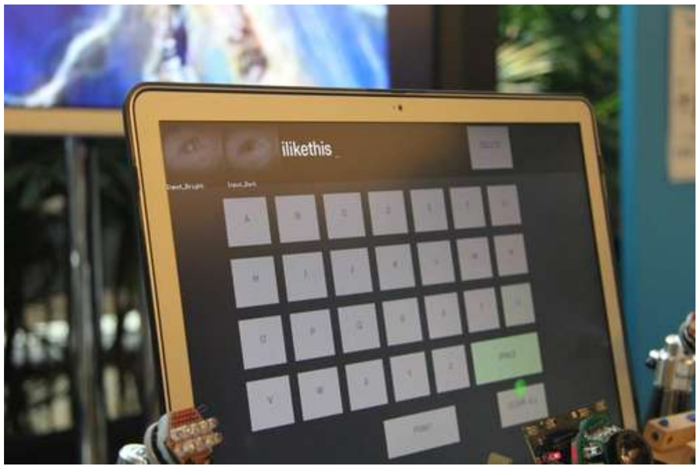
This eyewriter system is cheap, and completely open source. At the moment, it costs about 200$ in parts. Traditional commercial eye trackers costs between $9000-$20,000, so this is a magnitude of order cheaper, and is designed to help anyone who wants or needs an eyetracker.

This fall, we've been showing off and demoing the 2.0 device -- check out the eyewriter 2.0 in action -- we even hooked it up to a robotic arm, to draw the artwork people make with their eyes:

http://www.switched.com/2010/12/13/eyewriter-teams-up-with-robotagger-to-print-kids-ocular-artwork/print/

(The 2.0 device was designed with help and input from Takayuki Ito, Kyle McDonald, Golan Levin and students of the eyewriter collab at Parsons MFADT. Thanks also to the Studio for Creative Inquiry / CMU for hosting a session for development)





Step 1: Overview

The basic idea approach is that we'll be doing a few things. First, we'll be making LED illuminators for the sides of the screen and the center. Second, we'll be hacking the PS3 eye camera to get the vertical sync (when the frame of video is being taken) and to make it sensative to IR. Third, we'll be programming and building the arduino / cirucit to control the blinking. Finally, we'll setup the base for the system and go through the basics of the software.

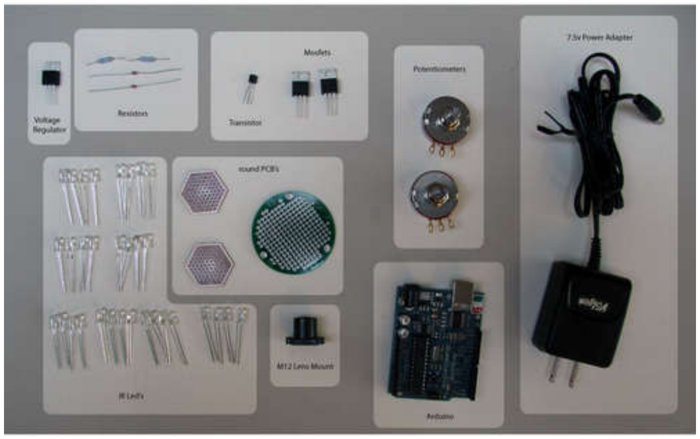
From a technical perspective, the 2.0 system works by strobing 3 IR illuminators every frame. On even frames, it uses the center illuminator (located around the camera lens) and on odd frames it uses the 2 side illuminators. On even frames, the pupil appears bright, since the IR light is actually bouncing off the back of your eye, like red eye effect. On odd frames, your pupil appears dark. The difference between the two allows us to isolate and track the pupil in realtime. Additionally, the glints (reflections of the IR illuminators) of the dark frame are tracked, and these, plus the info on the pupil, is calibrated to screen position using a least squares fitting process for an equation that provides a mapping of glint/pupil position to screen position.

Step 2: Parts list

to begin with, we will need a fair number of parts to make this 2.0 device. Please see this image to get a sense of what we will be working with, as well as this detailed parts list pdf

see parts image

download detailed parts list:



Step 3: Software - openFrameworks & EyeWriter

The Eyewriter 2.0 requires a few pieces of software for building and running. In this step we will explain how to download and install an IDE, openFrameworks, and the eyeWriter software.

A. Integrated Development Environment (IDE)

- An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.
- Download and install an Integrated Development Environment (IDE) to run openFrameworks if necessary. http://www.openframeworks.cc/setup

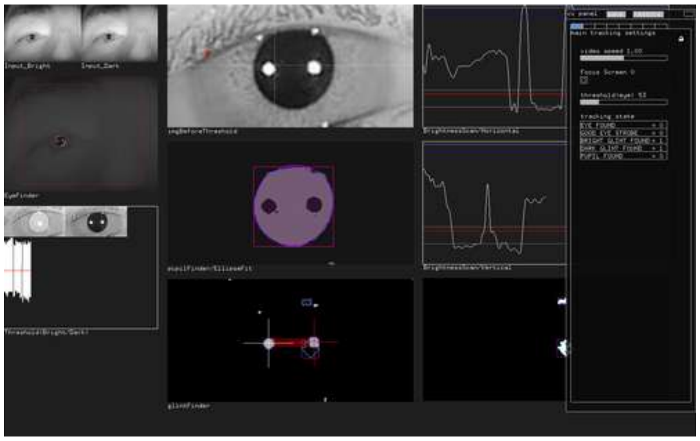
B. openFrameworks

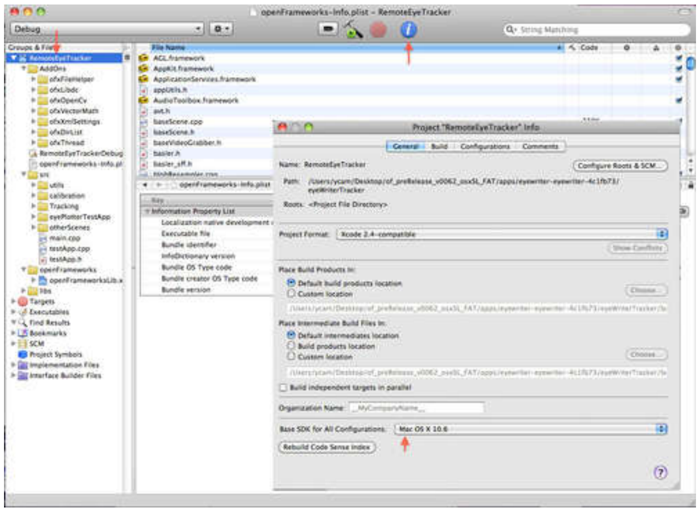
- Openframeworks is a c++ library designed to assist the creative process by providing a simple and intuitive framework for experimentation.
- Download and install openFrameworks if necessary.
  http://www.openframeworks.cc/download

C. EyeWriter GitHub

- GitHub is a web-based hosting service for projects that use the Git revision control system. It is a platform that allows people to exchange and share code.
- Visit the EyeWriter source page on GitHub.
  http://github.com/eyewriter/eyewriter/tree/remoteEyetracker

- Click Download Source on the top right menu.
- Choose ZIP format.
- After download is complete, unzip the file and place the "eyewriter-xxxxxxx" folder into openFrameworks "apps" folder.
- Open the "apps/eyewriter-xxxxxxx/eyeWriterTracker/RemoteEyeTracker.xcodeproj" file to test that all installations are working correctly. The source code should load in your IDE software.
- please be sure you're compiling for your current Operating System (the eyewriter software was originally compiled for OSX 10.5 so you might need to change compiling from 'base SDK' to 'OSX 10.6')
- Build and Run the source code. The Tracking screen should load in video demo mode.





Project "RemoteEyeTracker" Info

Name: RemoteEyeTracker

Path: /Users/ycam/Desktop/of_preRelease_v0062_osxSL_FAT/apps/eyewriter-eyewriter-4c1fb73/eyeWriterTracker

Roots: <Project File Directory>

Project Format: Xcode 2.4-compatible

Place Build Products In: [✓ Default build products location] Custom location

Place Intermediate Build Files In: [✓ Default intermediates location] Build products location Custom location

Build independent targets in parallel

Organization Name: __MyCompanyName__

Base SDK for All Configurations: Mac OS X 10.6

Step 4: Software - Camera & Arduino

We will also need to install two additional pieces of hardware. Macam will allow our PS3 eye camera to talk to our computer and the Arduino software will permit our physical hardware to communicate with our software.

Installing PS Eye drivers

For Mac:

- Macam is a driver for USB webcams on Mac OS X. It allows hundreds of USB webcams to be used by many Mac OS X video-aware applications. Since we are using a PS3 camera, this software will allow our computers to recognize the hardware.
- Download the Macam driver from SourceForge. http://sourceforge.net/projects/webcam-osx/files/cvs-build/2009-09-25/macam-cvs-build-2009-09-25.zip/download
- After download is complete, unzip the file and place the macam.component file into your hard drives /Library/Quicktime/ folder.

For PC:

- download the CL-Eye-Driver:http://codelaboratories.com/downloads/

Arduino

- Arduino is a tool for the design and development of embedded computer systems, consisting of a simple open hardware design for a single-board microcontroller, with embedded I/O support and a standard programming language
- Download and install the Arduino software. http://arduino.cc/en/Main/Software
- Follow the Getting Started tips if you're unfamiliar with the Arduino environment. http://arduino.cc/en/Guide/HomePage

Step 5: Load Arduino sketch

In this step you will have to load the Arduino sketch for the PS eye camera to work.

A. Arduino Sketch (Only for PS Eye)

- Load the Arduino EyeWriter sketch "apps/eyewriter-xxxxxxx/eyeWriterTracker/StrobeEye/StrobeEye.pde" in the Arduino IDE software. This needs to be done in order that the eyewritter software can recognize the hardware.
- With your Arduino board connected, upload the sketch to your board. Follow the Getting Started tips if you're unfamiliar with the Arduino environment. http://arduino.cc/en/Guide/HomePage

Step 6: Hardware: Power Adapter

Power Adapter

In this step you will cut the wire of a power adapter to power your breadboard

- Clip off the connector jack of your 7.5 Volt Power Adapter. see image here
- Use a Voltmeter to determine the positive and negative wires in the adapters exposed cord.
- Using a short strip of red and black wire, solder the red wire to the adapters positive wire, and solder the black wire to the adapters negative wire.
- Tape the exposed wires separately to keep positive and negative apart, then tape both together to ensure no wire is exposed.

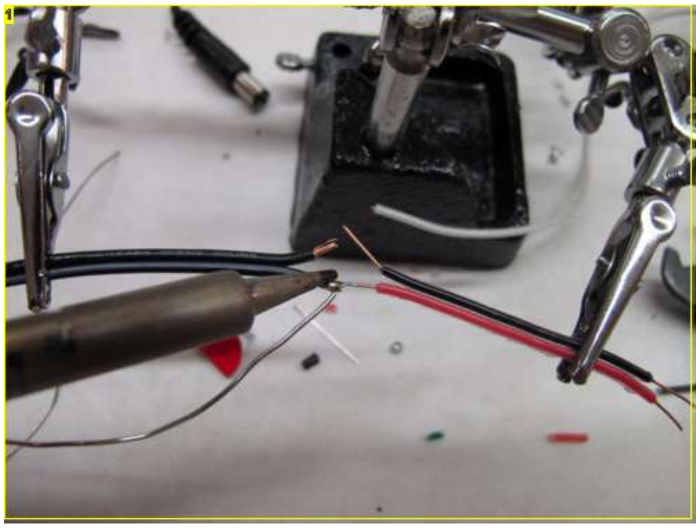


Image Notes

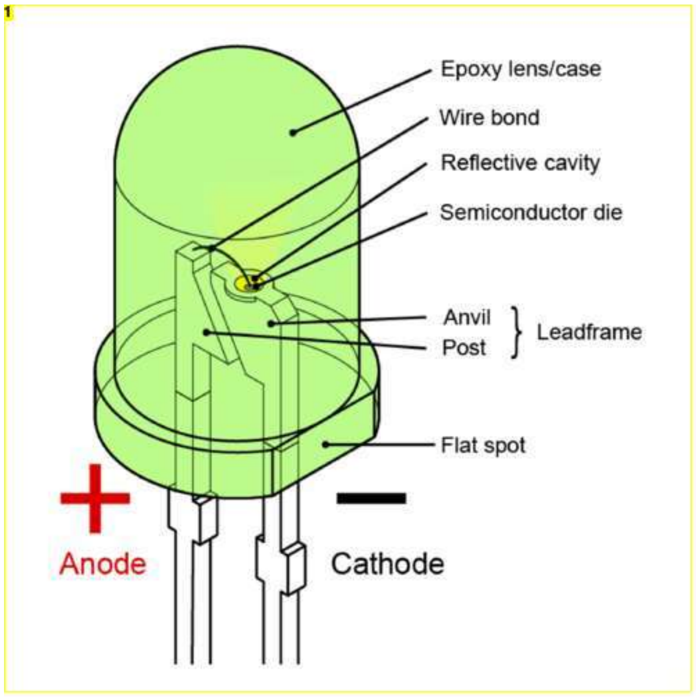
1. Using a short strip of red and black wire, solder the red wire to the adapter’s positive wire, and solder the black wire to the adapter’s negative wire.

Step 7: Hardware: Infrared LED's

IR LED's

- Gather 8 Infrared (IR) Light-Emitting Diodes (LED) and a small round Printed Circuit Board (PCB).
- To build LED arrays on the PCBs youll need to know the positive and negative ends of each LED. Generally speaking the longer leg of the LED is the anode (positive), and the shorter leg is the cathode (negative). see image here

  On most LEDs, there will also be a flat spot on the cathodes side of the lens.
  From overhead, take note of which direction the wire bond points relative to positive and negative.
- Setup a circuit of 4 LEDs in series, in parallel with another set of 4 LEDs in series.see image here Clip the legs of the LEDs and solder them together. see image here
- After soldering the LED legs together to form the circuit, solder about 2 feet (60 centimeters) of the red & green intercom wire to the LED circuits positive & negative ends.see image here
- To test the LED PCB panel, build the circuit below. Look carefully to see if your IR LEDs are glowing a faint red. see image here
- After confirming your IR LEDs are working, cover the back of the LED PCB panel with hot glue to keep all connections in place.
- Repeat steps 1 - 5 above to create another LED PCB panel.
- Using a larger round PCB, carefully drill press a hole into the center of the board. see image here
- On the outer rim of the PCB, build a circuit of 4 parallel sets of 4 LEDs in series. The placement of the LEDs should allow the PS Eye camera to fit through snugly, without the camera blocking the LEDs. see image here
- After soldering the LED legs together to form the circuit, solder wiring to connect all 4 positive ends together and all 4 negative ends together, putting all 4 LED sets in parallel. see image here
- Solder about 2 feet (60 centimeters) of the red & green intercom wire to the LED circuits positive & negative ends.
- To test the larger LED PCB panel, build the circuit below. Look carefully to see if your IR LEDs are glowing a faint red. see schematic here
- After confirming your IR LEDs are working, cover the back of the LED PCB panel with hot glue to keep all connections in place.



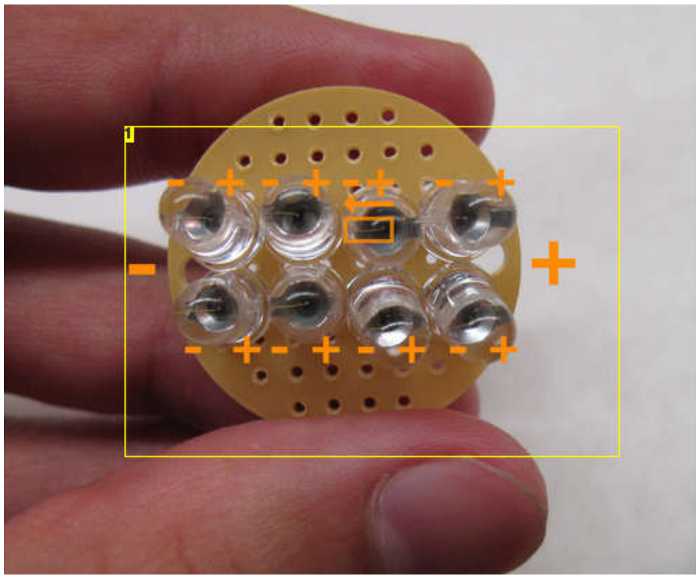


Image Notes
1. From overhead, take note of which direction the wire bond points relative to positive and negative. Setup a circuit of 4 LEDs in series, in parallel with another set of 4 LEDs in series.

Image Notes
1. To build LED arrays on the PCBs you'll need to know the positive and negative ends of each LED. Generally speaking the longer leg of the LED is the anode (positive), and the shorter leg is the cathode (negative).

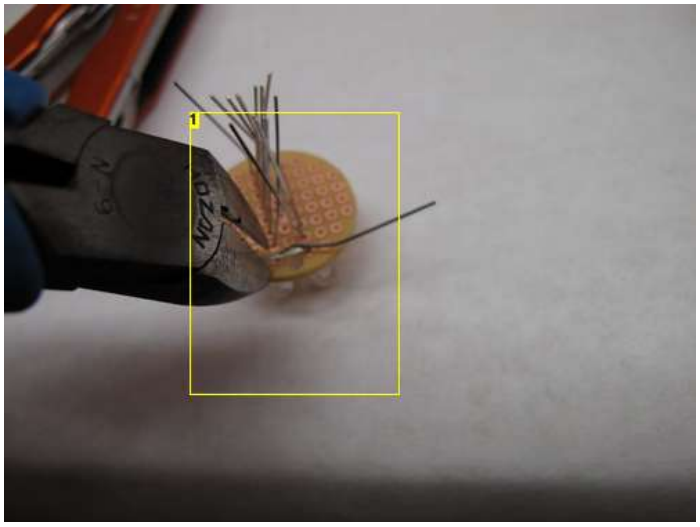


Image Notes
1. Clip the legs of the LEDs and solder them together.

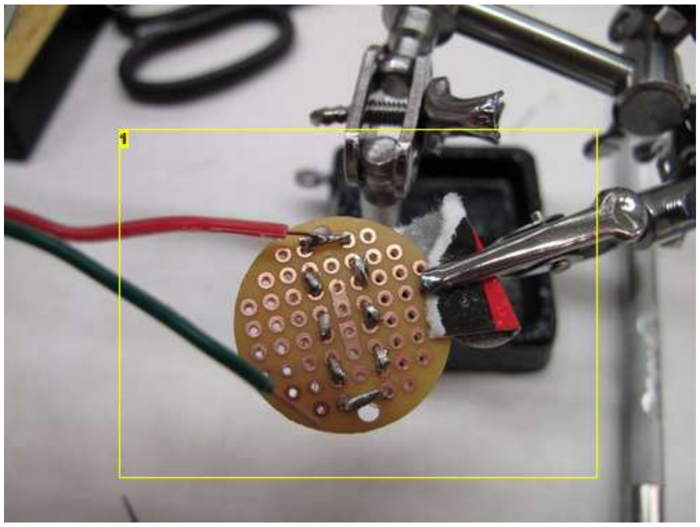


Image Notes
1. After soldering the LED legs together to form the circuit, solder about 2 feet (60 centimeters) of the red & green intercom wire to the LED circuit s positive & negative ends.

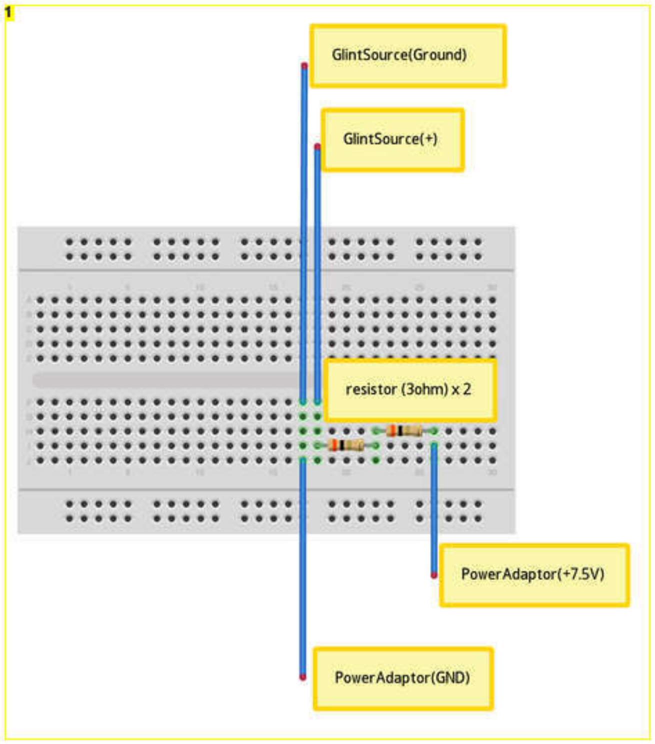


Image Notes
1. To test the LED PCB panel, build the circuit below. Look carefully to see if your IR LEDs are glowing a faint red.

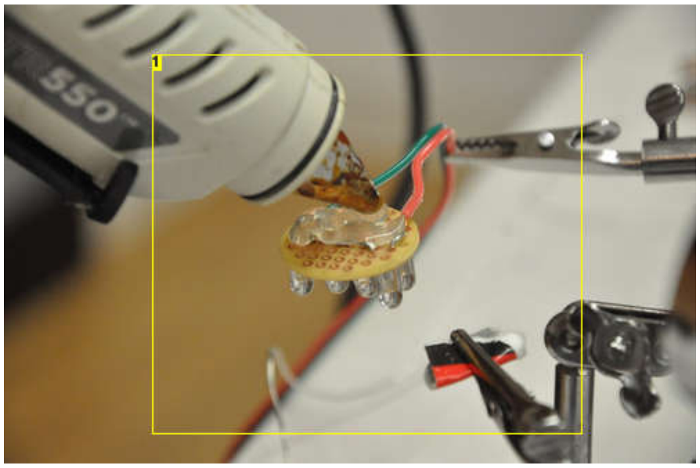


Image Notes
1. After confirming your IR LEDs are working, cover the back of the LED PCB panel with hot glue to keep all connections in place.

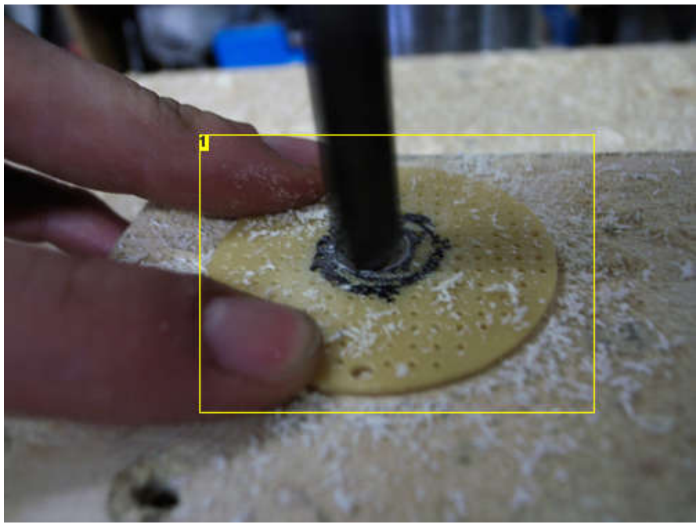


Image Notes
1. Using a larger round PCB, carefully drill press a hole into the center of the board.

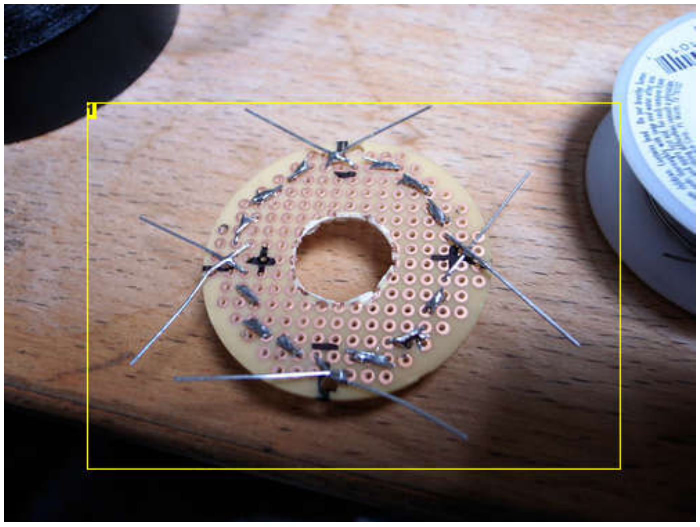


Image Notes
1. On the outer rim of the PCB, build a circuit of 4 parallel sets of 4 LEDs in series. The placement of the LEDs should allow the PS Eye camera to fit through snugly, without the camera blocking the LEDs.

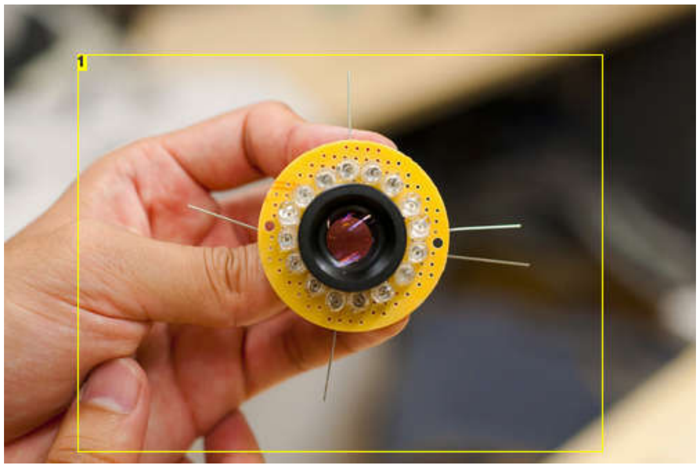


Image Notes
1. After soldering the LED legs together to form the circuit, solder wiring to connect all 4 positive ends together and all 4 negative ends together, putting all 4 LED sets in parallel.

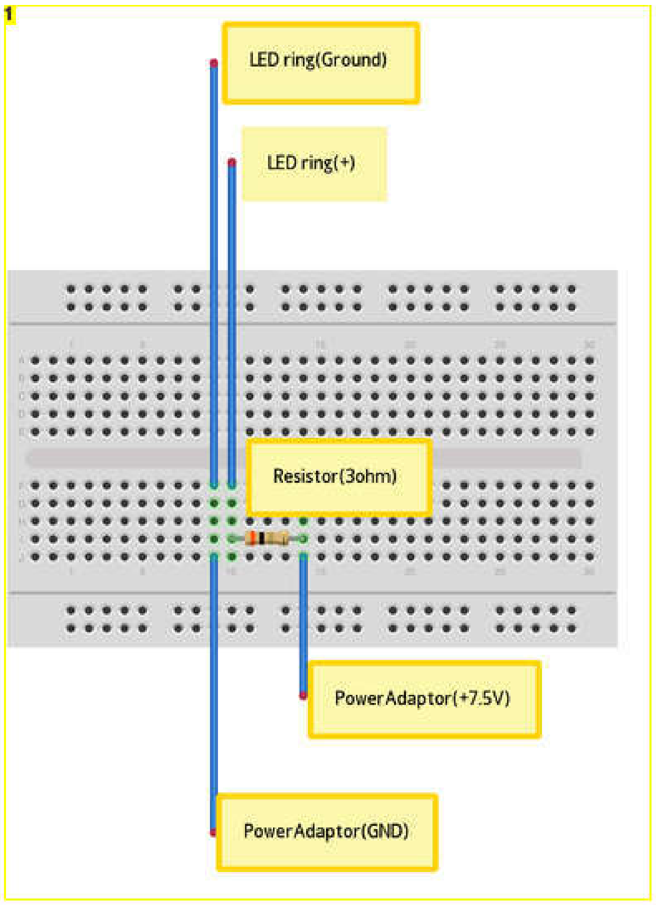


Image Notes
1. To test the larger LED PCB panel, build the circuit below. Look carefully to see if your IR LEDs are glowing a faint red.

Step 8: Hacking the PS Eye camera - preparing

In this step we will talk about how to take apart a PS Eye camera. This is necessary for you to be able to replace the lens on the camera, insert a infrared filter and wire the v-sync.

- Get a PlayStation (PS) Eye camera. Use at your own risk because the camera will undergo modifications voiding its warranty.
- Pry the four plastic screw caps off the back of the casing. see image here
- Unscrew the four screws underneath where the screw caps were. Keep these screws because you will need some later.
- With all four screws removed, pry off the back half of the casing. A flathead screwdriver and hammer, or a pair of pointed pliers should work. It requires significant force so be very careful not to damage anything inside or hurt yourself. see image here
- Pull the cord aside and unscrew the two bottom screws beside the plastic holder. Keep these screws also. see image here
- Remove the stand piece.
- Unscrew the five screws around the board (two screws on the side, three screws on top). Keep these screws also. see image here
- With all five screws removed, lift the board out of the front casing.
- There are four microphones across the top of the board. Using wire cutters, clip off the microphones because they won't be used. see image here
- Now the PS Eye board is prepared for wiring. The next steps will connect wiring to the Vertical Synchronization (V-Sync) and Ground joints on the PS Eye board.

Image Notes
1. Get a PlayStation (PS) Eye camera. Use at your own risk because the camera will undergo modifications voiding its warranty.

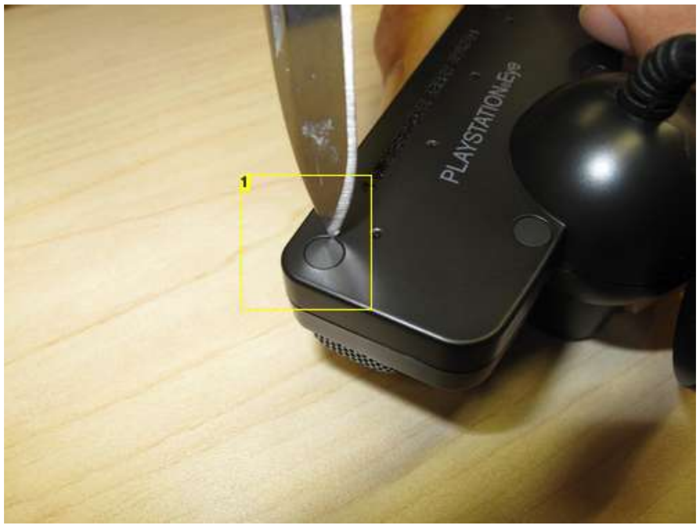


Image Notes
1. Pry the four plastic screw caps off the back of the casing.

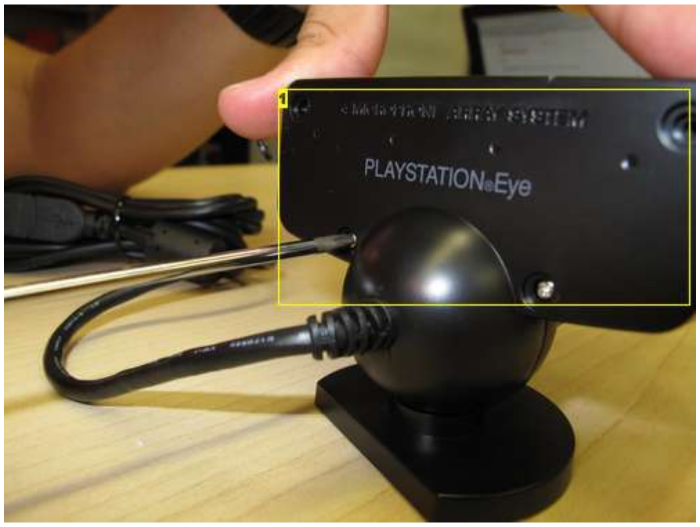


Image Notes
1. Unscrew the four screws underneath where the screw caps were. Keep these screws because you will need some later.

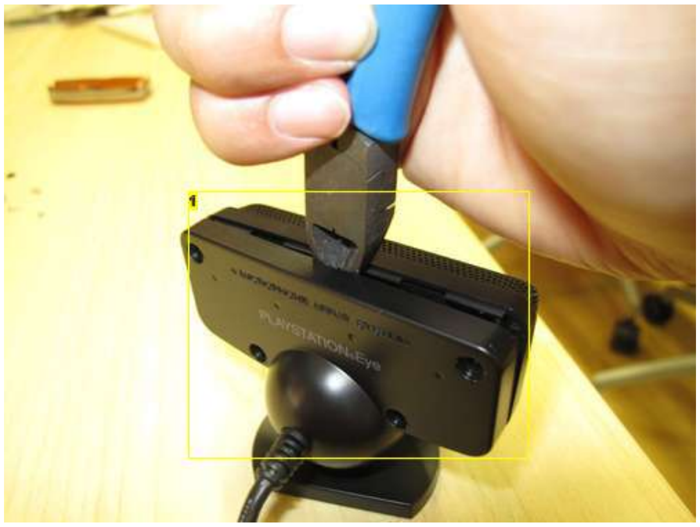


Image Notes
1. With all four screws removed, pry off the back half of the casing. A flathead screwdriver and hammer, or a pair of pointed pliers should work. It requires significant force so be very careful not to damage anything inside or hurt yourself.

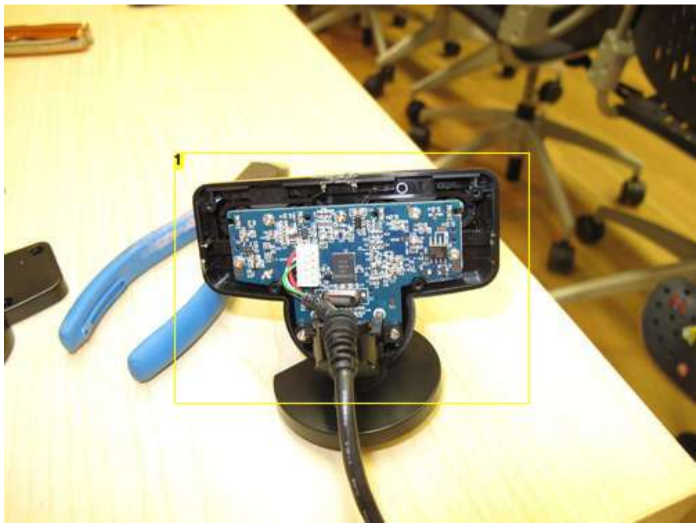


Image Notes
1. Pull the cord aside and unscrew the two bottom screws beside the plastic

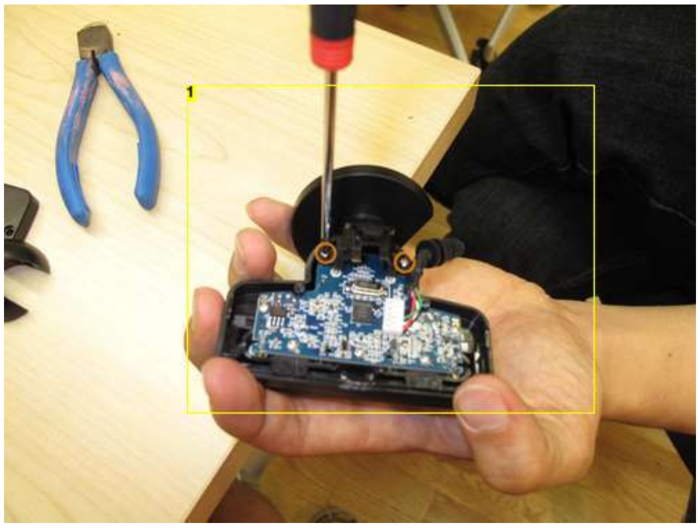


Image Notes
1. Pull the cord aside and unscrew the two bottom screws beside the plastic

holder. Keep these screws also.

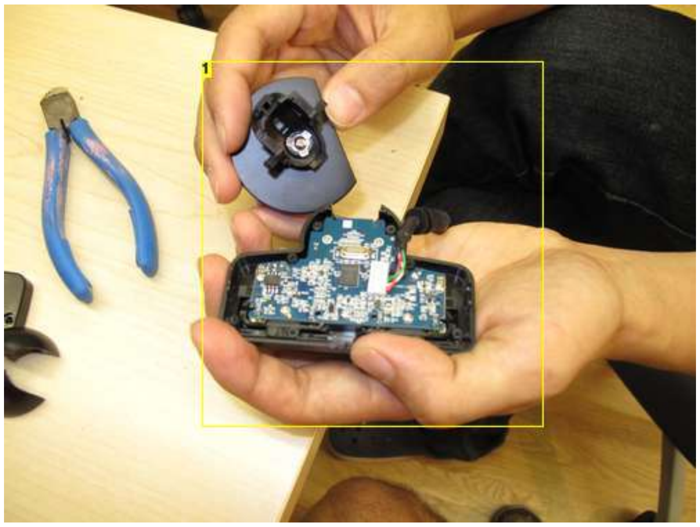


Image Notes
1. Remove the stand piece.

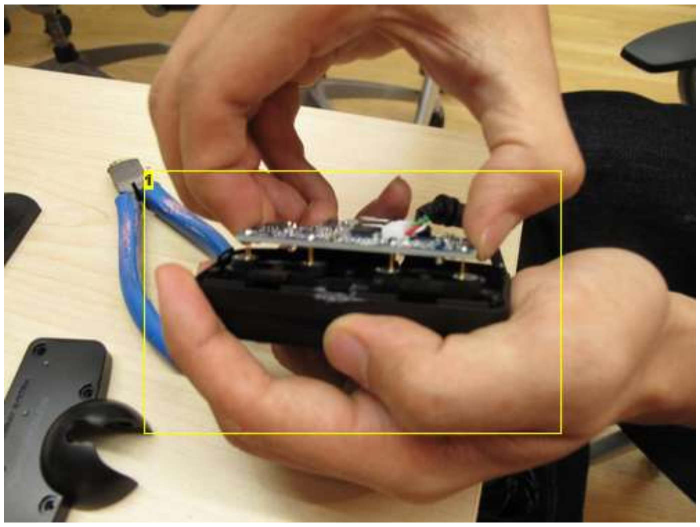
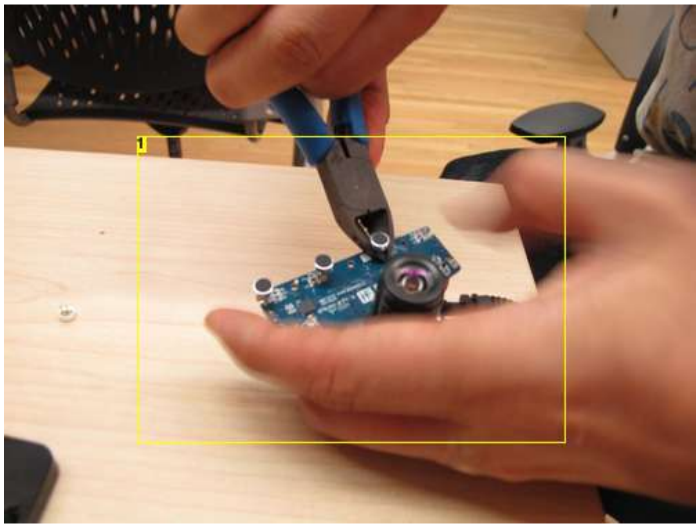


Image Notes
1. With all five screws removed, lift the board out of the front casing.



holder. Keep these screws also.

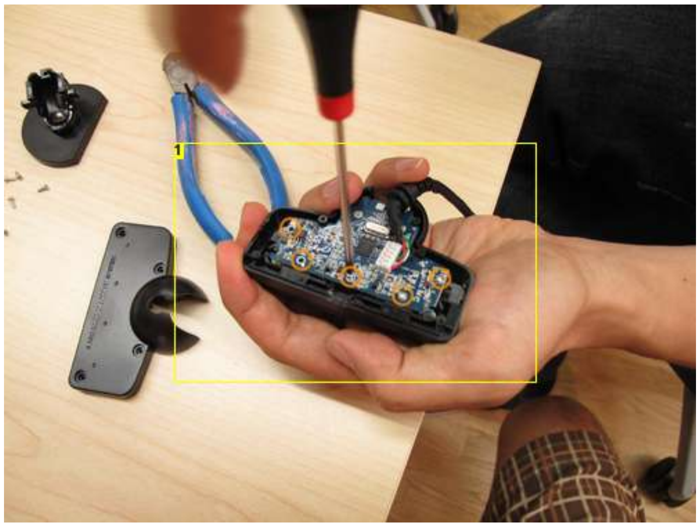


Image Notes
1. Unscrew the five screws around the board (two screws on the side, three screws on top). Keep these screws also.

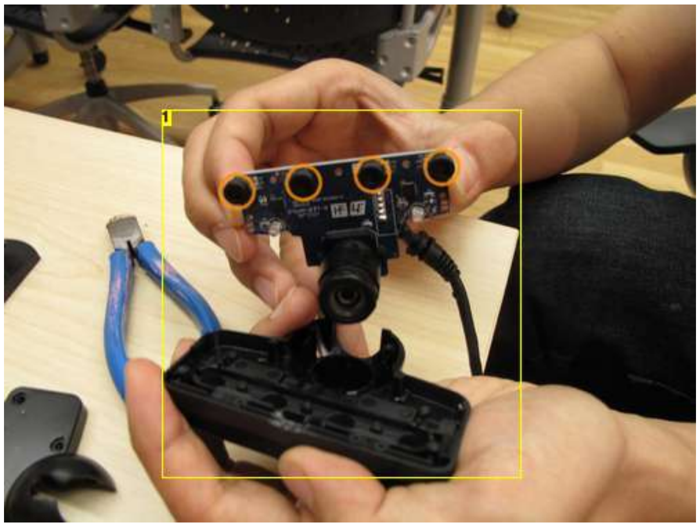


Image Notes
1. There are four microphones across the top of the board. Using wire cutters, clip off the microphones because they won't be used.

Image Notes
1. There are four microphones across the top of the board. Using wire cutters, clip off the microphones because they won't be used.

Step 9: Hacking the PS Eye camera - VSync

In this step we will go through getting the v-sync off the camera. The v-sync is an electrical signal that comes from the camera which communicates the camera's refresh rate. Getting the camera's v-sync is crucial for this application to work because it is the only way we can match the camera's refresh rate to our infrared LED's.

- Locate the Ground joint on your PS Eye board. Some PS Eye models have 5 joints near the lens mount (left image below), while some have 4 joints (right image below). If your model has 5 joints, the Ground joint is at the end closest to the lens mount. If your model has 4 joints, the Ground joint is also at the end closest to the lens mount, and twice as wide as the other joints. see image here
- Cut about 2 feet (60 centimeters) of your 4-color intercom wire, and split the red and green from the black and white.
- Split the red and green wire about 2 inches (5 centimeters) from one end, and strip off a small section of insulation at the end of the green wire. The green wire will be soldered to the PS Eyes Ground joint.
- Clip the PS Eye board and green wire to a stand, and prepare to solder the green wire tip to the Ground joint. Use a piece of thick paper or cardboard in between the clips teeth to prevent scrapes on the board. see image here
- Solder the green wire to the PS Eyes Ground joint.
- Locate the V-Sync via on the board. Its the via circled in the image below. see image here. **Attention:** for more recent models of the PSEye camera (identified by the golden rim around the board) the VSync hotspot can be found on the front of the PCB, directly above the R19 resistor. see image here. Very REcently a newer model was also introduced in the market (v9.2)see how to identify it in this image and how to find the vSync spot in this image
- Using a sharp knife, carefully pivot the knife tip on the via, and scrape off enough insulation coating to expose the metal contact below. see image here
- The red wire needs to connect to the exposed V-Sync via, but the wire is too thick to be soldered neatly to the small via, so a 30 gauge wire will be used in between. Strip the ends of a 2 piece of 30 gauge wire.
- Shorten the red wire, then solder one end of the 30 gauge wire to the end of the red wire. see image here
- Before soldering the 30 gauge wire to V-Sync, a test should be performed to ensure all connections are correct. Build the circuit below. When the 30 gauge wire contacts the V-Sync via, the LED on the breadboard should flicker rapidly. see schematic here
- Using thin 0.022 inch (0.56 millimeters) solder, carefully solder the 30 gauge wire to the exposed V-Sync via. To confirm, ensure the LED on the breadboard is flickering.

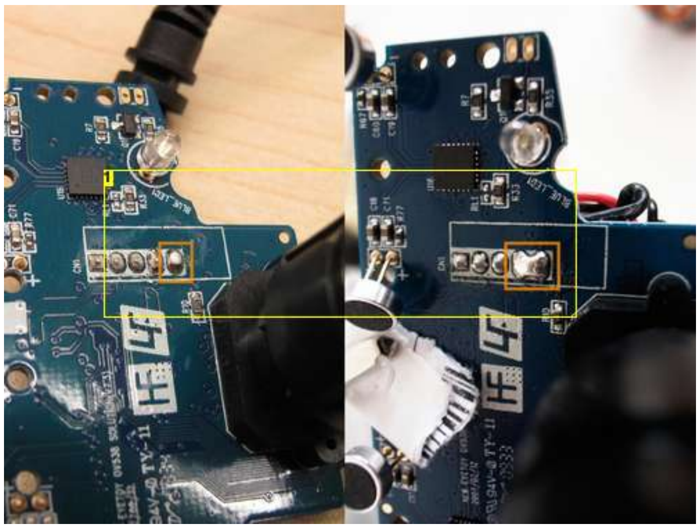


Image Notes
1. Locate the Ground joint on your PS Eye board. Some PS Eye models have 5 joints near the lens mount (left image below), while some have 4 joints (right image below). If your model has 5 joints, the Ground joint is at the end closest to the lens mount. If your model has 4 joints, the Ground joint is also at the end closest to the lens mount, and twice as wide as the other joints.

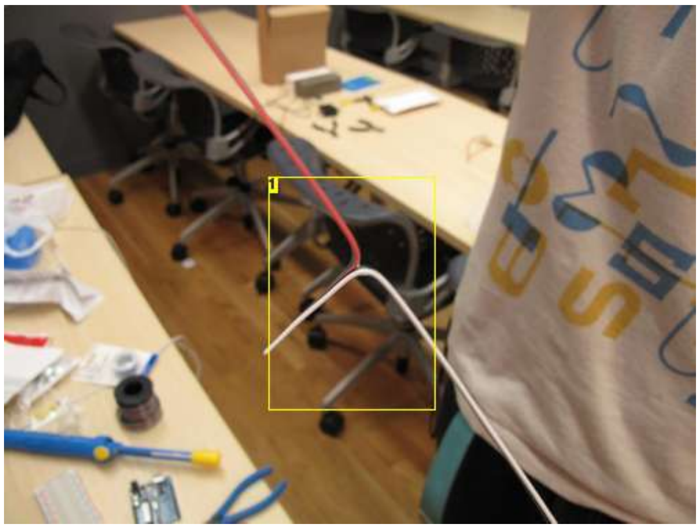


Image Notes
1. Cut about 2 feet (60 centimeters) of your 4-color intercom wire, and split the red and green from the black and white.

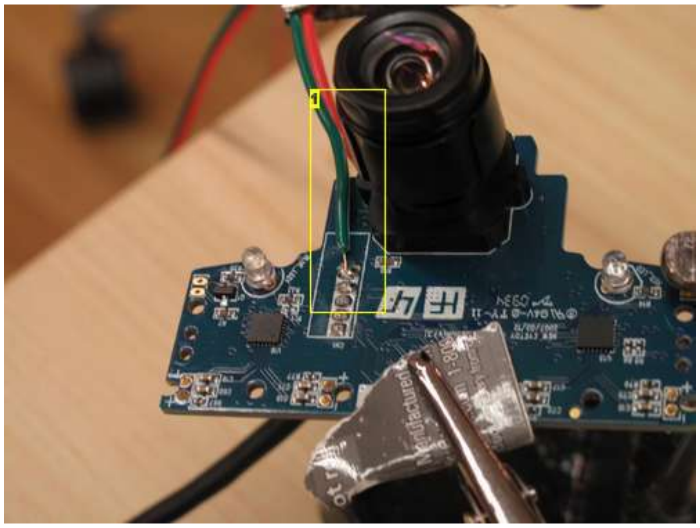


Image Notes
1. Split the red and green wire about 2 inches (5 centimeters) from one end, and strip off a small section of insulation at the end of the green wire. The green wire will be soldered to the PS Eye s Ground joint. Clip the PS Eye board and green wire to a stand, and prepare to solder the green wire tip to the Ground joint. Use a piece of thick paper or cardboard in between the clip s teeth to prevent scrapes on the board.

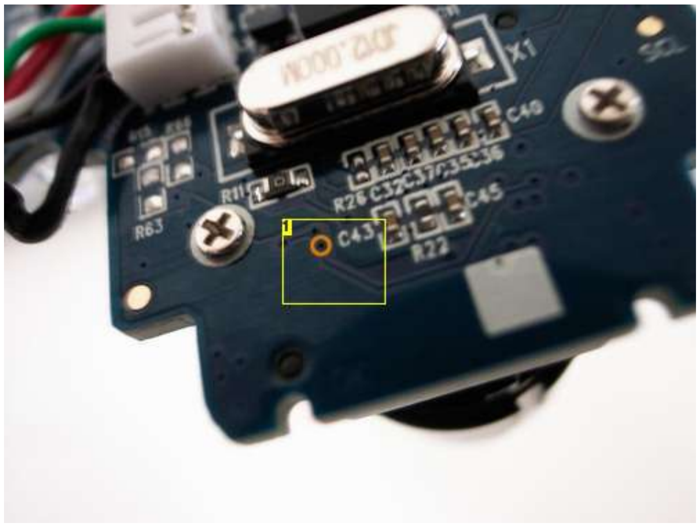


Image Notes
1. Locate the V-Sync via on the board. It s the via circled in the image below.

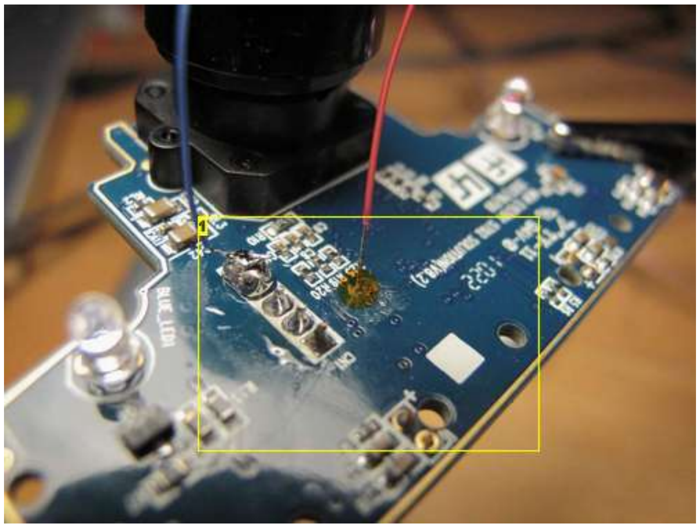


Image Notes
1. for more recent models of the PSEye camera (identified by the golden rim around the board) the VSync hotspot can be found on the front of the PCB, directly above the R19 resistor

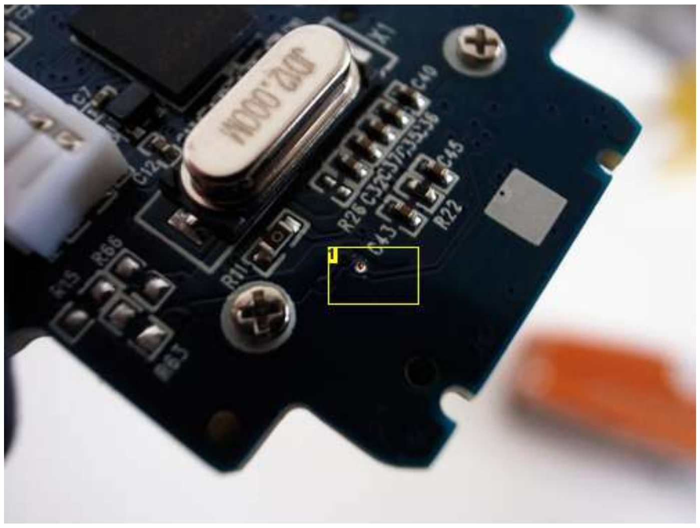


Image Notes
1. Using a sharp knife, carefully pivot the knife tip on the via, and scrape off enough insulation coating to expose the metal contact below.

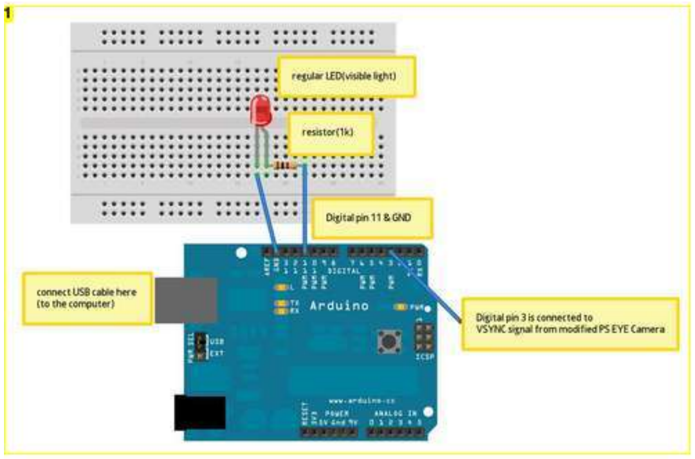

regular LED(visible light)
resistor(1k)
Digital pin 11 & GND
connect USB cable here (to the computer)
Digital pin 3 is connected to VSYNC signal from modified PS EYE Camera
Arduino

Image Notes

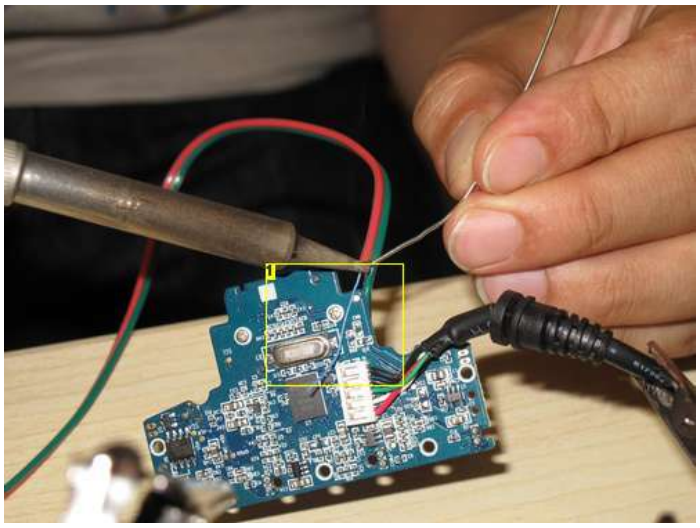
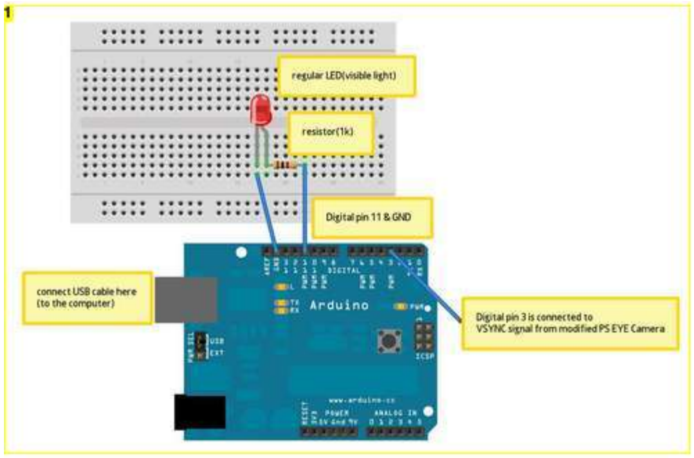


Image Notes
1. The red wire needs to connect to the exposed V-Sync via, but the wire is too thick to be soldered neatly to the small via, so a 30 gauge wire will be used in between. Strip the ends of a 2 piece of 30 gauge wire. Shorten the red wire, then solder one end of the 30 gauge wire to the end of the red wire.


1
regular LED(visible light)
resistor(1k)
Digital pin 11 & GND
connect USB cable here (to the computer)
Arduino
Digital pin 3 is connected to VSYNC signal from modified PS EYE Camera

1. Before soldering the 30 gauge wire to V-Sync, a test should be performed to ensure all connections are correct. Build the circuit below. When the 30 gauge wire contacts the V-Sync via, the LED on the breadboard should flicker rapidly.

Step 10: Hacking the PS Eye camera - finishing

In this step we will talk about how to put your camera back into one piece.

- Unscrew the 2 screws holding the lens in place. Be careful not to break the fragile V-Sync connection. Detach the lens and keep both screws. see image here
- Measure the square opening of the new lens mount. Cut a square from the filter sheet that is minutely smaller, and place it into the lens mount opening. see image here
- With the filter in place, screw in the new lens mount. This will require some force, and one screw will go in at an angle because the new lens mount is a little too big for the board. see image here
- Screw the new lens into the new lens mount on the board.
- Use hot glue to cover and secure the V-Sync connection. see image here

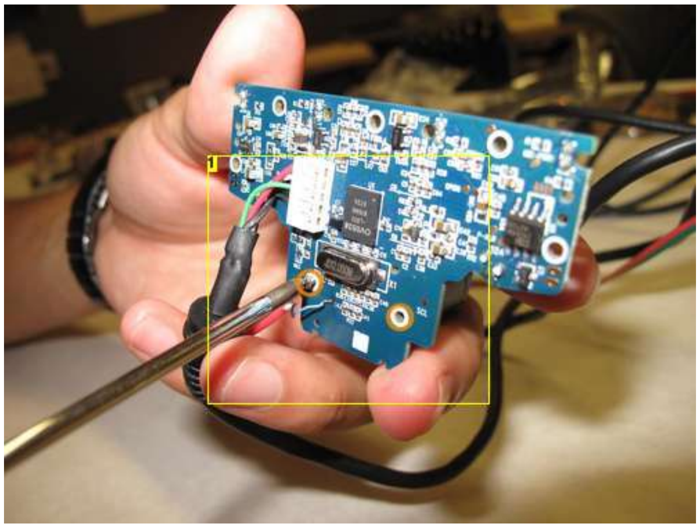


Image Notes
1. Unscrew the 2 screws holding the lens in place. Be careful not to break the fragile V-Sync connection. Detach the lens and keep both screws.

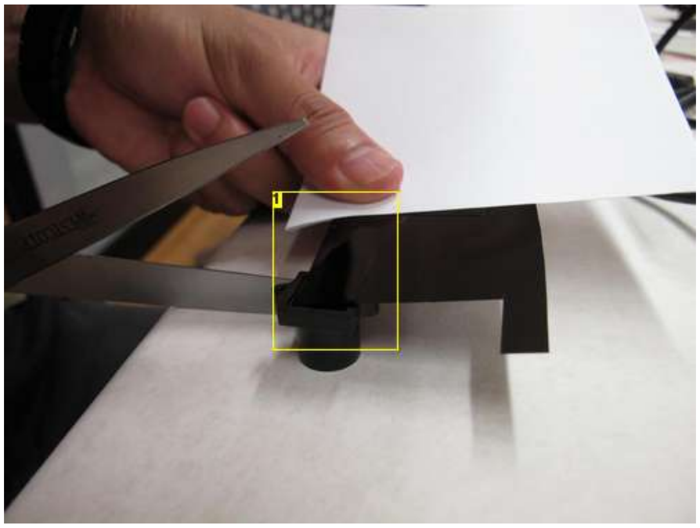


Image Notes
1. Measure the square opening of the new lens mount. Cut a square from the filter sheet that is minutely smaller, and place it into the lens mount opening.

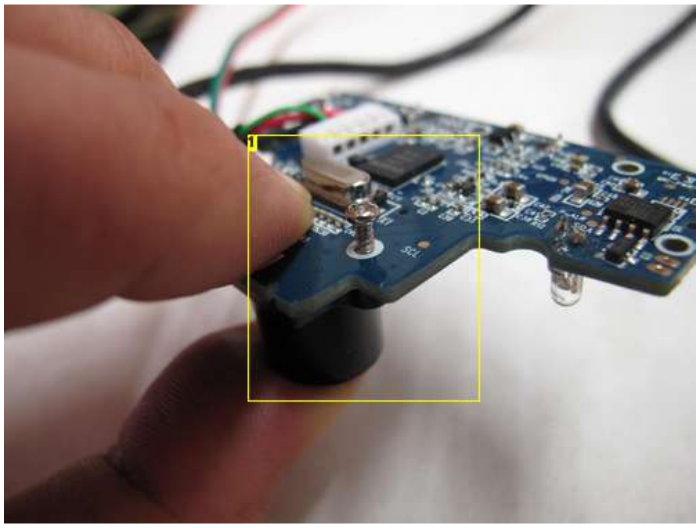


Image Notes
1. With the filter in place, screw in the new lens mount. This will require some force, and one screw will go in at an angle because the new lens mount is a little too big for the board.

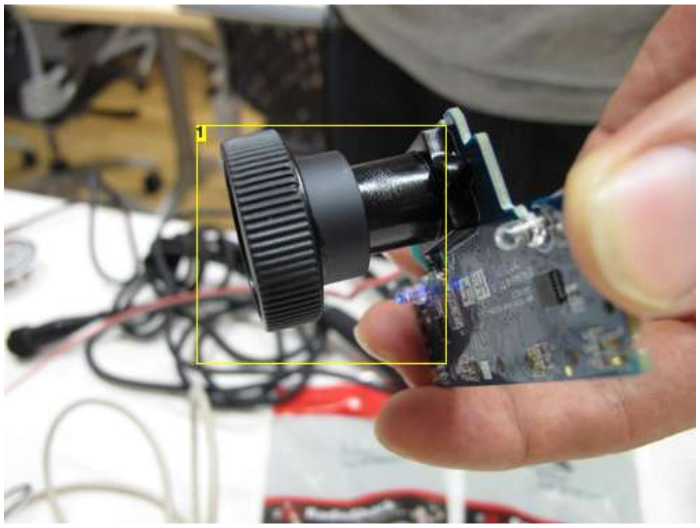


Image Notes
1. Screw the new lens into the new lens mount on the board.

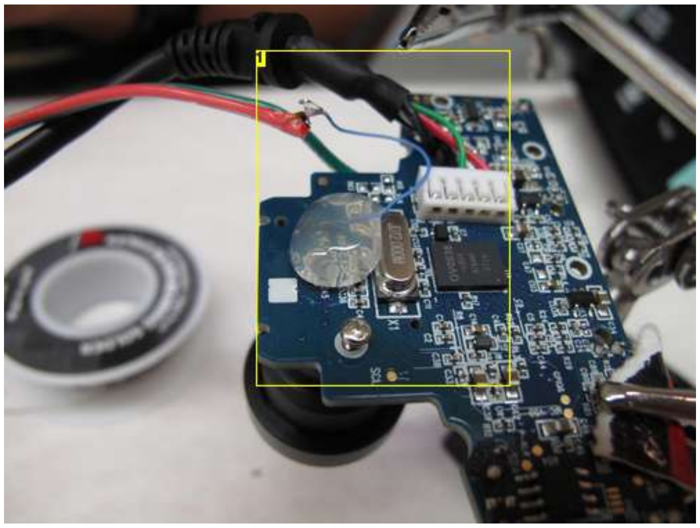
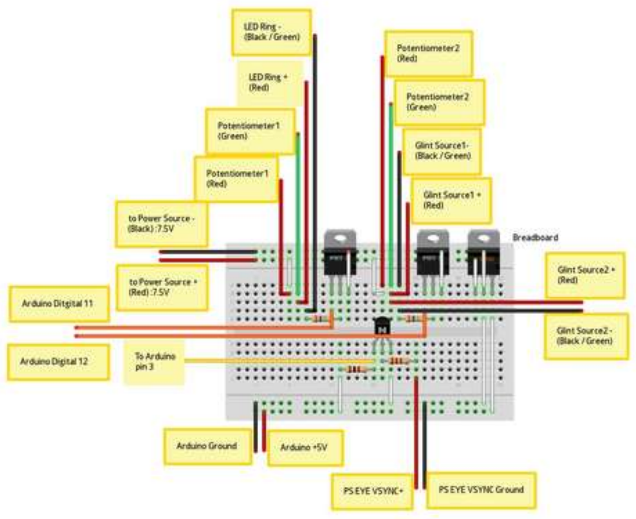


Image Notes
1. Use hot glue to cover and secure the V-Sync connection.

Step 11: Full Circuit

In this step we will show how to put together the circuit on the breadboard. This is the initial step to getting your Arduino to work with the eyeWriter software.

- Build the circuit in the schematic below. see schematic here
- After assembling the full circuit, the EyeWriter code is ready for live camera input. To switch from video demo mode to live camera mode, open the "apps/eyewriter-xxxxxxx/eyeWriterTracker/bin/data/Settings/inputSettings.xml", and edit the mode tag from 1 to 0.
- Open the "apps/eyewriter-xxxxxxx/eyeWriterTracker/RemoteEyeTracker.xcodeprof" file, and Build and Run the source code. The Tracking screen should load with input from the PS Eye camera.



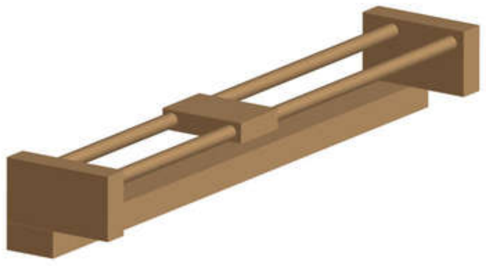
Step 12: Building a wood base

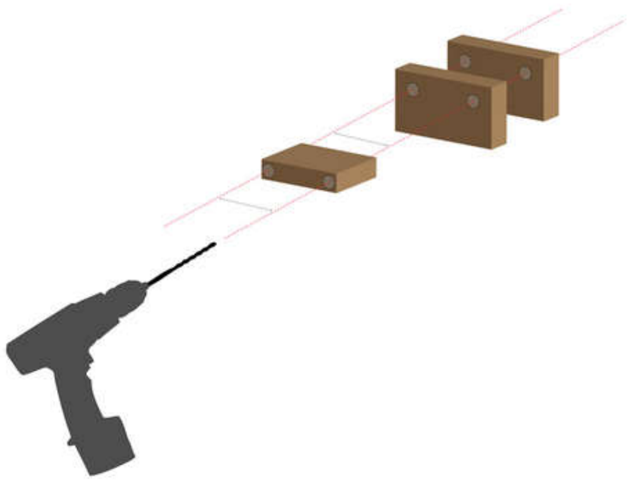
In this step we talk about building a portable wood base. It is interesting to build this so that your system can have a stable infrastructure to rest on. This makes it easier to test, calibrate, and work with the eyeWriter.

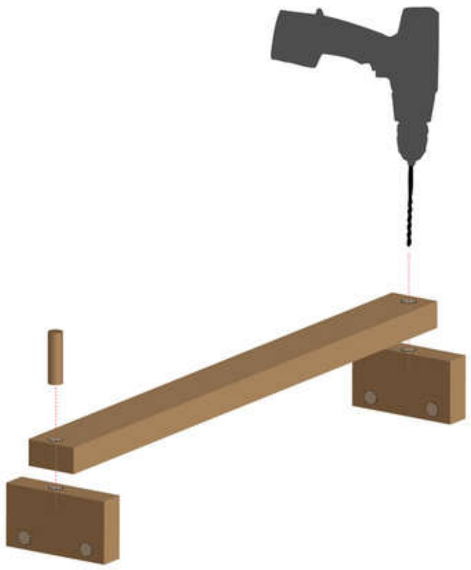
list of materials/parts needed for the base:

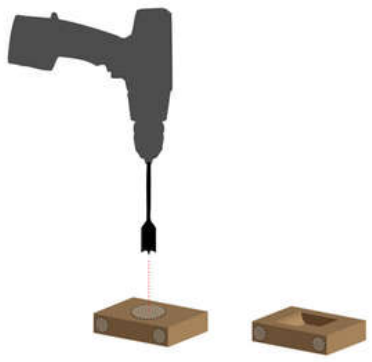
- 2* 5/16 wood rods - approx. 20 inches long (A)
- 2* 5/16 wood rods - approx. 1 1/2 inch long (D)
- 1* 20 x 4 x 1/2 inch wood piece (B)
- 3* 3 x 1 3/4 x 1 3/4 inch wood pieces (C)
- drill bit with approx. diameter of the wood rods

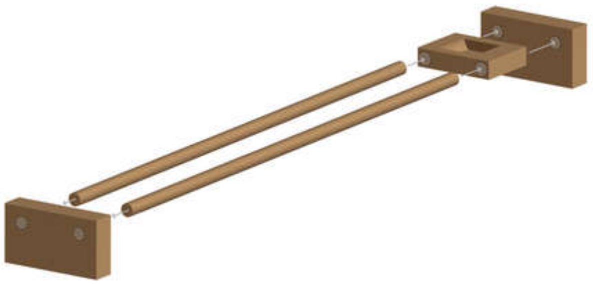
- step 1:
  align the 2 pieces (C) with the third piece (C) as shown in the picture, clamp them together and drill through them at approx. 3/4inch close to the edge. see image here
- step 2:
  using the two pieces (C) that have the same holes aligned, place each of them on the edges of the piece (B), clamp the aligned (see picture for example) and drill a hole through them till about 1 1/2 inch deep on the (B) piece.
  use the short wood rods and put them throught the holes in the piece(B) edges and through each of the pieces(C) see image here
- step 3:
  drill a hole with enough diameter for the tripod head mount screw see image here v
- step 4:
  with the bottom bar(B) and edge pieces (C) assembled, insert the rods (A) through the holes aligning them with the bottom bar length. see image here









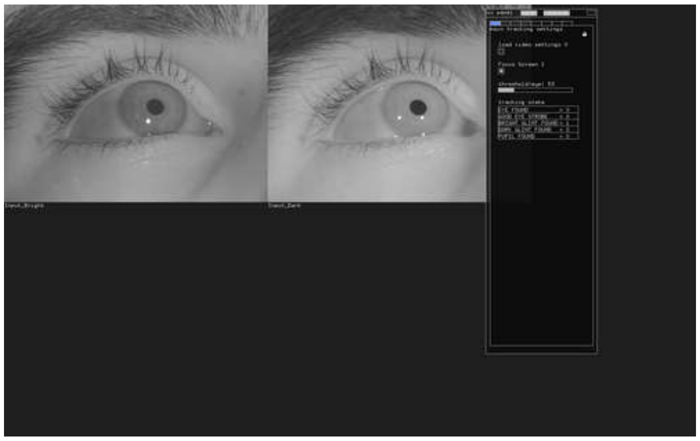
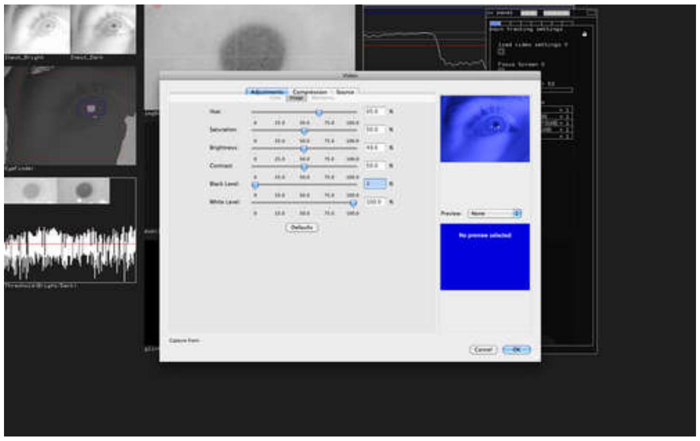


Step 13: Using EyeWriter Software - Setup & Tracking Screen

In this step we will take you around the eyeWriter software so that you can set it up.

- Focus your camera by selecting Focus Screen on the first tab of the Computer Vision (CV) panel on the right. Rotate the lens of your camera until both video feeds look sharp, then deselect Focus Screen to return to the Tracking screen.
- Select load video settings on the first tab of the right panel.
  see image here

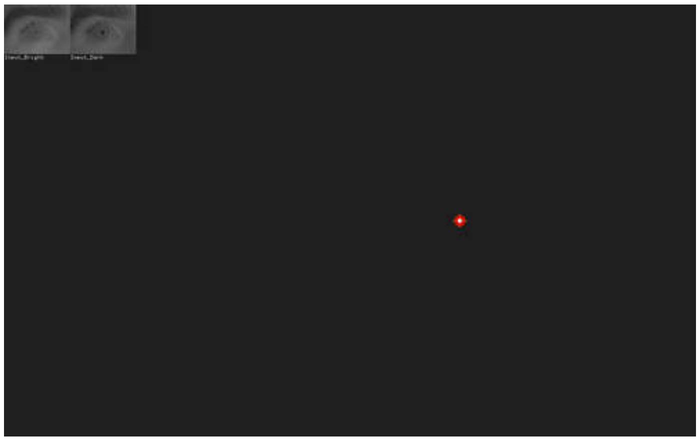
  For PS3 Eye Camera:
  Ideally you want a bright, balanced image with minimal noise. An example image is shown below. Under the Webcam tab, slide the Gain and Shutter settings back and forth until the video looks ideal.
- Under the Compression tab, if youre using a faster computer set your Frames per second (Fps) to 30. If youre using a slower computer set your Fps to 15.

Step 14: Using EyeWriter Software - Calibration Screen

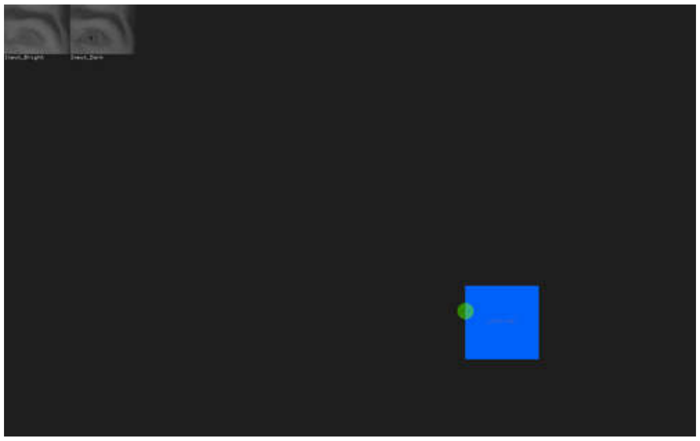
In this step we will go through the calibration setup.

- Press spacebar for instructions, then spacebar again to start. Look at the red dots as they appear.
- At the end of the calibration, the blue lines show any calibration inaccuracies. If there are any long blue lines, reset the calibration and press spacebar to start again.



Step 15: Using EyeWriter Software - Catch Me

- Stare at the Catch Me box. As you stare, the boxs color will turn green.
- When the box is fully green, it is caught and will appear somewhere else. Keep catching the boxes to test your eye-tracking calibration.



Step 16: Using EyeWriter Software - Drawing

LETTER DRAWING

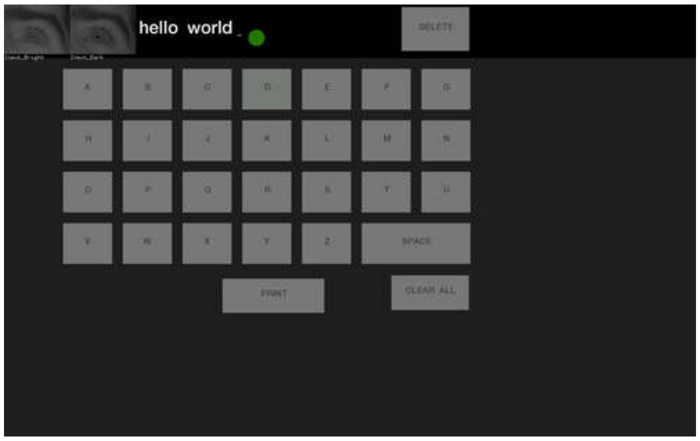
- Drawing mode starts paused by default. Before you start drawing, you can toggle the background grid on or off. The background grid can be toggled at any time by pausing.
- To start drawing, switch to recording mode by staring at the paused button. As you stare, the button will turn green and switch to recording.
- Drawing works with vector points. Stare at a place on the canvas for about a second to make a point. Your green eye-tracking circle needs to stay very still to make a point.
- As you add points, they will stay connected by a stroke. You can create shapes and letters with these strokes. To make a new line, stare at the next stroke button.
- To change what you are drawing, you can undo point which removes the last point drawn, or undo stroke which removes the entire last stroke drawn.
- To save the current shape or letter and move on to the next one, stare at next letter. Your recently drawn letter will appear at the top of the screen, and you have a new blank canvas to draw a new letter on.
- When you are finished drawing shapes and letters, stare at NEXT MODE to move on to Positioning mode.

POSITIONING

- By default, all your letters are selected and ready for positioning. You can select individual letters by staring at Select Letter.
- Select Rotate allows you to rotate your selection right (clockwise) and left (counter-clockwise).
- Select Shift allows you to move your selection up, down, left and right.
- Select Zoom allows you to zoom out and zoom in which shrinks and enlarges your selection.
- Auto Place will place your letters side by side in the order you drew them.
- When you are finished positioning your shapes and letters, stare at NEXT MODE to move on to Effects mode.

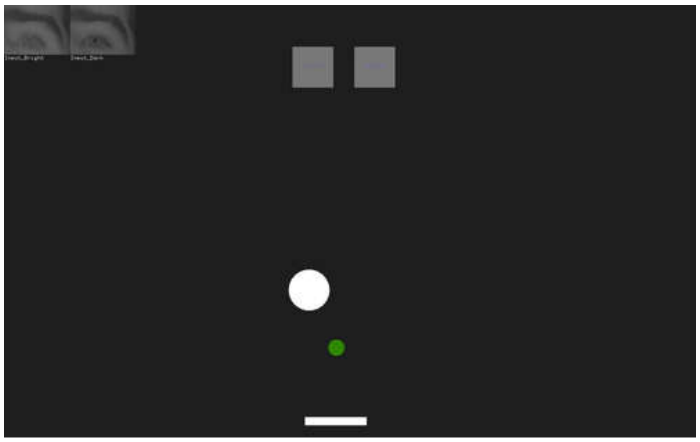
Step 17: Using EyeWriter Software - Typing

- Stare at whichever key you wish to press. As you stare, the keys color will turn green then flash blue.
- When the key flashes blue, it has been pressed. You can see what youve typed at the top of the screen.
- To speak the words typed, press the SPEAK key on the bottom left of the screen. On the middle right of the screen, SPEAK WORDS OFF/ON toggles the option to speak words automatically after they are typed and a space is entered.
- Note the CAPS OFF/ON key on the bottom right of the screen. This toggles caps lock on and off, and is required to use the alternate characters on the number keys (! @ # $ % etc).

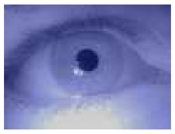


Step 18: Using EyeWriter Software - Pong

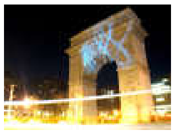
- The goal is to block the ball from passing your paddle at the bottom of the screen.
- The paddle will slide aligned with the x-position of your gaze. So you can stare at the moving ball and the paddle will slide horizontally in tandem.

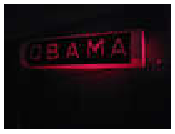


Related Instructables


The EyeWriter by Q-Branch

How To Start Your Own Graffiti Research Lab by fi5e

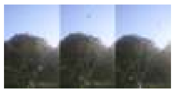

PROJECTION BOMBING by fi5e



LED Throwie - Instant Messanger Tube by adambehman

Corpse Bride Wedding Photo by kristylynn84

How 2.0: Hack a Bat - the Ryan Howard Speed Test by 2pointhome


Build a USB Orange Thrower Machine by XicoMBD

LASER GRAFFITI by luifer78

Twitter Mood Light - The World's Mood in a Box

by RandomMatrix on April 22, 2010

Intro: Twitter Mood Light - The World's Mood in a Box

How's the world feeling right now? **This box tells you** .

Powered by: an **Arduino,** a **WiFly wireless module,** an **RGB LED, Twitter.com** and a **9v battery.**

I'm a news junkie. I want to know everything that is going on in the world as soon as it happens. I want to wake up and know immediately if something big has happened overnight.

However, I'm an extraordinarily busy man; I don't have time to read news feeds; reading that headline that I already knew about or don't care about is time that I'm never getting back!

No. What I need is some way to be constantly in touch with the world's events as they unfold, alerted when something big happens, and to be made aware of it all faster than awareness itself!

...A way to get a glimpse of the collective human consciousness as an extension of my own. Something that I don't have to continually check or poll, but instead, like a part of my body, it will tell me when it's feeling pain or generally in need of my attention ...leaving me time to get on with other things.

And so, I present: **The World Mood in a Box!**

The Arduino connects directly to any wireless network via the WiFly module, continually searches Twitter for tweets with emotional content, collates the tweets for each emotion, does some math, and then fades the color of the LED to reflect the current World Mood; **Red for Anger, Yellow for Happy, Pink for Love, White for Fear, Green for Envy, Orange for Surprise,** and **Blue for Sadness.**

If an unexpectedly high number of tweets of a particular emotion are found, then the LED will flash to alert us to the possibility of a world event that has caused this unusually strong emotional reaction.

For example, a world disaster and it may flash Blue or Red (sadness or anger), if the strong favourite loses a big football game it may fade to Orange (surprise), …and If i flashes White, the collective human mind is feeling extreme fear, and it's probably best to go hide in a cupboard and sit it out, waiting for sunnier skies and a return to Yellow or Pink. ...OK, I'm not that busy.

Image Notes
1. The world is happy!

Image Notes
1. uhhoh! the world is full of envy. ...back to bed!

Image Notes
1. World, why so sad?

Step 1: How it works

An Arduino connects directly (no computer required!) to any wireless network via the WiFly module, repeatedly searches Twitter for tweets with emotional content (aka sentiment extraction or tapping into the moodosphere), collates the tweets for each emotion, analyzes the data, and fades the color of an LED to reflect the current World Mood:

- Red for Anger
- Yellow for Happy
- Pink for Love
- White for Fear
- Green for Envy
- Orange for Surprise
- Blue for Sadness

Example search terms to find tweets that may express surprise:

- "wow"
- "can't believe"
- "unbelievable"
- "O_o"

If an unexpectedly high number of tweets of a particular emotion are found, then the LED will flash to alert anyone nearby to the possibility of a big world event that has caused this unusually strong emotional reaction.

Example signals:

- A world disaster and it may flash Blue or Red indicating it best to check a news site to see why everyone is so sad and/or angry.
- If the strong favourite loses a big football game, it may flash Orange to express the surprise at this unlikely event.
- If there is a heat wave in London it might turn Yellow to reflect how much happier people now are.
- If it flashes White, the collective human consciousness is feeling extreme fear and something terrifyingly bad is probably about to happen. Time to hide and/or panic.

Uses

- You could put it on your desk to get an early warning of something big happening somewhere in the world
- A literal 'mood light' at a party or a game whereby you guess what colour it will change to next and for what reason
- A world mood barometer perhaps next to your bed to decide if it is best to hit snooze until it's less angry outside
- A gauge of public sentiment to help you decide when to sell all your stocks and shares, and head to the hills.
- In a foyer or waiting area or other public space for people to look at and contemplate.
- Set it to connect to any wireless network and carry it around in the streets, stopping strangers to explain to them that you have managed to capture the world's mood and have it locked in this here box.

Step 2: All you need is...

I ordered most of the electronics from Sparkfun, and picked up the rest from the local Radioshack. The acrylic I got from a local plastic shop(!) - they cut it and drilled a hole free of charge.

Materials

- Arduino Duemilanove
- Wifly Shield www.sparkfun.com/commerce/product_info.php
- Breakaway headers www.sparkfun.com/commerce/product_info.php
- 9v battery
- 9v to Barrel Jack Adapter
- 5mm RGB LED
- 3x resistors (2x100 ohm,1x180 ohm)
- Wire
- Small printed circuit board
- USB Cable A to B to connect Arduino to computer
- Rosin-core solder
- Source code

The Acrylic Box

- 1 x (5" x 5" x 0.25") - the top
- 4 * (4.75" x 4.75" x 0.25") - the 4 walls
- 1 x (4.5" x 4.5" x 0.25") - the base
- 1 x (4.5" x 4.5" x 0.125") - the mirror with a 6mm hole drilled in the middle
- 4 x (4.25 x 1" x 0.25") - the 4 inside walls
- Acrylic solvent cement
- Sand paper (to help diffuse the light)

Tools

- Soldering iron
- A computer
- Arduino development environment
- A wireless network (802.11b/g)
- Pliers
- Wire stripper

Useful links

The Arduino development tools can be downloaded from here:
www.arduino.cc/en/Main/Software

and Arduino tutorials start here:
http://arduino.cc/en/Guide/HomePage

Arduino / WiFly:
arduino.cc/en/Reference/HomePage
http://www.arduino.cc/en/Tutorial/SPIEEPROM
http://www.lammertbies.nl/comm/info/serial-uart.html
http://www.tinyclr.com/downloads/Shield/FEZ_Shields_WiFly.cs
http://www.sparkfun.com/commerce/tutorial_info.php?tutorials_id=158

http://www.sparkfun.com/datasheets/Components/SMD/sc16is750.pdf
http://www.sparkfun.com/datasheets/Wireless/WiFi/WiFlyGSX-um.pdf
http://www.sparkfun.com/datasheets/Wireless/WiFi/rn-131G-ds.pdf
http://www.societyofrobots.com/microcontroller_uart.shtml

Related:
nlp.stanford.edu/courses/cs224n/2009/fp/22.pdf
www.webservius.com/corp/docs/tweetfeel_sentiment.htm
i8news.uterm.org/mood/twitter-mood-reader/
community.openamplify.com/content/docs.aspx/
www.instructables.com/id/The-Twittering-Office-Chair/
http://www.tweetfeel.com

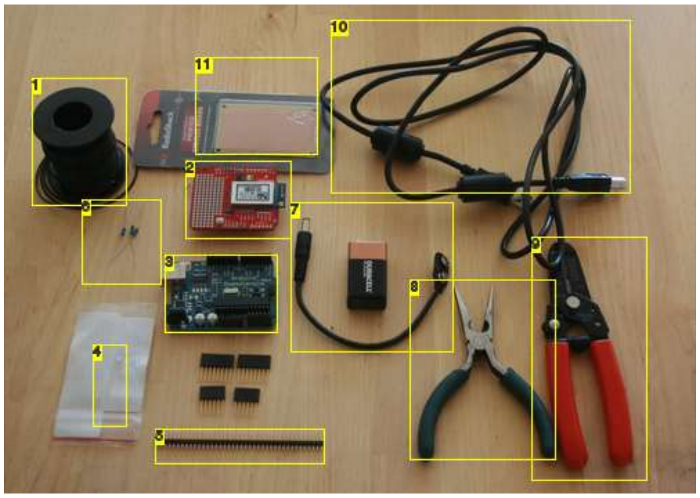


Image Notes
1. wire
2. Wifly Shield
3. Arduino Duemilanove
4. RGB LED
5. Breakaway headers
6. 3x resistors (2x100 ohm,1x180 ohm)
7. 9v battery and Barrel Jack Adapter
8. Pliers
9. Wire cutters
10. USB Cable A to B
11. small printed circuit board

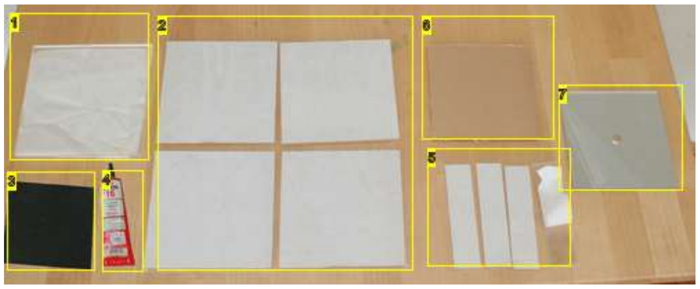


Image Notes
1. 1 x (5" x 5" x 0.25") - the top
2. 4 * (4.75" x 4.75" x 0.25") - the 4 walls
3. sandpaper
4. Acrylic solvent cement
5. 4 x (4.25 x 1" x 0.25") - the 4 inside walls
6. 1 x (4.5" x 4.5" x 0.25") - the base
7. 1 x (4.5" x 4.5" x 0.125") - the mirror with a 6mm hole drilled in the middle

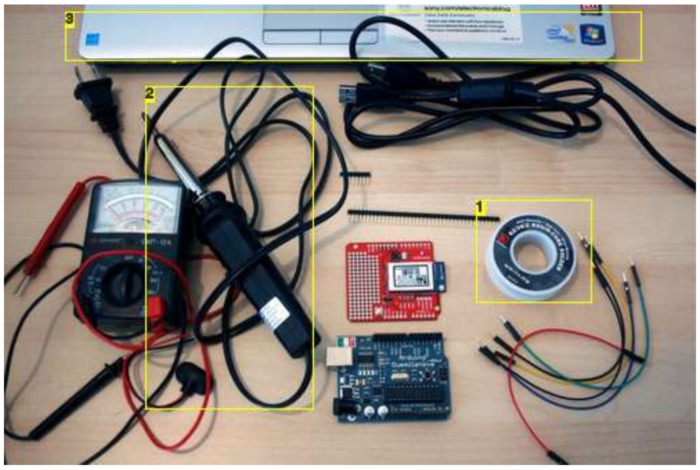


Image Notes
1. solder
2. soldering iron
3. laptop

Step 3: Connect the Arduino and WiFly to a computer

Sparkfun have a decent tutorial on how to do this:

www.sparkfun.com/commerce/tutorial_info.php

Firstly, the Wifly breakout board needs to be stacked on top of the arduino and the RX, TX, Vin, Gnd, pin 10, pin 11, pin 12 and pin 13 needed to be connected. I used breakaway headers and soldered the required pins.

Connect to a computer using an A to B USB cable.

Download the Arduino software from here:
arduino.cc/en/Main/Software

Check that you can compile and upload a sample program by following the instructions here:
(remember to set the board and COM ports correctly)
arduino.cc/en/Guide/HomePage

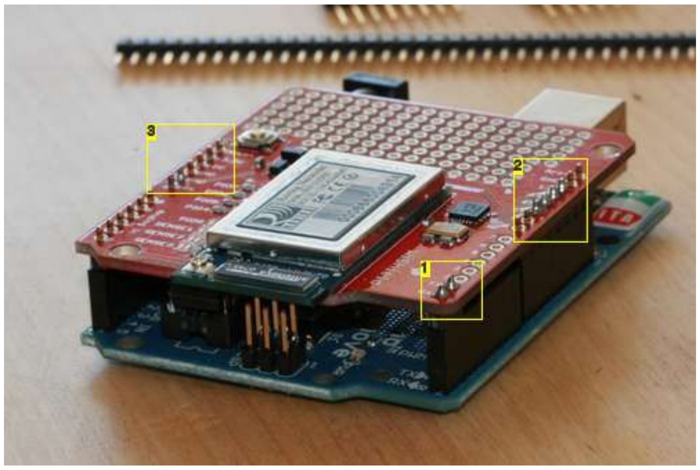
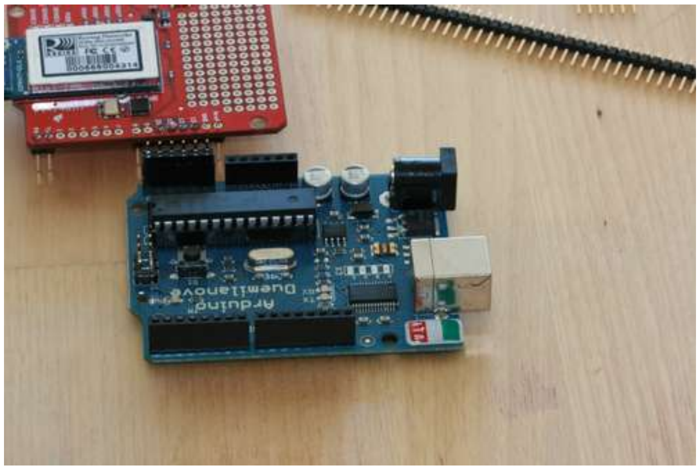
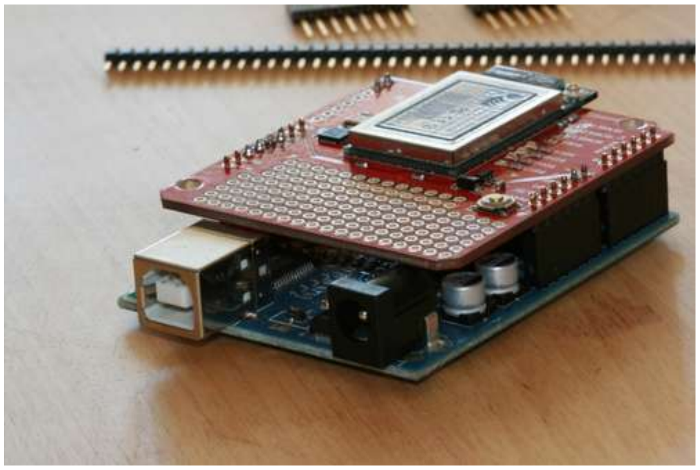
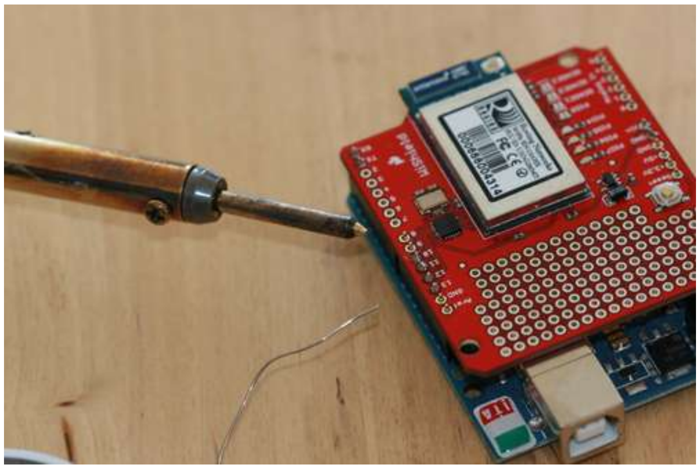
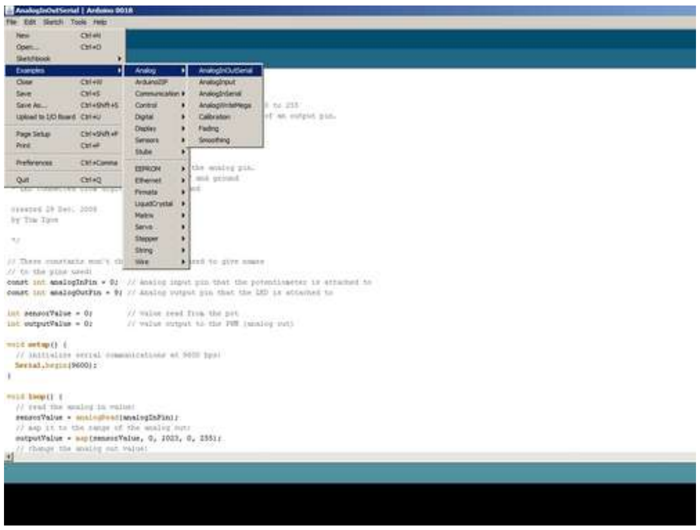


Image Notes
1. soldered RX, TX
2. Soldered 10,11,12,13
3. Soldered Vin, Gnd









```
 created 29 Dec. 2008
 by Tom Igoe

 */

// These constants won't ch                 ed to give names
// to the pins used:
const int analogInPin = 0;  // Analog input pin that the potentiometer is attached to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
```

Step 4: Connecting the LED

Only some pins provide 8-bit PWM (Pulse-width modulation)
This gives 256 steps of control from full off (0) to full on (255) for each of the Red, Green and Blue channels of the LED.

PWM pins on the Arduino are 3,5,6,9,10,11. (see www.arduino.cc/en/Main/ArduinoBoardDuemilanove)

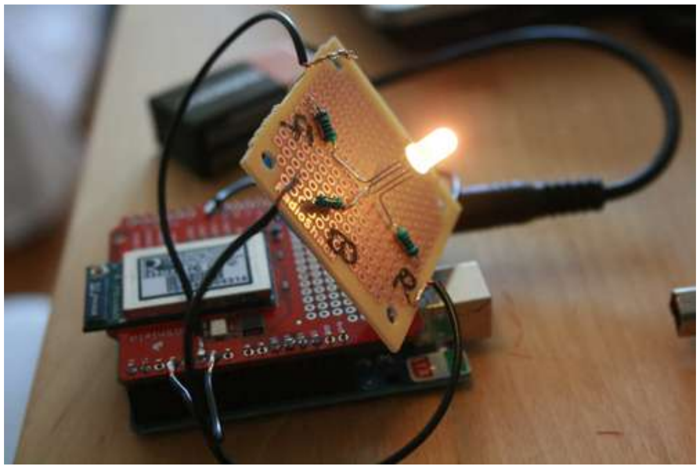
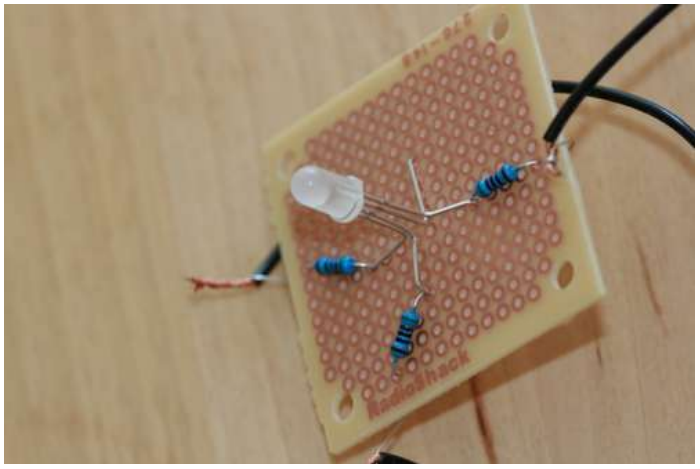
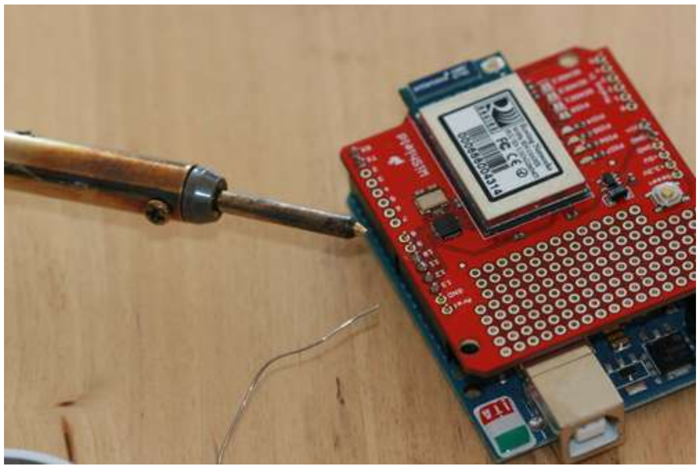
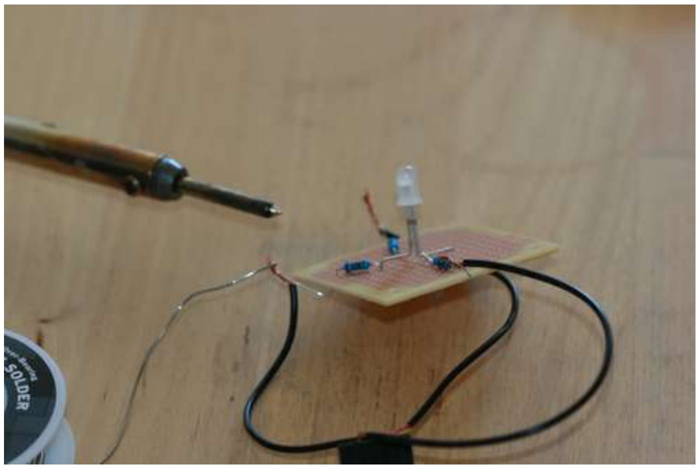
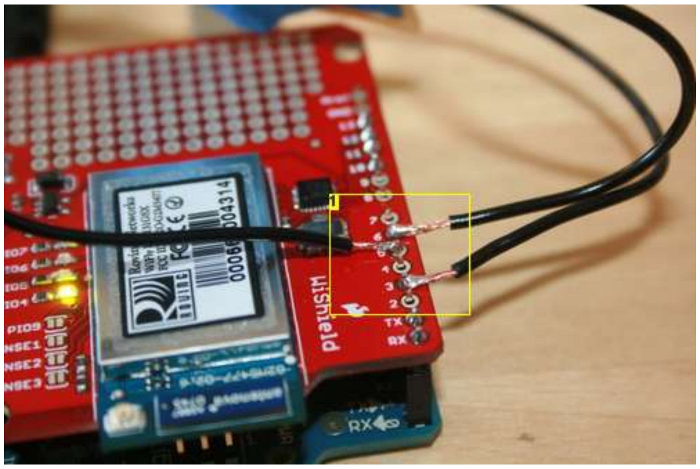
I used 3, 5 and 6.
I used the pliers to bend the legs of the LED, and mounted it on the circuit board. Each resistor is then mounted next to each of the RGB legs, and the wires are twisted together. Then I added the 4 connecting wires and twisted them. Finally, I soldered all the connections.

Note: The pictures illustrate using the same resistor for each colour channel, but I should have used the resistance levels in the data sheet:

180 Ohm for Red
100 Ohm for Green
100 Ohm for Blue

Also note, I covered the back with insulating tape to stop any shorts when putting it all into the box.
Also, from the datasheet, "the Sensor inputs SENS0-7 are extremely sensitive to over voltage. Under no conditions should these pins be driven above 1.2VDC. Placing any voltage above
this will permanently damage the radio module and render it useless."

wiring.org.co/learning/basics/rgbled.html
www.sparkfun.com/datasheets/Components/YSL-R596CR3G4B5C-C10.pdf

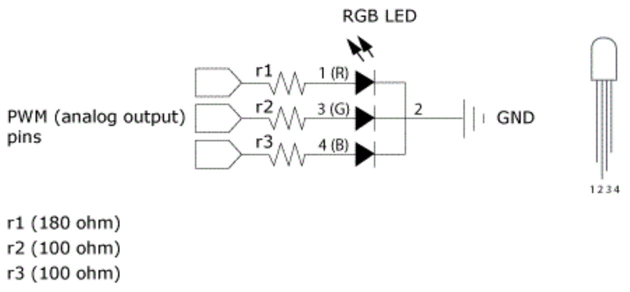


Image Notes
1. PWM pins 3,5,6

Step 5: Choosing good search terms

Twitter allows you to search for recent tweets that contain particular words or phrases.

You can search for tweets that contain any of a list of phrases by using the "+OR+" conjunction.

For example, here is a search request that might find tweets that express Fear:

GET /search.json?q="i'm+so+scared"+OR+"i'm+really+scared"+OR+"i'm+terrified"+OR+"i'm+really+afraid"+OR+"so+scared+i"&rpp=30&result_type=recent

I spent a long time finding good search phrases.

The search phrases needed to produce tweets that:

1. very often express the desired emotion.

2. very rarely express the opposite emotion or no emotion.

Many search phrases that I thought would work, turned out to not work that well when I searched with them.

Smileys have been used with some success to extract whether the sentence is positive or negative, but I didn't find them useful for extracting anything more.

The trouble with smileys is that a smile can mean so many things ;D

It is often used, it seems, as a kind of qualifier for the whole sentence; since people have to compress their thoughts into 140 characters, the meaning can become ambiguous.

The smiley often then acts as a qualifier that:

- 'this is a friendly comment'
- 'don't take this the wrong way'
- 'i am saying hello/goodbye with a smile'
- 'this is almost a joke'
- 'I know I'm being cheeky'
- 'I don't really mean this'

Phrases using **adverbs** seemed to produce better results.
"so scared" or "really scared" is better than just "scared" which returns bad results: for example, "not scared".

Phrases in the **first person** seemed to produce better results.
Some search phrases give tweets that suggest the author feels the emotion: for example, "i really hate...", often sounds like they really are full of hate or angry, whereas other phrases containing the word "hate" give tweets that do not seem to express much emotion, like "why do you hate..."

Hyperbole is your best friend, ever:
Using phrases with hyperbole produced good results. Tweets with "I'm terrified" or "I'm petrified" in them were generally more fearful sounding than "I'm scared"

Regardless, the approach is still naive, but statistically, from my tests, it does seem to work well.

While testing the code, I did at one point get the horribly ominous "Flashing White" that signifies the world is feeling intense **fear** , but since I was still testing it all, I did not hide under the table straight away, but instead, threw caution to the winds, and went on to Twitter to see what people were suddenly so fearful about.

The recent tweets containing the Fear search string (see top of page) were largely relating to a large thunderstorm that had just started somewhere near Florida.

If you're interested, here are some of those tweets:

- "Ahhh Thunder I'm so scared of Thunder !!!!! Help some 1"
- "I'm so scared of lightning now. Like I just ran home praying "
- "On our way to Narcosses at @Disney world's Grand Floridian hotel and there's a tropical storm right now. I'm terrified! ..."
- "I'm in my bathroom til the rain stops. I'm terrified of lightning and thunder..."
- "I'm terrified of thunder storms *hides in corner*"
- "I'm terrified of Thunder :("
- "If only I was wit my becky during this thunderstorm cause I'm really scared cause of a bad experience"

So... it works! ...Well, it needs the numbers tweaking to ignore the world's "tantrums", the short-lived fits of emotional outburst, and be more concerned with larger changes that signify bigger news.

Image Notes
1. Scary!! http://www.flickr.com/photos/thru_the_night/3807826324/

Step 6: Download the code

The attached WorldMood.zip contains 4 subdirectories (or "libraries") and the Arduino sketch WorldMood.pde

The four libraries need to be copied into the Arduino library directory and then they can be imported as shown.

WorldMood/WorldMood.pde (see below) should be opened in the Arduino development environment.

You then need to correct the "[your network]" and "[your network password]" fields. eg.

```
#define network ("mynetwork")
#define password ("mypassword")
```

Then the sketch (and libraries) should be compiled and uploaded to the Arduino board.

see arduino.cc/en/Hacking/LibraryTutorial

The next 5 programming steps just give an overview of each of the components and include the most noteworthy parts of the source code...

****** Update ******

If you have a newer board then you may need to change this

```
struct SPI_UART_cfg SPI_Uart_config = {0x50,0x00,0x03,0x10};
```

to this:

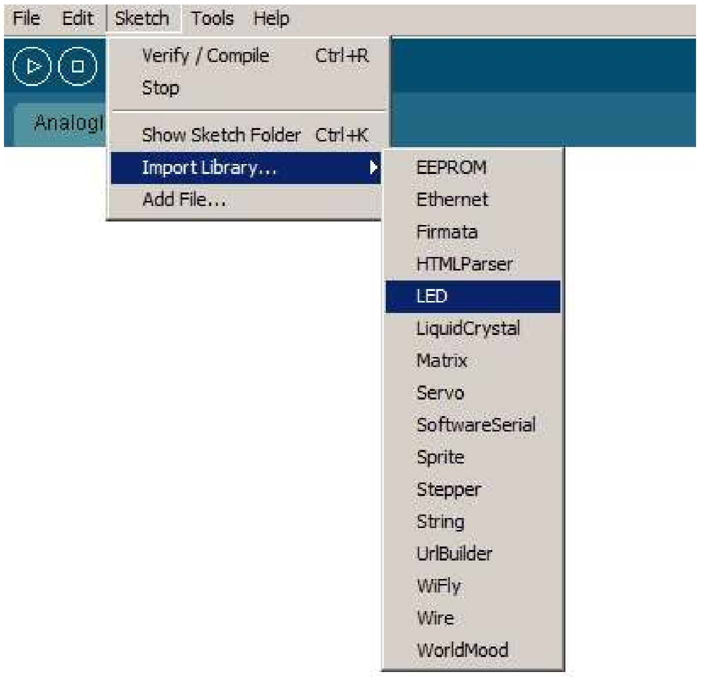
```
struct SPI_UART_cfg SPI_Uart_config = {0x60,0x00,0x03,0x10};
```

See here for more info:
http://forum.sparkfun.com/viewtopic.php?f=13&t=21846&sid=24282242d4256db0c7b7e814d7ca6952&start=15

http://www.sparkfun.com/commerce/product_info.php?products_id=9367

******* End Update ******

```
// LED setup - only some pins provide 8-bit PWM (Pulse-width modulation)
// output with the analogWrite() function.
// http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove
// PWM: 3,5,6,9,10,11
#define  redPin (3)
#define  greenPin (5)
#define  bluePin (6)
// delay in ms between fade updates
// max fade time = 255 * 15 = 3.825s
#define  fadeDelay (15)
// Wifi setup
#define  network ([your network])
#define  password ([your network password])
#define  remoteServer ("twitter.com")
const  char* moodNames[NUM_MOOD_TYPES] = {
"love",
"joy",
"surprise",
"anger",
"envy",
"sadness",
"fear",
};
const  char* moodIntensityNames[NUM_MOOD_INTENSITY] = {
"mild",
"considerable",
"extreme",
};
// the long term ratios between tweets with emotional content
// as discovered by using the below search terms over a period of time.
float  tempramentRatios[NUM_MOOD_TYPES] = {
0.13f,
0.15f,
0.20f,
0.14f,
0.16f,
0.12f,
0.10f,
};
// these numbers can be tweaked to get the system to be more or less reactive
// to be more or less susceptible to noise or short term emotional blips, like sport results
// or bigger events, like world disasters
#define  emotionSmoothingFactor (0.1f)
#define  moodSmoothingFactor (0.05f)
#define  moderateMoodThreshold (2.0f)
#define  extremeMoodThreshold (4.0f)
// save battery, put the wifly to sleep for this long between searches (in ms)
#define  SLEEP_TIME_BETWEEN_SEARCHES (1000 * 5)
// Store search strings in flash (program) memory instead of SRAM.
// http://www.arduino.cc/en/Reference/PROGMEM
// edit TWEETS_PER_PAGE if changing the rpp value
prog_char  string_0[] PROGMEM = "GET
/search.json?q=\"i+love+you\"+OR+\"i+love+her\"+OR+\"i+love+him\"+OR+\"all+my+love\"+OR+\"i'm+in+love\"+OR+\"i+really+love\"&rpp=30&result_type=recent";
prog_char  string_1[] PROGMEM = "GET
/search.json?q=\"happiest\"+OR+\"so+happy\"+OR+\"so+excited\"+OR+\"i'm+happy\"+OR+\"woot\"+OR+\"w00t\"&rpp=30&result_type=recent";
prog_char  string_2[] PROGMEM = "GET /search.json?q=\"wow\"+OR+\"O_o\"+OR+\"can't+believe\"+OR+\"wtf\"+OR+\"unbelievable\"&rpp=30&result_type=recent";
prog_char  string_3[] PROGMEM = "GET
/search.json?q=\"i+hate\"+OR+\"really+angry\"+OR+\"i+am+mad\"+OR+\"really+hate\"+OR+\"so+angry\"&rpp=30&result_type=recent";
prog_char  string_4[] PROGMEM = "GET /search.json?q=\"i+wish+i\"+OR+\"i'm+envious\"+OR+
\"i'm+jealous\"+OR+\"i+want+to+be\"+OR+\"why+can't+i\"+&rpp=30&result_type=recent";
prog_char  string_5[] PROGMEM = "GET
/search.json?q=\"i'm+so+sad\"+OR+\"i'm+heartbroken\"+OR+\"i'm+so+upset\"+OR+\"i'm+depressed\"+OR+\"i+can't+stop+crying\"&rpp=30&result_type=recent";
prog_char  string_6[] PROGMEM = "GET
/search.json?q=\"i'm+so+scared\"+OR+\"i'm+really+scared\"+OR+\"i'm+terrified\"+OR+\"i'm+really+afraid\"+OR+\"so+scared+i\"&rpp=30&result_type=recent";
// be sure to change this if you edit the rpp value above
#define  TWEETS_PER_PAGE (30)
PROGMEM  const char *searchStrings[] =
{
string_0,
string_1,
string_2,
string_3,
string_4,
string_5,
string_6,
};
void  setup()
{
Serial.begin(9600);
delay(100);
}
void  loop()
{
// create and initialise the subsystems
WiFly wifly(network, password, SLEEP_TIME_BETWEEN_SEARCHES, Serial);
WorldMood worldMood(Serial, emotionSmoothingFactor, moodSmoothingFactor, moderateMoodThreshold, extremeMoodThreshold, tempramentRatios);
LED led(Serial, redPin, greenPin, bluePin, fadeDelay);
TwitterParser twitterSearchParser(Serial, TWEETS_PER_PAGE);
wifly.Reset();
char searchString[160];
while (true)
{
for (int i = 0; i < NUM_MOOD_TYPES; i++)
{
twitterSearchParser.Reset();
// read in new search string to SRAM from flash memory
strcpy_P(searchString, (char*)pgm_read_word(&(searchStrings[i])));
bool ok = false;
int retries = 0;
// some recovery code if the web request fails
while (!ok)
{
ok = wifly.HttpWebRequest(remoteServer, searchString, &twitterSearchParser);
if (!ok)
{
Serial.println("HttpWebRequest failed");
retries++;
if (retries > 3)
{
wifly.Reset();
retries = 0;
}
}
}
float tweetsPerMinute = twitterSearchParser.GetTweetsPerMinute();
// debug code
Serial.println("");
Serial.print(moodNames[i]);
Serial.print(": tweets per min = ");
Serial.println(tweetsPerMinute);
worldMood.RegisterTweets(i, tweetsPerMinute);
}
MOOD_TYPE newMood = worldMood.ComputeCurrentMood();
MOOD_INTENSITY newMoodIntensity = worldMood.ComputeCurrentMoodIntensity();
Serial.print("The Mood of the World is ... ");
Serial.print(moodIntensityNames[(int)newMoodIntensity]);
Serial.print(" ");
Serial.println(moodNames[(int)newMood]);
led.SetColor((int)newMood, (int)newMoodIntensity);
// save the battery
wifly.Sleep();
// wait until it is time for the next update
delay(SLEEP_TIME_BETWEEN_SEARCHES);
Serial.println("");
}
}
```



File Downloads

WorldMood.zip (18 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'WorldMood.zip']

Step 7: Programming step 1: SPI UART

The WiFly Shield equips your Arduino with the ability to connect to 802.11b/g wireless networks.
The featured components of the shield are:

- a Roving Network's RN-131G wireless module
- SC16IS750 SPI-to-UART chip.

Serial Peripheral Interface Bus (or **SPI**) is a "four wire" serial bus capable of high rates of data transmission. A serial bus allows data to be sent serially (synchronously), i.e. one bit at a time, rather than in parallel (asynchronous)

The Universal asynchronous receiver/transmitter (or **UART**) is a type of asynchronous receiver/transmitter, a piece of computer hardware that translates data between parallel and serial forms.

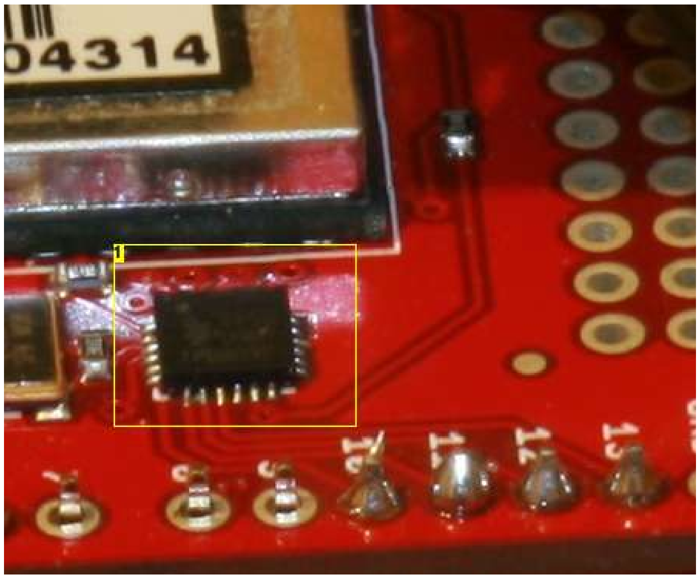
1. The PC communicates over UART with the Arduino through pins RX and TX
2. The Arduino communicates over SPI with the SPI-UART chip on the WiFly shield (SC16IS750 SPI-to-UART chip) though pins 10-13 (CS, MOSI, MISO, SCLK respectively)
3. The RN-131G wireless module accesses network and send/receive serial data over UART.

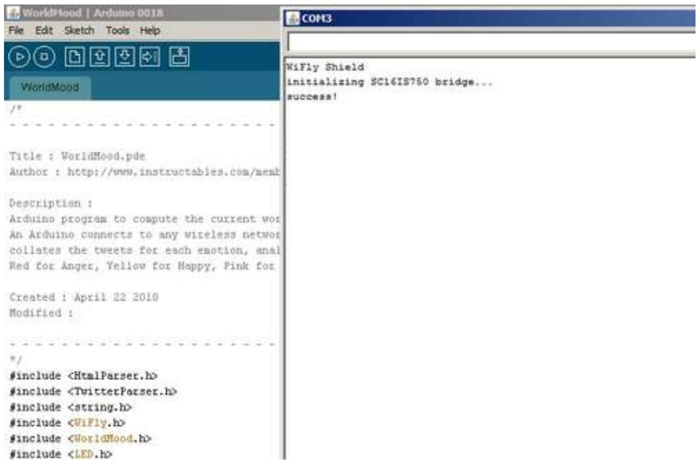
The SPI-to-UART bridge is used to allow for faster transmission speed and to free up the Arduino's UART.

The code below is based on a number of sources, but primarily from this tutorial over at sparkfun:
www.sparkfun.com/commerce/tutorial_info.php
WiFly Wireless Talking SpeakJet Server

```
/* Test if the SPI<->UART bridge has been set up correctly by writing a test
character via SPI and reading it back.
returns true if success
*/
bool  WiFly::TestSPI_UART_Bridge()
{
// Perform read/write test to check if SPI<->UART bridge is working
// write a character to the scratchpad register.
WriteByteToRegister(SPR, 0x55);
char data = ReadCharFromWiFly(SPR);
if(data == 0x55)
{
return true;
}
else
{
```

```
m_printer->println("Failed to init SPI<->UART chip");
return false;
}
}
/* A series of register writes to initialize the SC16IS750 SPI-UART bridge chip
see http://www.tinyclr.com/downloads/Shield/FEZ_Shields_WiFly.cs
*/
void  WiFly::SPI_UART_Init(void)
{
WriteByteToRegister(LCR,0x80); // 0x80 to program baudrate
WriteByteToRegister(DLL,SPI_Uart_config.DivL); //0x50 = 9600 with Xtal = 12.288MHz
WriteByteToRegister(DLM,SPI_Uart_config.DivM);
WriteByteToRegister(LCR, 0xBF); // access EFR register
WriteByteToRegister(EFR, SPI_Uart_config.Flow); // enable enhanced registers
WriteByteToRegister(LCR, SPI_Uart_config.DataFormat); // 8 data bit, 1 stop bit, no parity
WriteByteToRegister(FCR, 0x06); // reset TXFIFO, reset RXFIFO, non FIFO mode
WriteByteToRegister(FCR, 0x01); // enable FIFO mode
}
```





```
/*
- - - - - - - - - - - - - - - - - - - - - -

Title : WorldMood.pde
Author : http://www.instructables.com/memb

Description :
Arduino program to compute the current wor
An Arduino connects to any wireless networ
collates the tweets for each emotion, anal
Red for Anger, Yellow for Happy, Pink for

Created : April 22 2010
Modified :

- - - - - - - - - - - - - - - - - - - - - -
*/
#include <HtmlParser.h>
#include <TwitterParser.h>
#include <string.h>
#include <WiFly.h>
#include <WorldMood.h>
#include <LED.h>
```

```
WiFly Shield
initializing SC16IS750 bridge...
success!
```

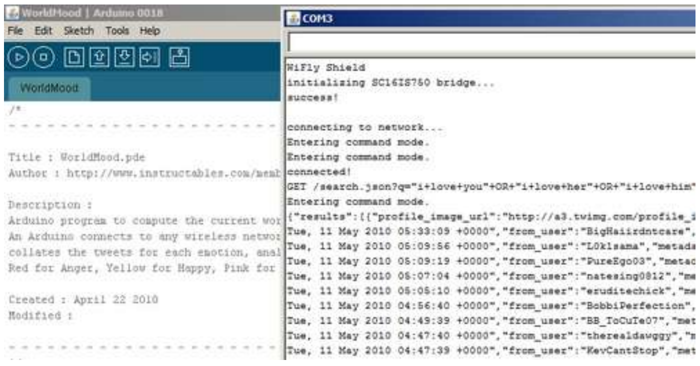
Image Notes
1. the SC16IS750 SPI-to-UART chip!

Step 8: Programming step 2: Connecting to a Wireless Network

Again, this is largely based on the sparkfun tutorial, but I've removed the delays with "waits for response". This speeds things up and is easier to error check.

www.sparkfun.com/commerce/tutorial_info.php

```
/*
Send the correct commands to connect to a wireless network using the parameters used on construction
*/
void  WiFly::AutoConnect()
{
delay(DEFAULT_TIME_TO_READY);
FlushRX();
// Enter command mode
EnterCommandMode();
// Reboot to get device into known state
WriteToWiFlyCR("reboot");
WaitUntilReceived("*Reboot*");
WaitUntilReceived("*READY*");
FlushRX();
// Enter command mode
EnterCommandMode();
// turn off auto joining
WriteToWiFlyCR("set wlan join 0");
WaitUntilReceived(AOK, ERR);
// Set authentication level to
WriteToWiFly("set w a ");
WriteToWiFlyCR(auth_level);
WaitUntilReceived(AOK, ERR);
// Set authentication phrase to
WriteToWiFly("set w p ");
WriteToWiFlyCR(m_password);
WaitUntilReceived(AOK, ERR);
// Set localport to
WriteToWiFly("set i l ");
WriteToWiFlyCR(port_listen);
WaitUntilReceived(AOK, ERR);
// Deactivate remote connection automatic message
WriteToWiFlyCR("set comm remote 0");
WaitUntilReceived(AOK, ERR);
// Join wireless network
WriteToWiFly("join ");
WriteToWiFlyCR(m_network);
delay(DEFAULT_TIME_TO_JOIN);
bool ok = WaitUntilReceived("IP=");
delay(DEFAULT_TIME_TO_WAIT);
FlushRX();
if(ok == false)
{
m_printer->print("Failed to associate with '");
m_printer->print(m_network);
m_printer->println("'\n\rRetrying...");
FlushRX();
AutoConnect();
}
else
{
m_printer->println("Associated!");
ExitCommandMode();
}
// TODO save this configuration
}
/*
Enter command mode by sending: $$$
Characters are passed until this exact sequence is seen. If any bytes are seen before these chars, or
after these chars, in a 1 second window, command mode will not be entered and these bytes will be passed
on to other side.
*/
void  WiFly::EnterCommandMode()
{
FlushRX();
delay(1000); // wait 1s as instructed above
m_printer->println("Entering command mode.");
WriteToWiFly("$$$");
WaitUntilReceived("CMD");
}
/*
exit command mode
send the "exit" command and await the confirmation result "EXIT"
*/
void  WiFly::ExitCommandMode()
{
WriteToWiFlyCR("exit");
WaitUntilReceived("EXIT");
}
```



```
/*
- - - - - - - - - - - - - - - - - - - - -

Title : WorldMood.pde
Author : http://www.instructables.com/memb

Description :
Arduino program to compute the current wor
An Arduino connects to any wireless netwo
collates the tweets for each emotion, anal
Red for Anger, Yellow for Happy, Pink for

Created : April 22 2010
Modified :

- - - - - - - - - - - - - - - - - - - - -
```

```
WiFly Shield
initializing SC16IS750 bridge...
success!

connecting to network...
Entering command mode.
Entering command mode.
connected!
GET /search.json?q="i+love+you"+OR+"i+love+her"+OR+"i+love+him"
Entering command mode.
{"results":[{"profile_image_url":"http://a3.twimg.com/profile_i
Tue, 11 May 2010 05:33:09 +0000","from_user":"BigHaiirdntcare",
Tue, 11 May 2010 05:09:56 +0000","from_user":"L0k1sama","metada
Tue, 11 May 2010 05:09:19 +0000","from_user":"PureEgo03","metac
Tue, 11 May 2010 05:07:04 +0000","from_user":"natesing0812","me
Tue, 11 May 2010 05:05:10 +0000","from_user":"eruditechick","me
Tue, 11 May 2010 04:56:40 +0000","from_user":"BobbiPerfection",
Tue, 11 May 2010 04:49:39 +0000","from_user":"BB_ToCuTe07","met
Tue, 11 May 2010 04:47:40 +0000","from_user":"therealdawggy","m
Tue, 11 May 2010 04:47:39 +0000","from_user":"KevCantStop","met
```

Step 9: Programming step 3: Searching Twitter with TCP/IP port 80

Http is just TCP/IP on port 80

for example:

"Open www.google.com 80"

will open a Http connection to www.google.com.

Twitter actually requires more of the Http protocol than google.

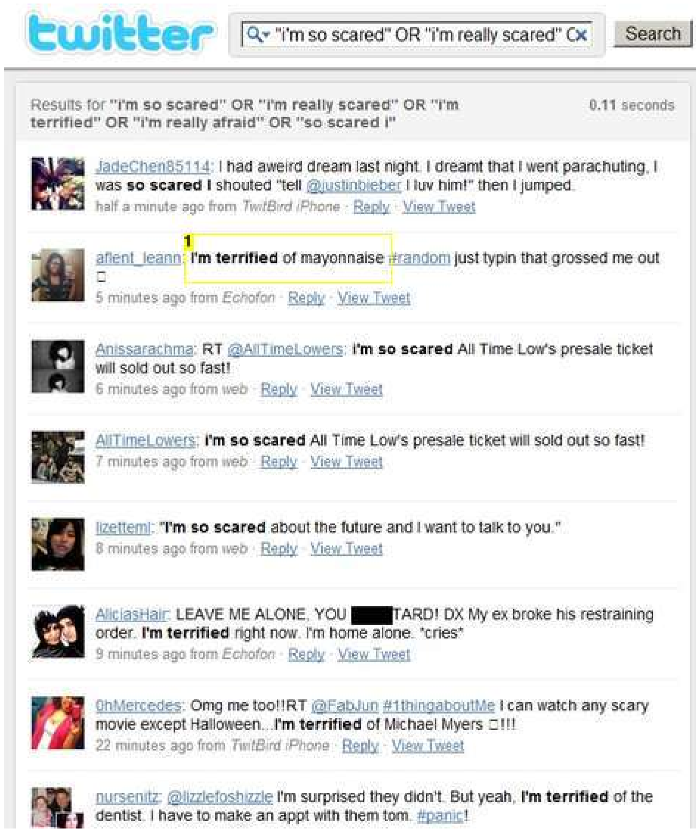
For example, the "Host" field is often required in case there's more than one
domain name mapped to the server's IP address so it can tell which
website you actually want.

Twitter also requires a final linefeed and carriage return ("\r\n")

"GET /\n"
"Host: server\r\n"
"\r\n"

I use search.json rather than search.atom to give results in non-html format, and more easily parsed. (see apiwiki.twitter.com/Twitter-API-Documentation)

```
/*
Parameters: The server to telnet into, the get command that needs to be sent, a custom HtmlParser that
is called every time a character is received. The parser is responsible for processing the HTML
that is returned.
*/
bool  WiFly::HttpWebRequest(const char* server, const char* getCommand, HtmlParser* parser)
{
m_printer->println(getCommand);
FlushRX();
FlushRX();
// Enter command mode
EnterCommandMode();
FlushRX();
// open a TCP connection, port 80 for HTTP
WriteToWiFly("open ");
WriteToWiFly(server);
WriteToWiFlyCR(" 80");
bool openOK = WaitUntilReceived(COMM_OPEN);
if (openOK == false)
{
m_printer->println("open port failed!");
delay(1000);
WriteToWiFlyCR("close");
WaitUntilReceived(COMM_CLOSE);
ExitCommandMode();
return false;
}
// eg. "GET /search.json?q=foo HTTP/1.1\r\n"
WriteToWiFlyCRLF(getCommand);
// eg. "Host: search.twitter.com\r\n"
WriteToWiFly("Host: ");
WriteToWiFlyCRLF(server);
// "\r\n"
WriteToWiFlyCRLF("");
// now wait for the response
int timeOut = 0;
bool ok = false;
while(timeOut < 5000)// timeout after 5 seconds
{
if((ReadCharFromWiFly(LSR) & 0x01))
{
char incoming_data = ReadCharFromWiFly(RHR);
m_printer->print(incoming_data,BYTE);
bool done = parser->Parse(incoming_data);
if (done)
{
ok = true;
break;
}
timeOut = 0; //reset the timeout
}
else
{
delay(1);
timeOut++;
}
}
FlushRX();
// disconnect TCP connection.
WriteToWiFlyCR("close");
WaitUntilReceived(COMM_CLOSE);
ExitCommandMode();
return ok;
}
```



"i'm so scared" OR "i'm really scared" OR "i'm terrified" OR "i'm really afraid" OR "so scared i"

Results for "i'm so scared" OR "i'm really scared" OR "i'm terrified" OR "i'm really afraid" OR "so scared i" 0.11 seconds

JadeChen85114: I had aweird dream last night. I dreamt that I went parachuting, I was **so scared I** shouted "tell @justinbieber I luv him!" then I jumped.
half a minute ago from TwitBird iPhone · Reply · View Tweet

aflent_leann: **I'm terrified** of mayonnaise #random just typin that grossed me out □
5 minutes ago from Echofon · Reply · View Tweet

Anissarachma: RT @AllTimeLowers: **i'm so scared** All Time Low's presale ticket will sold out so fast!
6 minutes ago from web · Reply · View Tweet

AllTimeLowers: **i'm so scared** All Time Low's presale ticket will sold out so fast!
7 minutes ago from web · Reply · View Tweet

lizetteml: "**I'm so scared** about the future and I want to talk to you."
8 minutes ago from web · Reply · View Tweet

AliciasHair: LEAVE ME ALONE, YOU [redacted]TARD! DX My ex broke his restraining order. **I'm terrified** right now. I'm home alone. *cries*
9 minutes ago from Echofon · Reply · View Tweet

0hMercedes: Omg me too!!RT @FabJun #1thingaboutMe I can watch any scary movie except Halloween... **I'm terrified** of Michael Myers □!!!
22 minutes ago from TwitBird iPhone · Reply · View Tweet

nursenitz: @lizzlefoshizzle I'm surprised they didn't. But yeah, **I'm terrified** of the dentist. I have to make an appt with them tom. #panic!

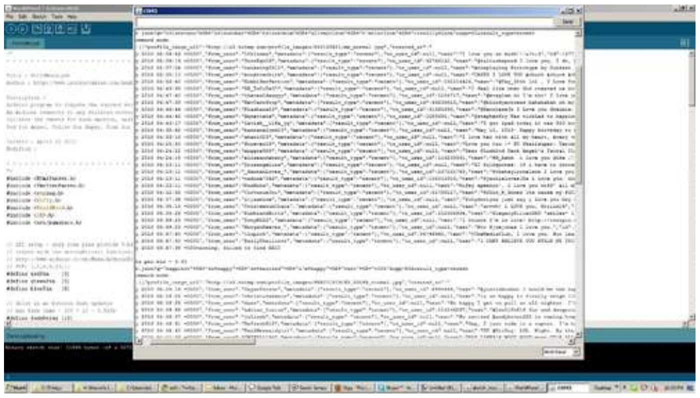


Image Notes
1. perfect

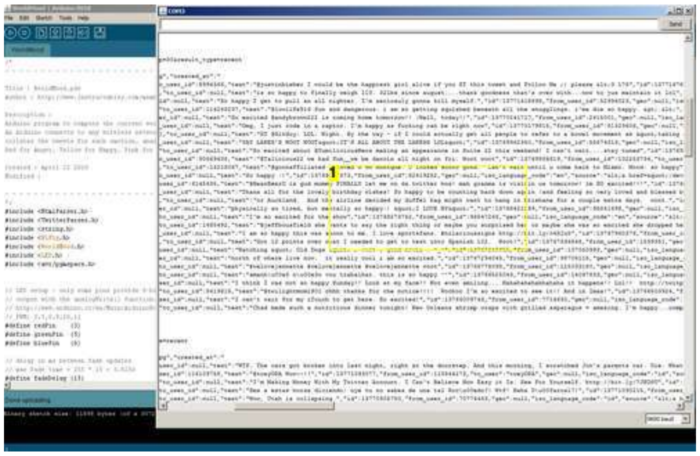


Image Notes
1. too many tweets...

Step 10: Programming step 4: RGB LED

A simple library for setting the colour of an RGB LED. The library will fade between the colours as the world mood changes, and will flash if it is a significant change in mood.

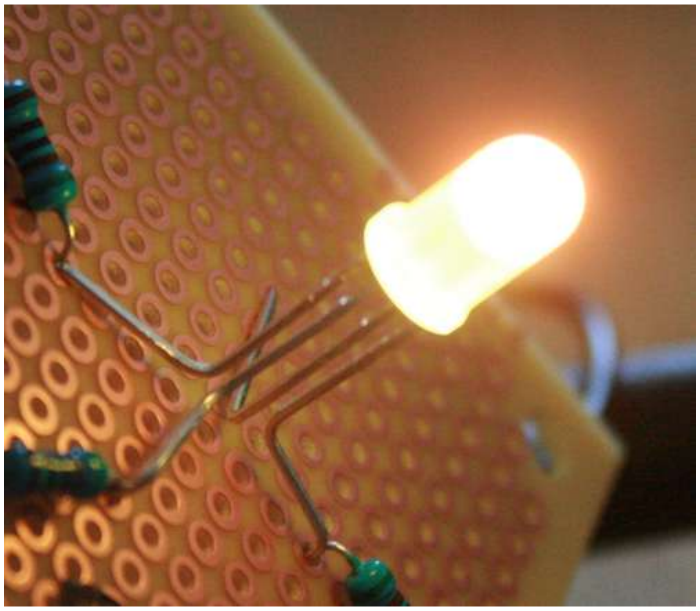
*** update ***

If you find the colours look wrong, try removing the "255 -" from the analogWrite calls.
Thanks to shobley for finding this.
More info at http://www.stephenhobley.com/blog/2010/06/11/arduino-world-mood-light-using-twitter-and-wishield/

*** end update ***

```
/*
The led is initially set to be currentColorID and over time will fade
to desiredColorID with a time delay, fadeDelay, measured in ms, between
```

```
each step. No effort is made to scale the step size for each rgb
channel so each may not complete at the same time.
*/
void  LED::FadeTo(int desiredColorID)
{
// check for valid colorID
if (desiredColorID >= NUM_COLORS ||
desiredColorID < 0)
{
//logger.log("invalid Color id")
return;
}
// get a local copy of the colors
Color currentColor;
currentColor.r = Colors[m_currentColorID].r;
currentColor.g = Colors[m_currentColorID].g;
currentColor.b = Colors[m_currentColorID].b;
Color desiredColor;
desiredColor.r = Colors[desiredColorID].r;
desiredColor.g = Colors[desiredColorID].g;
desiredColor.b = Colors[desiredColorID].b;
bool done = false;
while (!done)
{
// move each of r,g,b a step closer to the desiredColor value
if (currentColor.r < desiredColor.r)
{
currentColor.r++;
}
else if (currentColor.r > desiredColor.r)
{
currentColor.r--;
}
if (currentColor.g < desiredColor.g)
{
currentColor.g++;
}
else if (currentColor.g > desiredColor.g)
{
currentColor.g--;
}
if (currentColor.b < desiredColor.b)
{
currentColor.b++;
}
else if (currentColor.b > desiredColor.b)
{
currentColor.b--;
}
// write the new rgb values to the correct pins
analogWrite(m_redPin, 255 - currentColor.r);
analogWrite(m_greenPin, 255 - currentColor.g);
analogWrite(m_bluePin, 255 - currentColor.b);
// hold at this color for this many ms
delay(m_fadeDelay);
// done when we have reach desiredColor
done = (currentColor.r == desiredColor.r &&
currentColor.g == desiredColor.g &&
currentColor.b == desiredColor.b);
} // while (!done)
m_currentColorID = desiredColorID;
}
```



Step 11: Programming 5: Computing the World Mood

The mood light should be responsive enough to reflect what has just happened in the world, but it must not be so overly sensitive as to be susceptible to noise, and also not be too sluggish to be late in informing you of a big world event.

The important thing is to carefully normalize and smooth the data, and to adjust the thresholds to give the right level of responsiveness and alarm. (i.e. it should flash when a headline news story
happens and not when a TV show starts, GMT)

Emotion, mood, and temperament

Firstly, the "world's emotion" is calculated by searching twitter for tweets with each of the 7 mood types (love, joy, surprise, anger, fear, envy, sad) .

A measure of "tweets per minute" is used to calculate the current emotion. A higher number of tweets per minute suggests more people are currently feeling that emotion.

Emotions are volatile, so these short-lived emotional states are smoothed over time by using a "fast exponential moving average"
(see en.wikipedia.org/wiki/Moving_average#Exponential_moving_average)

This gives us ratios for the different moods.

Each mood ratio is then compared to a base line, a "slow exponential moving average", that I call the "world temperament".

The mood that has deviated furthest from its baseline temperament value is considered to be the current world mood.

The deviation is measured as a percentage, so, for example, if fear changes from accounting for 5% of tweets to 10% then this is more significant than joy changing from 40% to 45% (They are both a +5% in additive terms, but fear increased by 100% in multiplicative terms.)

Finally, the world temperament values are tweaked slightly in light of this new result. This gives the system a self adjusting property so that the world temperament can very slowly change over time.

These values in WorldMood.pde are used to adjust how sensitive the system is to information.

- Do you want it to pick up when people are happy about a sport result or scared about the weather?
- Or would you prefer to only track big events like natural disasters or terrorist attacks?

adjust accordingly...

```
#define emotionSmoothingFactor (0.1f)
#define moodSmoothingFactor (0.05f)
#define moderateMoodThreshold (2.0f)
#define extremeMoodThreshold (4.0f)

MOOD_TYPE  WorldMood::ComputeCurrentMood()
{
// find the current ratios
float sum = 0;
for (int i = 0; i < NUM_MOOD_TYPES; i++)
{
sum += m_worldMoodCounts[i];
}
if (sum < 1e-4f)
{
#ifdef  DEBUG
m_printer->print("unexpected total m_worldMoodCounts");
#endif  // ifdef DEBUG
return m_worldMood;
}
for (int i = 0; i < NUM_MOOD_TYPES; i++)
{
m_worldMoodRatios[i] = m_worldMoodCounts[i] / sum;
}
// find the ratio that has increased by the most, as a proportion of its moving average.
// So that, for example, an increase from 5% to 10% is more significant than an increase from 50% to 55%.
float maxIncrease = -1.0f;
for (int i = 0; i < NUM_MOOD_TYPES; i++)
{
float difference = m_worldMoodRatios[i] - m_worldTemperamentRatios[i];
if (m_worldTemperamentRatios[i] < 1e-4f)
{
#ifdef  DEBUG
m_printer->print("unexpected m_worldTemperamentRatios");
#endif  // ifdef DEBUG
continue;
}
difference /= m_worldTemperamentRatios[i];
if (difference > maxIncrease)
{
maxIncrease = difference;
m_worldMood = (MOOD_TYPE)i; // this is now the most dominant mood of the world!
}
}
// update the world temperament, as an exponential moving average of the mood.
// this allows the baseline ratios, i.e. world temperament, to change slowly over time.
// this means, in affect, that the 2nd derivative of the world mood wrt time is part of the current mood calcuation.
// and so, after a major anger-inducing event, we can see when people start to become less angry.
sum = 0;
for (int i = 0; i < NUM_MOOD_TYPES; i++)
{
if (m_worldTemperamentRatios[i] <= 0)
{
#ifdef  DEBUG
m_printer->print("m_worldTemperamentRatios should be initialised at construction");
#endif  // #ifdef DEBUG
m_worldTemperamentRatios[i] = m_worldMoodRatios[i];
}
else
{
const float a = m_moodSmoothingFactor;
m_worldTemperamentRatios[i] = (m_worldTemperamentRatios[i] * (1.0f - a)) + (m_worldMoodRatios[i] * a);
}
sum += m_worldTemperamentRatios[i];
}
if (sum < 1e-4f)
{
#ifdef  DEBUG
m_printer->print("unexpected total m_worldTemperamentRatios total");
#endif  // #ifdef DEBUG
return m_worldMood;
}
// and finally, renormalise, to keep the sum of the moving average ratios as 1.0f
for (int i = 0; i < NUM_MOOD_TYPES; i++)
{
m_worldTemperamentRatios[i] *= 1.0f / sum;
#ifdef  DEBUG
m_printer->print("temperament ratio: ");
m_printer->println(m_worldTemperamentRatios[i]);
#endif
}
#ifdef  DEBUG
// debug code - check sum is 1.
sum = 0;
for (int i = 0; i < NUM_MOOD_TYPES; i++)
{
sum += m_worldTemperamentRatios[i];
}
if (sum > 1.0f + 1e-4f || sum < 1.0f - 1e-4f)
{
m_printer->println("unexpected renormalise result");
}
#endif  // #ifdef DEBUG
return m_worldMood;
}
```

Image Notes
1. World, why so sad?

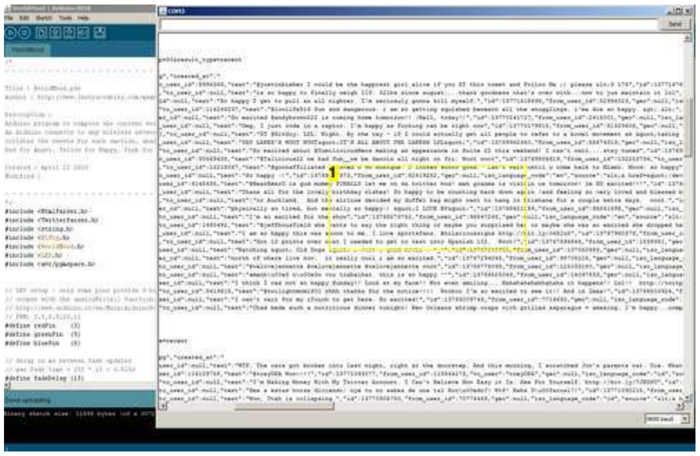
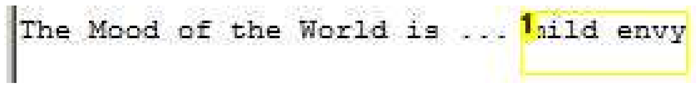


Image Notes
1. too many tweets...



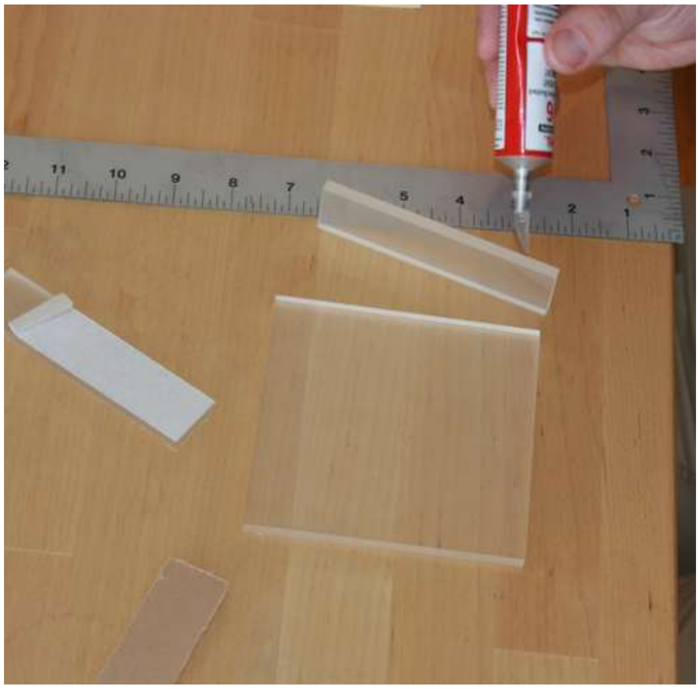
The Mood of the World is ... mild envy

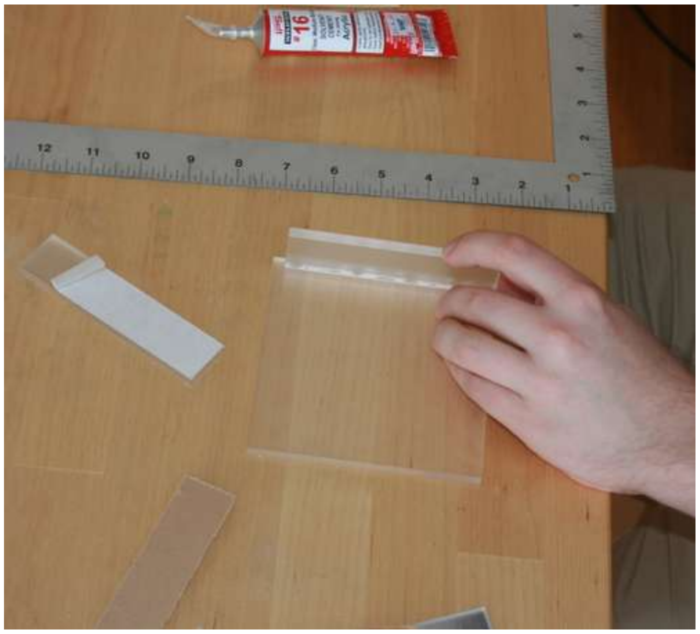
Image Notes
1. ...and so the LED turns green.

Step 12: Building the Box

Build an acrylic box ala this Instructable:

www.instructables.com/id/LED-Cube-Night-Light/





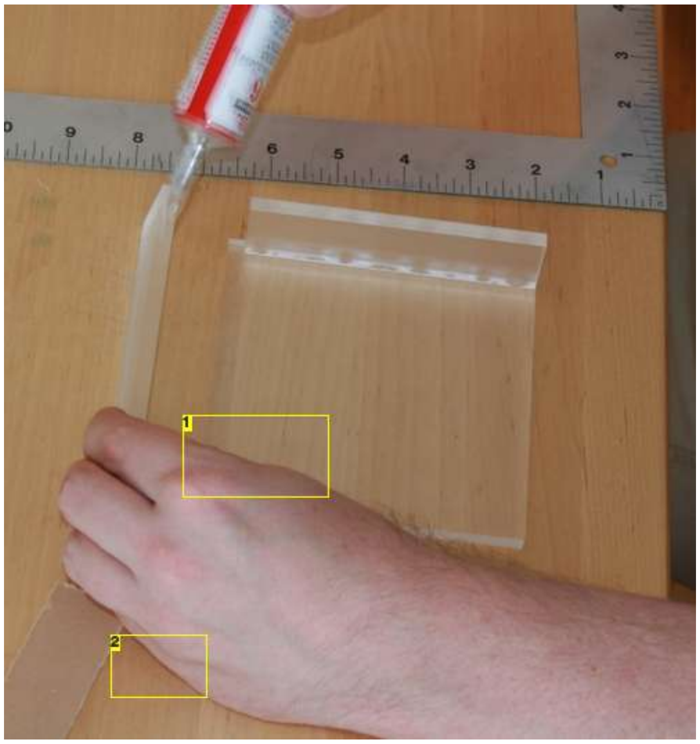


Image Notes
1. I do have a thumb
2. and pinkie

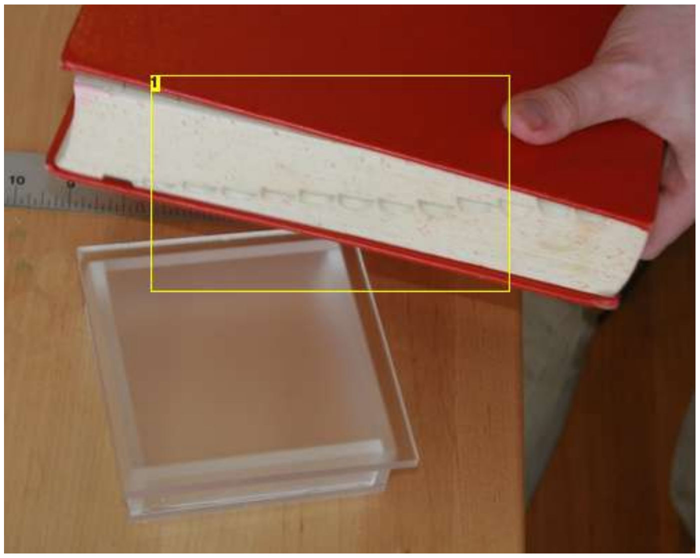
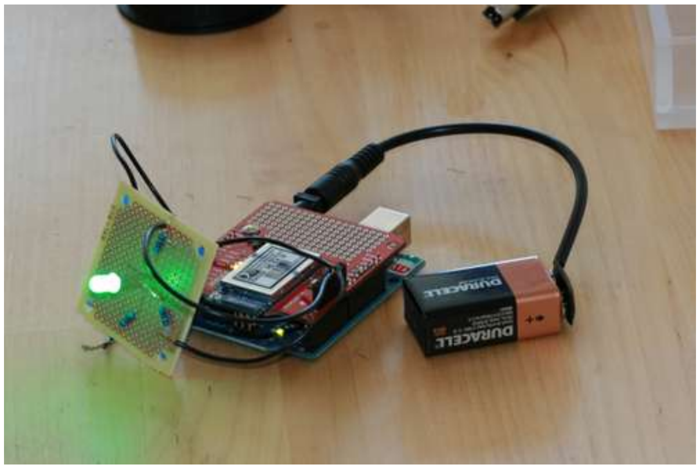


Image Notes
1. put a big book on it and leave it to dry



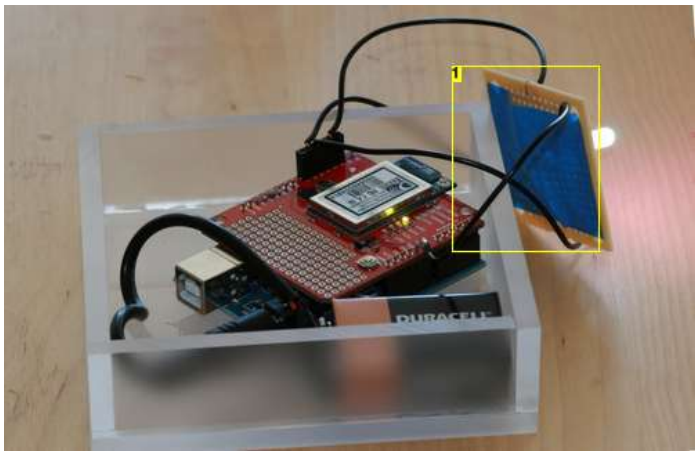
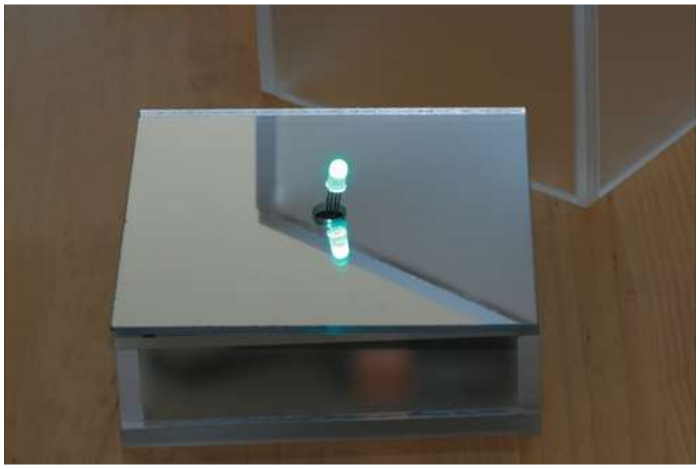


Image Notes
1. the tape helps prevent shorts



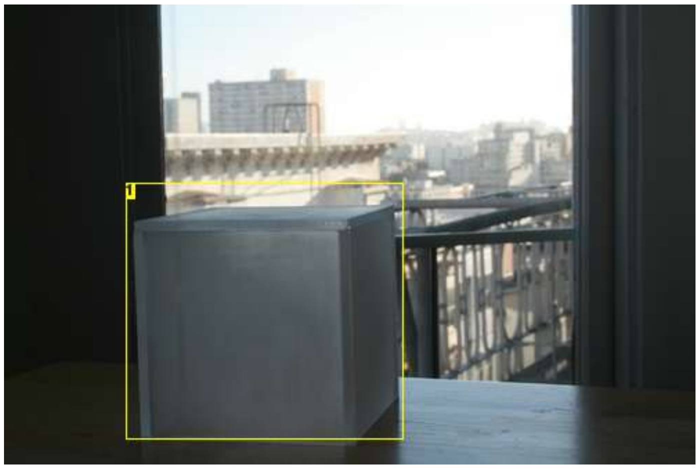


Image Notes
1. sanded to help diffuse the LED

Step 13: Enjoy!

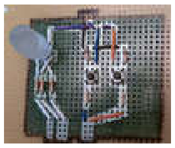
Some possible extensions include:

- Making it multilingual and not just English speaking places.
- Perhaps just associating with a keyword, for example every tweet must contain the word "Obama", then you could gauge public opinion on just that subject.
- Location specific. Perhaps you just care about your town or country. Twitter allows you to use the geocoding to do this.
- Make it tweet what the world mood is so as to complete the circle
- Ability to connect to it from a computer to see what keywords people are so emotive about.

I am very interested to hear any comments, corrections or questions. Please do contact me, if you so wish.

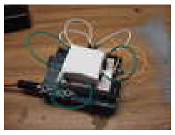
Related Instructables

Rotary Emotiphone by zvizvi

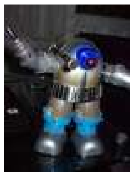

Twitter Mention Mood Light by pdxnat

Arduino mood lighting by sapc

Web-controlled Twittering Roomba by matchlighter


Arduino Led mood cube (Small) (Video Included) by 'earl

=!TRI-COLOR LED MOOD-LIGHT!=- by building_boy


USB MOOD DETECTOR BOT. (Photos) by Mr.Sanchez

How to Make an LED Ambient Mood Light: A Beginner Tutorial by elevenbytes

Flamethrowing Jack-O'-Lantern

by **randofo** on October 18, 2011

Author:randofo Randy Sarafan loves you!
I am the Technology Editor here at Instructables. I am also the author of the books 'Simple Bots,' and '62 Projects to Make with a Dead Computer'. Subscribing to me = fun and excitement!

Intro: Flamethrowing Jack-O'-Lantern

A flamethrowing jack-o'-lantern keeps the trick-or-treaters a safe distance from your house and is a fine addition to any anti-Halloween arsenal. At the first sign of any sugar-obsessed imp, simply press the trigger button and wirelessly shoot a one-second burst of flames out of the jack-o'-lantern's mouth. This plume of hellfire will make even the most bold of people think twice about approaching your door. Very few people are willing to risk life and limb for the chance of a tiny box of milk duds.

WARNING!: This pumpkin is extremely dangerous and you definitely should not make one of these. The instructions were posted here are for entertainment purposes only. I do not condone the manufacture or use of flamethrowing jack-o'-lanterns. Seriously, nothing good will come of making one of these. Don't do it.

Step 1: Go get stuff

For carving the jack-o'-lantern, you will need:

- A large pumpkin (mine was probably about 18" in diameter)
- An assortment of cutting knives. Serrated seemed to work the best.
- A marker
- Paper and pencil
- Scissors
- A spoon
- Other scraping implements. I found a chisel worked very well.

For the remote controlled flamethrower:

- Door lock actuator
- SquidBee transmitter and receiver . I had these lying around from a previous project. Any Arduino/Xbee combination should do.
- An extra ATMEGA168 or ATMEGA28 (only if using the Squidbee setup above as the receiver has no chip)
- Small can of WD-40
- 12" x 12" x 1/8" sheet of black acrylic
- SPST 5V relay
- Perfboard
- 5" x 2.5" x 2" project box.
- SPST momentary pushbutton switch
- 10K resistor
- (x2) 9V battery snap
- (x2) M-type plug adapters
- Misc. long zip ties
- 16" x 2" x 1/4" aluminum extrusion
- 3-1/2" x 1/4 bolts
- (x6) 1/4 nuts
- Tea light
- Matches

Step 2: Cut a cap

Cut around the stem of the pumpkin at an angle (with the knife slanted in towards the stem of the pumpkin)

After you are done cutting all the way around, remove the stem. This will serve as your lid later on.

Step 3: Gut it

Remove the guts from the pumpkin. To start it should be easy simple to pull them out by hand, but this is going to quickly become too difficult.

Using a metal spoon or other scraping tool (I found a chisel works best) scrape the sides of the pumpkin and remove all of the slimy innards. The inside should be reasonably smooth and clean when it is done.

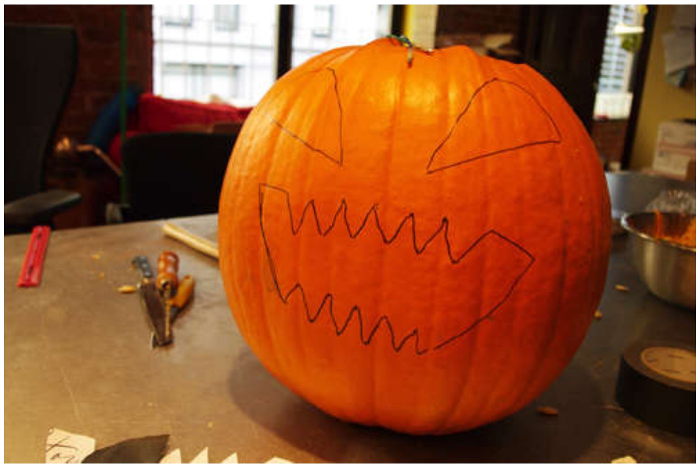
Step 4: Design a face

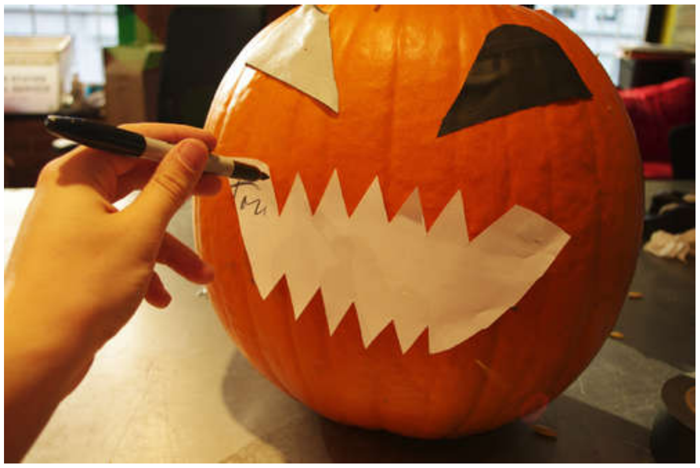
Draw a face on a piece of paper and then cut it out and tape it to the pumpkin.

One thing to keep in mind is that the mouth needs to be large and about halfway up the pumpkin or the flames aren't going to be able to shoot out.

Step 5: Trace

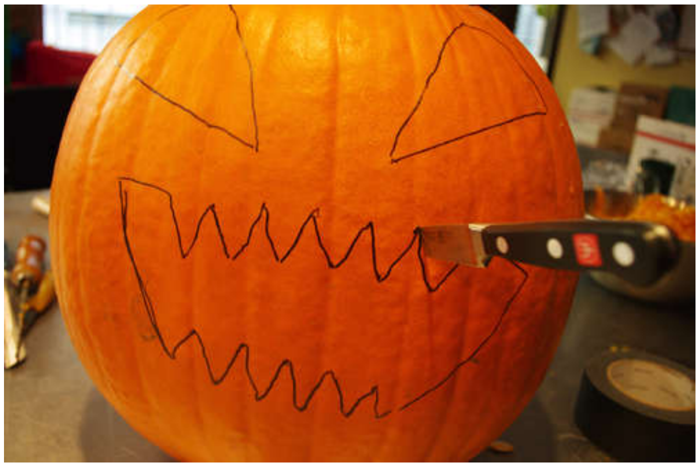
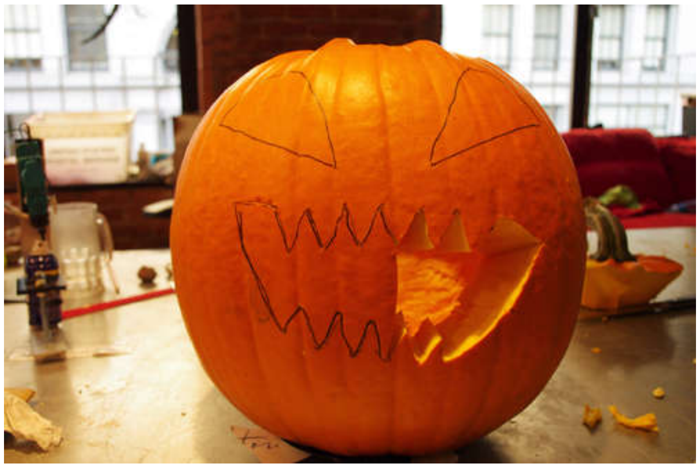
With a marker, trace the outline of the face onto the pumpkin and remove the paper.





Step 6: Cut

Cut out the pumpkin's face. For the larger and more complicated shapes like the mouth, it help to cut it out in smaller pieces instead of trying to remove one large chunk from the pumpkin.

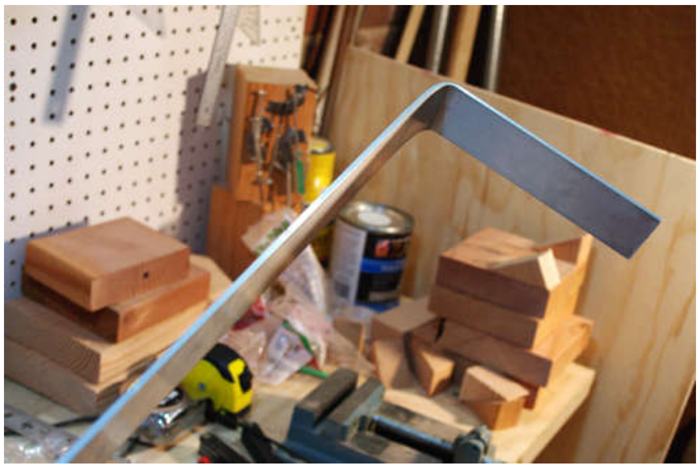



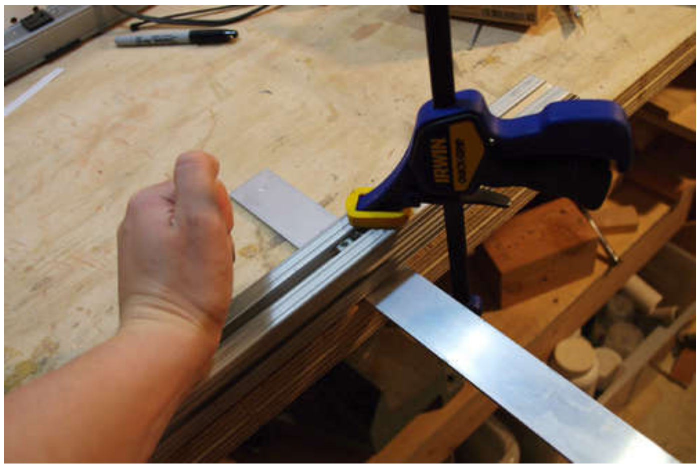
Step 7: Bend

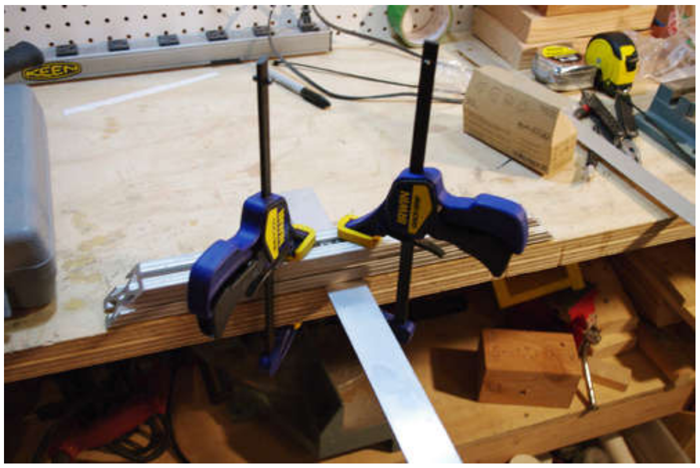
Make a mark about 6" from one of the edges of the aluminum extrusion.

Line up this mark with the edge of the workbench and clamp it between the workbench and something stiff and flat like a 2x4 or metal bar.

Grab the protruding edge firmly and push down until it is bent to 90 degrees. In doing so, you may want to push it slightly past 90 degrees as the aluminum tends to spring back a little when done.





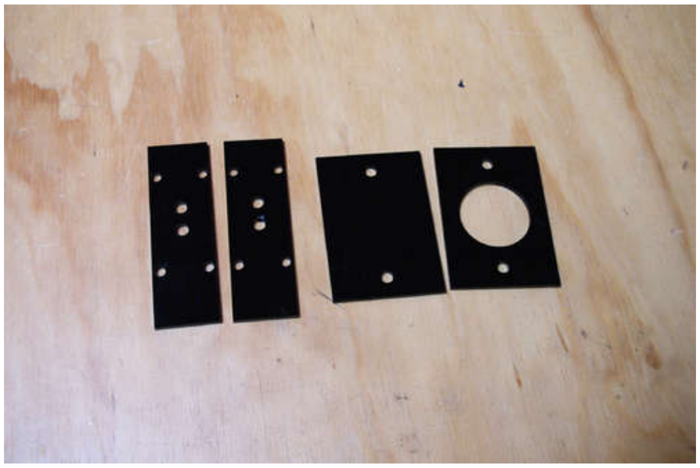


Step 8: Brackets

Download the following files for the motor mount and candle holder.

Use these files as cutting guides to cut the pieces out of 1/8" acrylic.

At times like these, having a laser cutter or using a laser cutter service comes in handy.



File Downloads

FTPCandle.eps (106 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'FTPCandle.eps']

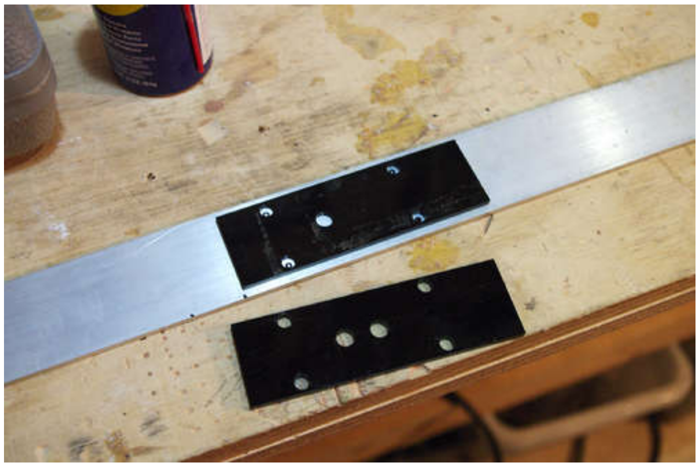
Step 9: Drill holes

Use the two mounts that you just cut out as drilling guides on the aluminum extrusion.

The motor mount should line up with the long edge of the extrusion and you should use a marker to mark all 4 corner holes.

The candle mount should be slightly backed off from the short edge. Make two marks for those holes as well.

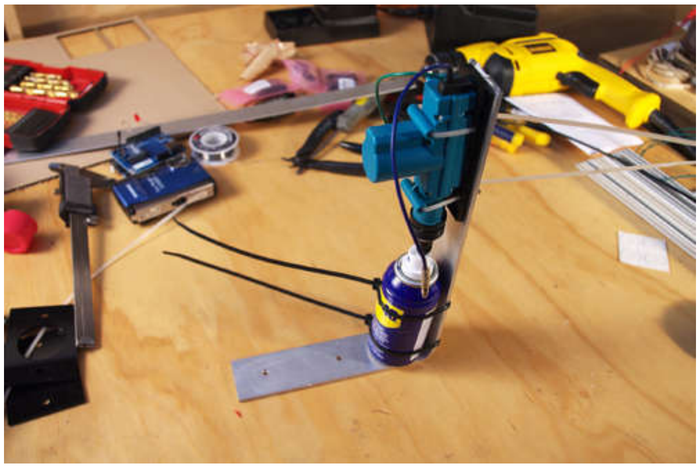
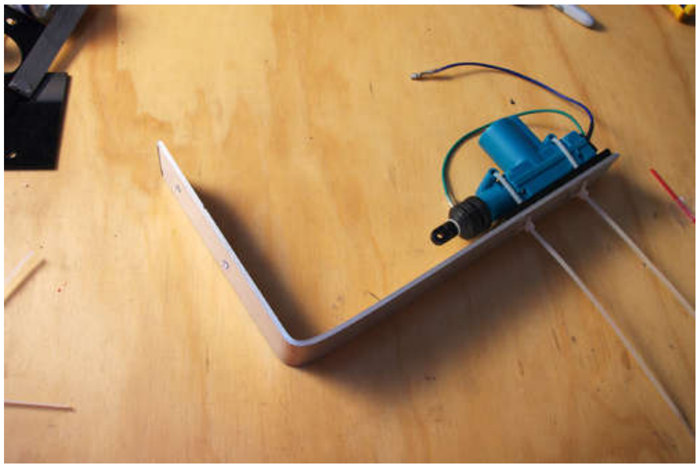
When you are done, drill 1/4" holes through the aluminum using a drill press.



Step 10: Attach things

Stack the two motor mounts and align the motor atop it. Zip tie it all to the aluminum bracket.

Below it zip tie the small WD-40 can. The actuator from the motor should be aligned and touching the top of the can, but not yet pressing down firmly onto it.

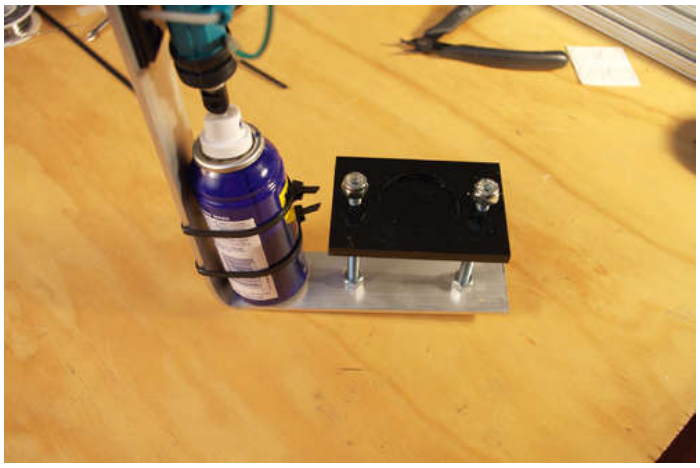
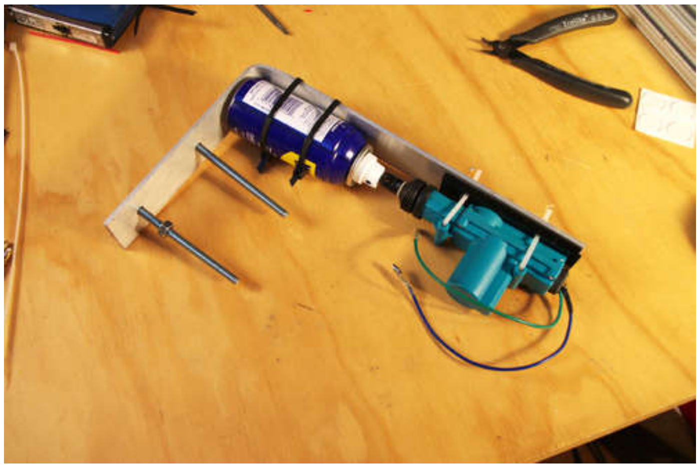
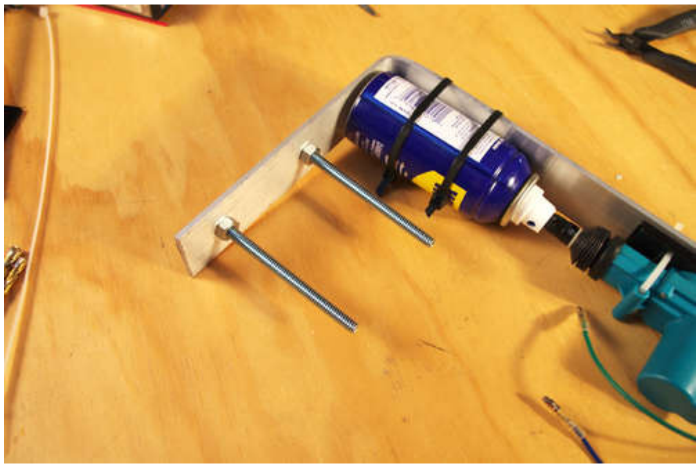
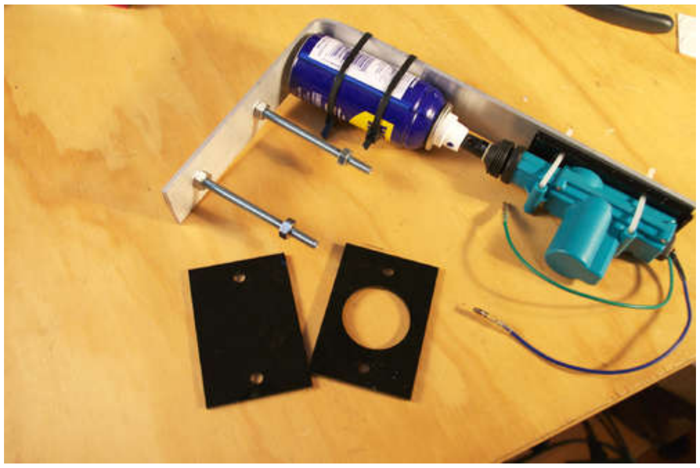
Step 11: Candle mount

Insert the two bolts upwards through the bottom of the aluminum bracket. Fasten them in place with bolts.

Thread on another bolts onto each. Twist this about 3/4" down.

Place the bottom of the candle holder (the side without the large hole) onto the bolts. Then place the top candle holder bracket.

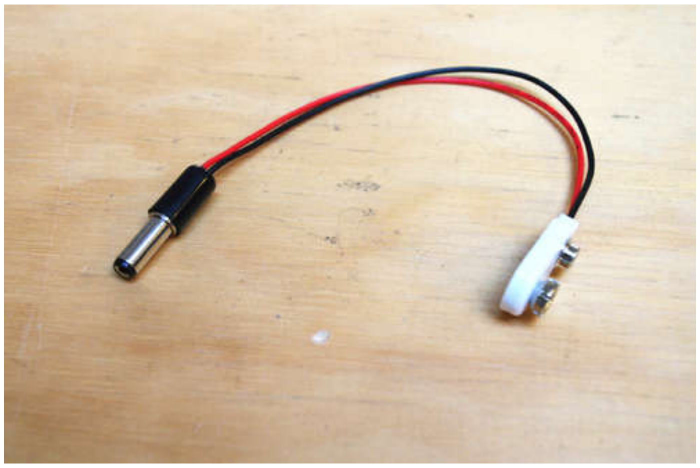
Fasten the whole thing in place by threading on another nut onto each.

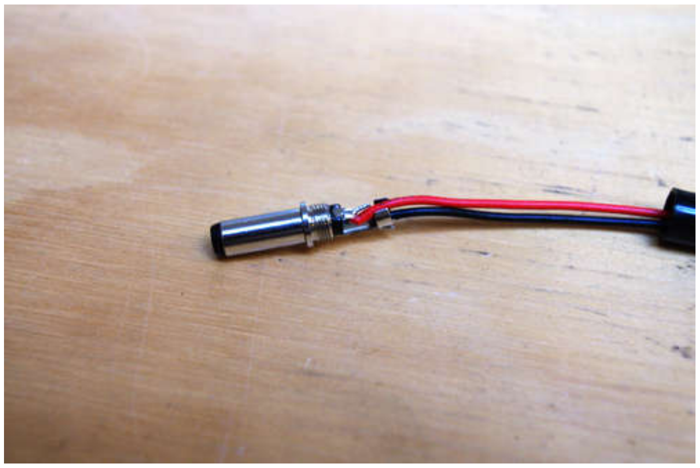





Step 12: Battery adapter

Solder the 9V battery snap to the M-type plug such that the red wire is connected to the tip and the black wire is connected to the barrel.

Don't forget to slip the plug's cover onto the wire before you solder.





Step 13: Program the Receiver

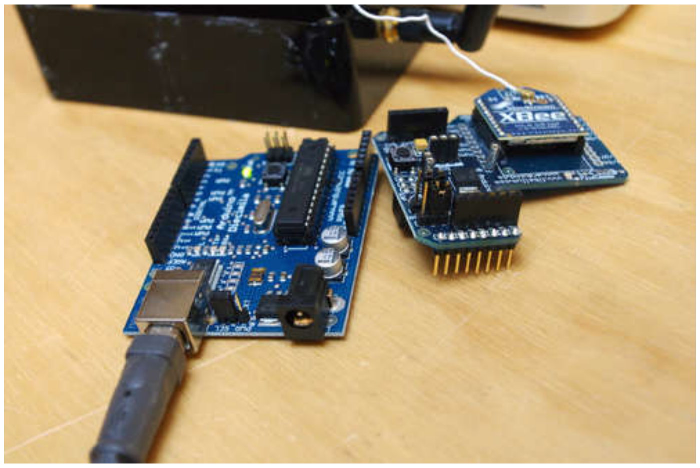
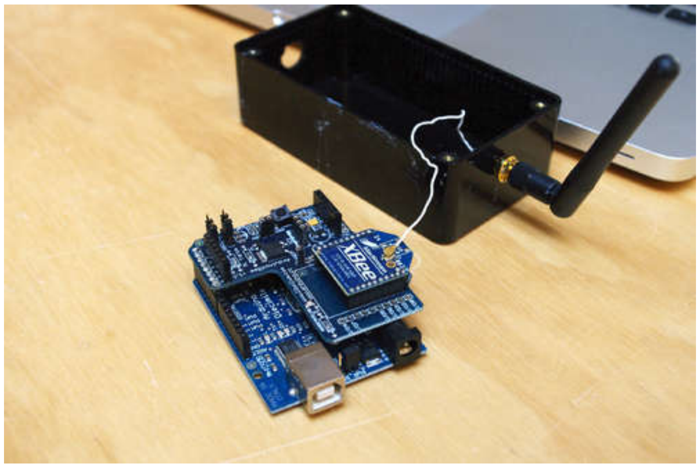
Open the SquidBee transmitter node and remove the Arduino from the XBee shield.

Change the power jumper on the Arduino to select USB power (if necessary).

Program the Arduino with the following code:

```
//Flamethrowing Jack-O'-Lantern Receiver code
//Copyleft 2011

int sentDat;

void setup() {
 Serial.begin(9600);
 pinMode(3, OUTPUT);
}

void loop() {
 if (Serial.available() > 0) {
sentDat = Serial.read();

if(sentDat == 'h'){
         //activate the pumpkin for one second and then stop
    digitalWrite(3, HIGH);
         delay(1000);
         digitalWrite(3, LOW);
}
 }
}
```

When done, disconnect the USB power, change the power selection jumper, and plug the XBee shield back in.

File Downloads

FTPumpkin.pde (392 bytes)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'FTPumpkin.pde']

Step 14: Program the transmitter

The transmitter is a little bit trickier if you are using a SquidBee setup because it is lacking an ATMEGA chip.

First unplug the XBee shield.

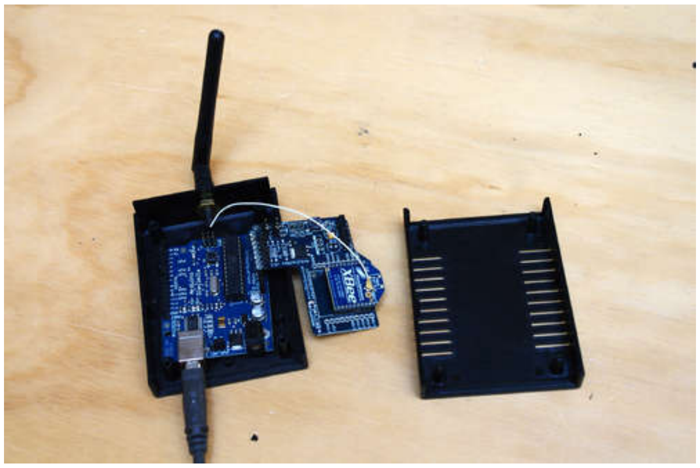
If necessary, add and bootload and the chip.

Then, like the other board, change the power selection jumper to USB, and then upload the following code:

```
/*

 Flamethrowing Jack-O'-Lantern Trigger code

 Based on Button example code
 http://www.arduino.cc/en/Tutorial/Button
 created 2005
 by DojoDave <http://www.0j0.org>
 modified 28 Oct 2010
 by Tom Igoe

The circuit:
* pushbutton attached to pin 2 from +5V
* 10K resistor attached to pin 2 from ground

This code is in the public domain.

*/

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
 // initialize serial communication:
 Serial.begin(9600);
 // initialize the LED pin as an output:
 pinMode(ledPin, OUTPUT);
 // initialize the pushbutton pin as an input:
 pinMode(buttonPin, INPUT);
}

void loop(){
 // read the state of the pushbutton value:
 buttonState = digitalRead(buttonPin);

 // check if the pushbutton is pressed.
 // if it is, the buttonState is HIGH:
 if (buttonState == HIGH) {
   // turn LED on:
   digitalWrite(ledPin, HIGH);
   //transmit a High command to the pumpkin and delay a second so that it does not recieve more than one command
   //per button press
   Serial.println('h');
   delay(1000);
 }
 else {
   // turn LED off:
   digitalWrite(ledPin, LOW);
 }
}
```

When you are done, unplug the USB, and reconnect the XBee shield. You will also need to swamp back the power jumpers on the Arduino.

Lastly, change both of the TX/RX jumpers on the XBee shield from USB to XBee.



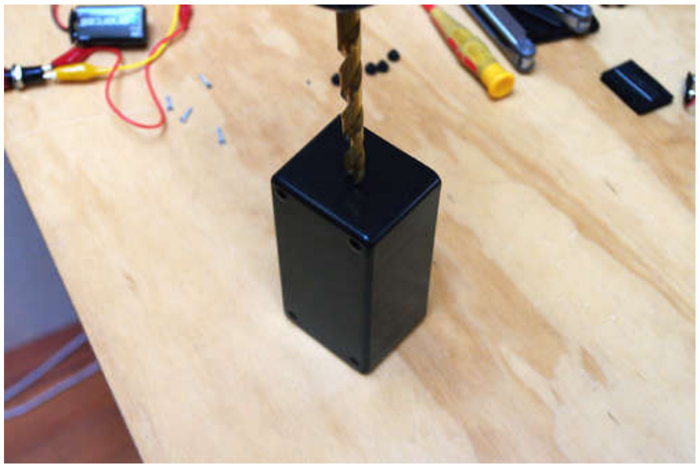
File Downloads

Pumpkin_Trigger.pde (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Pumpkin_Trigger.pde']

Step 15: Switch

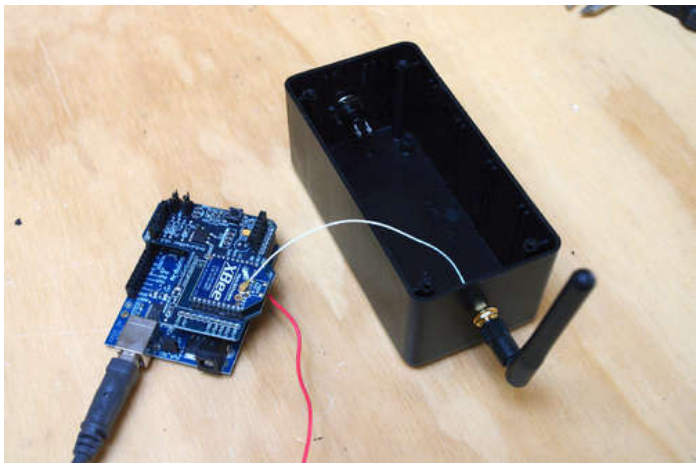
Drill a 3/8" hole (or whatever is appropriate for your switch) in the side of your project enclosure.

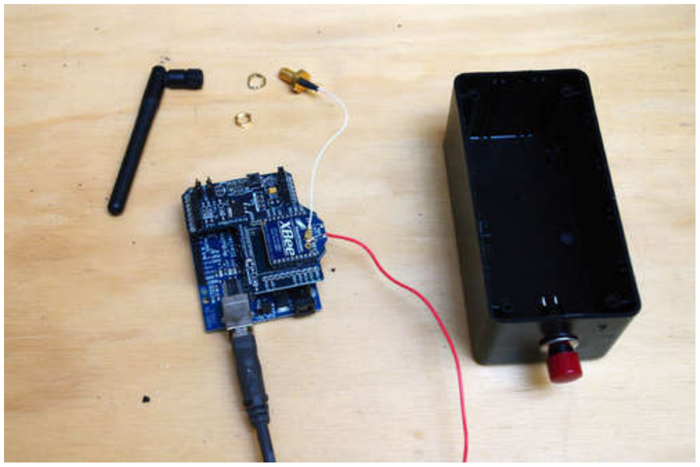
Install the pushbutton switch.

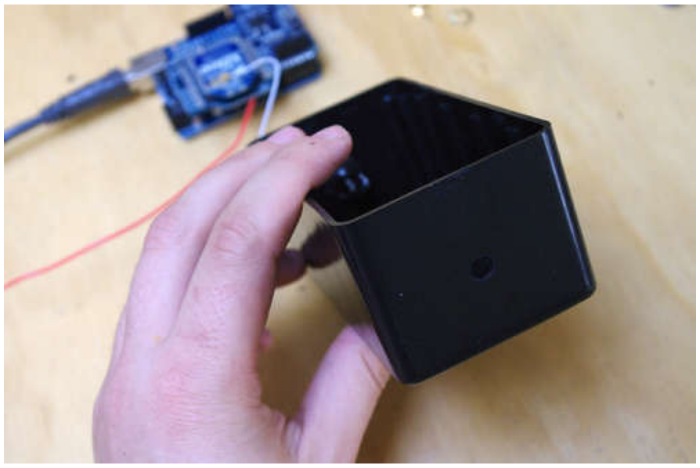
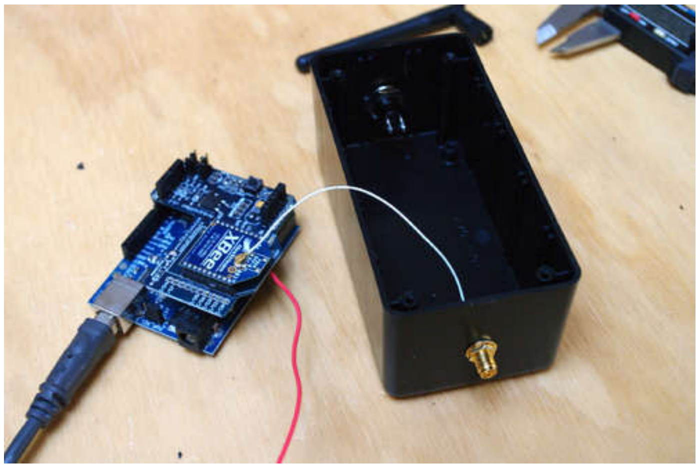


Step 16: Antenna

Install the antenna into the side of the enclosure opposite the switch. Be careful not to break the wire connecting the antenna to the XBee.





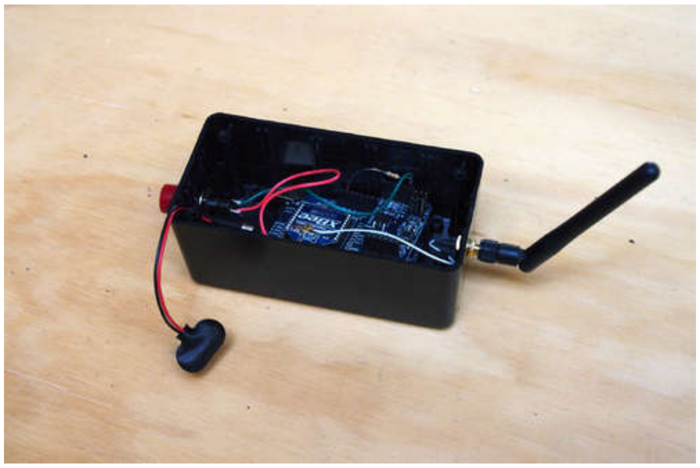
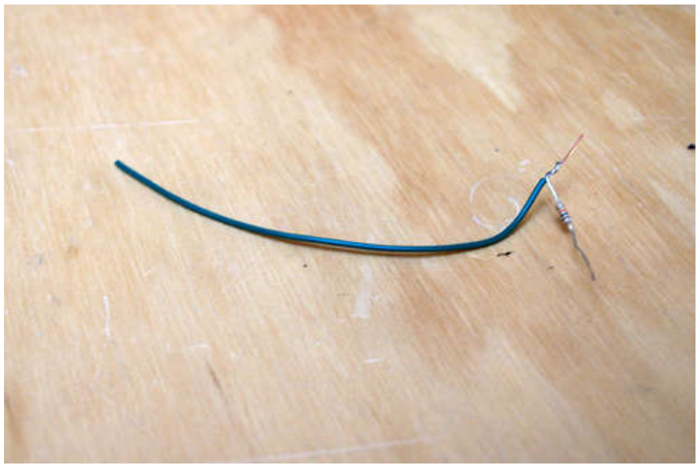
Step 17: Wire the transmitter

Solder a wire to one leg of a 10K resistor. Solder the opposite end of this wire to one of the switch terminals.

Plug in the side of the resistor with the wire soldered to it into pin 2 of the Arduino. Connect the other end of the resistor to ground.

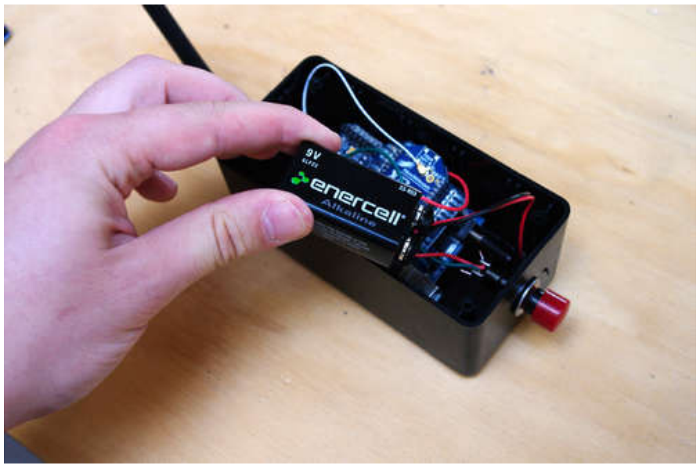
Solder a wire to the other terminal on the switch and connect this wire with 5V on the Arduino board.

Lastly, plug your 9V battery connector into the power socket on the Arduino board.

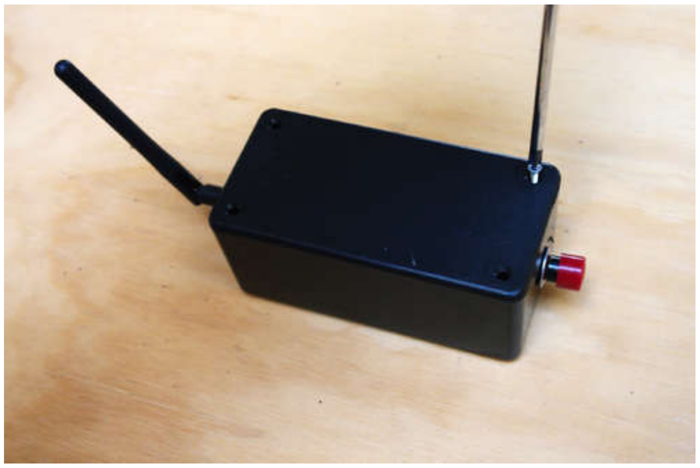
Step 18: Power

Plug in a 9V battery to power up the transmitter.



Step 19: Case closed

Fasten shut the transmitter's case.



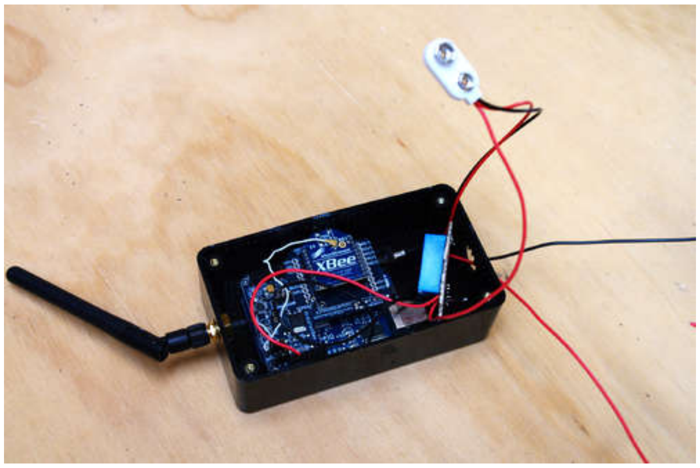
Step 20: Wire the reciever

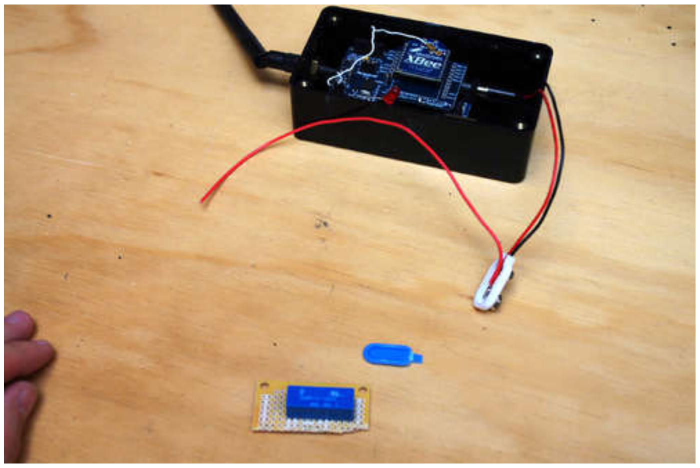
Affix the relay to a small piece of perfboard.

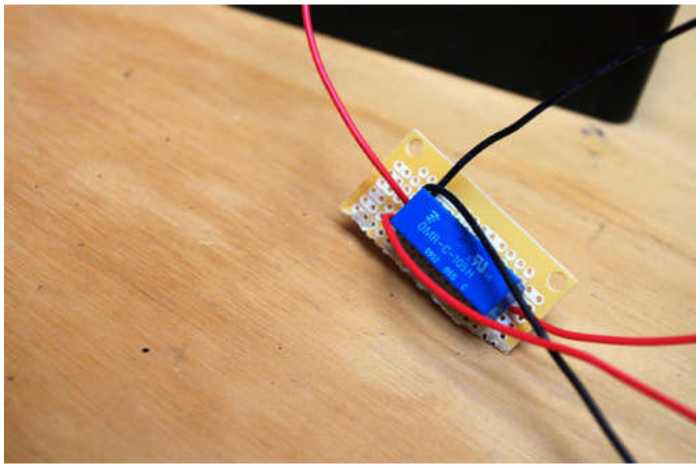
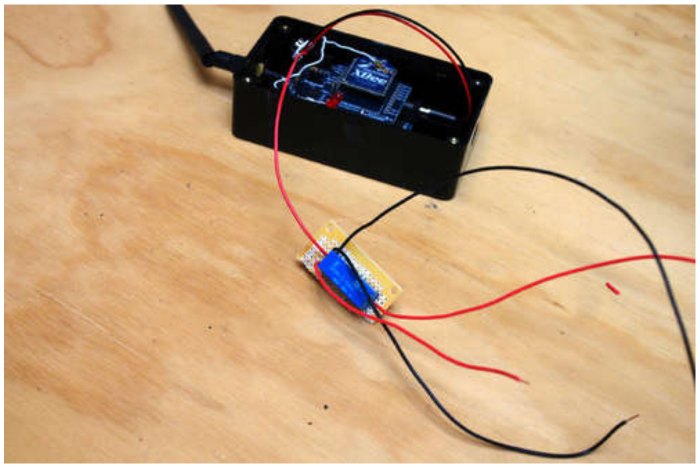
Connect one of the relay's coils to ground on the Arduino board and the other to pin 3.

Attach 9V to one of the relay's load pins and a long red wire to the other. To get easy access to the 9V power source, I broke off the top of the 9V battery snap and soldered a wire directly to the +9V battery connector tab (notice the extra red wire coming from the 9V battery snap).

Attach an extra long black wire to ground.





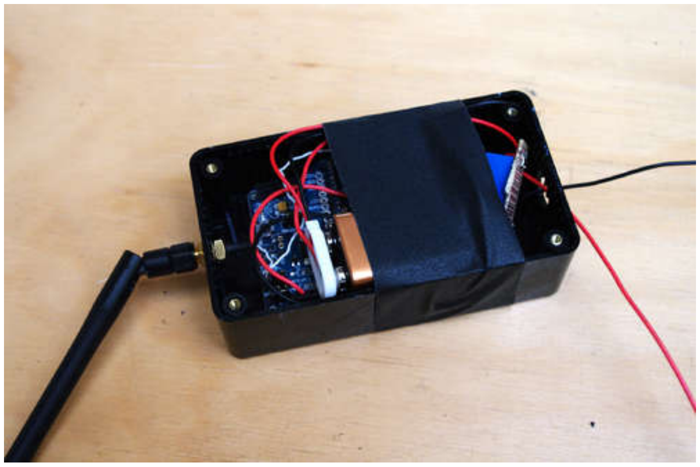



Step 21: Put it together

I lost the case for my SquidBee transmitter node (the pumpkin receiver) a long time ago. I find that a piece of black gaffers tape typically gets the job done.

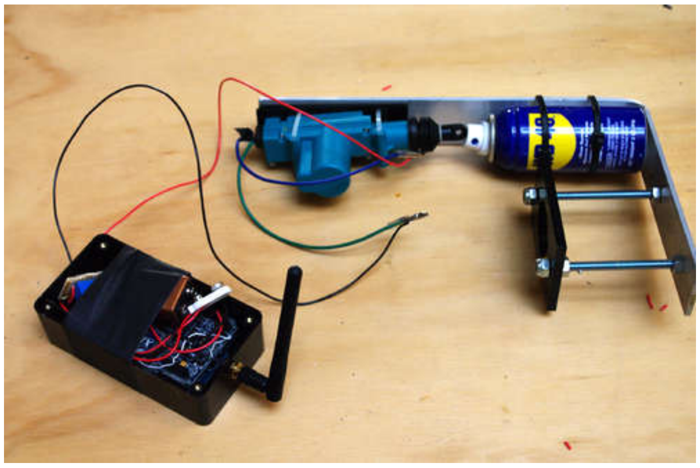
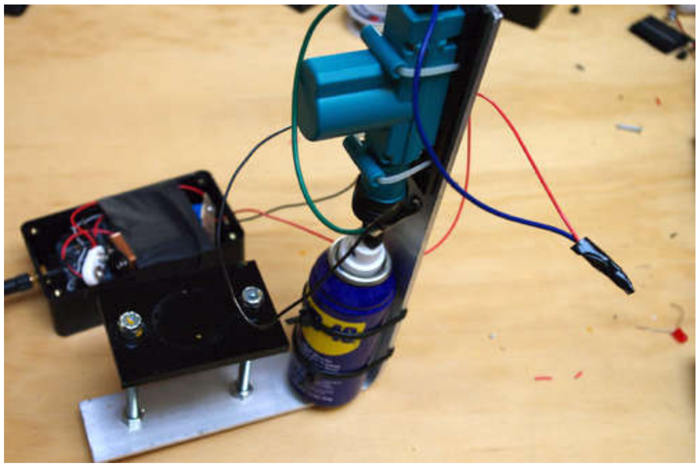
I plugged in the 9V battery and passed the red and black wires through the hole in the side of the case neat-like.

Then, I slapped a piece of black tape on top and called it a day. This will be inside the pumpkin, so aesthetics don't matter quite as much.



Step 22: Wire the motor

Wire the motor to the relay wires such that when the relay closes, the motor's actuator pushes down. In this case, red went to blue and black to green. It may be different for another motor.

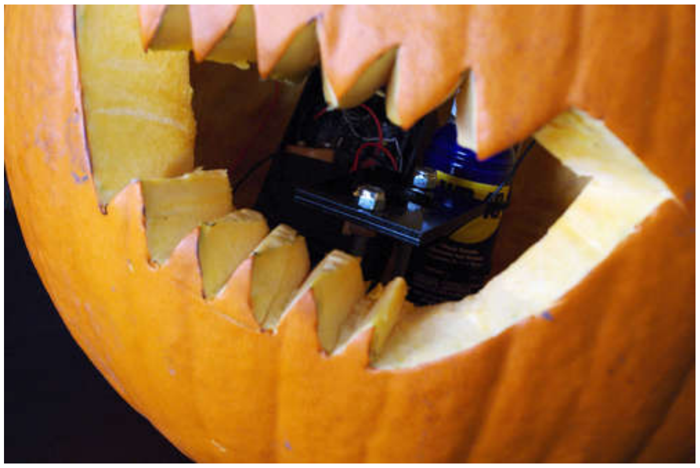
Step 23: Put it in the pumpkin

Place the whole contraption inside of the pumpkin.

Make sure that the battery is plugged in.

Also, make sure that the lid fits. If the lid does not lay flat, trim it appropriately to work.

Finally, it is a good idea to test to see if the the WD-40 sprays when the button on the transmitter is pressed down. It is easier and exponentially safer to debug this when there is no flame present.



Step 24: Candle

Once it is certain that everything is working as it should, it is time to add fire.

First off, find the transmitter. Make sure no one or nothing is pressing down on the button and it is somewhere safe.

Light a tea light and place it in the candle holder.

Step 25: Fire!

Take a number of steps way back from the pumpkin and press the trigger on the transmitter. If all is well with the world, the jack-o'-lantern will blast a burst of hellfire out of its mouth.

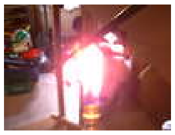
Shock and awe all innocent bystanders. This is the stuff nightmares are made of.

All of that said... **SERIOUSLY, DON'T MAKE THIS.**

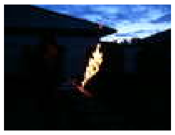
Related Instructables

How to Make a Flame Thrower Pen (video) by CyberTech2000

Great little Flame "Thrower" from Matches by JamesHD1997

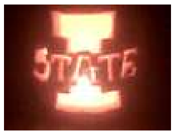


Easy Flame Thrower ! by Opolopolis



Wrist mounted Flame Thrower (Photos) by thund3rcock

LEGO team Fortress 2 Pyro's Flamethrower by BioGuns



How to Carve a Pumpkin by Xjac0bmichaelX

K-TON Palm Flamethrower (Photos) by Valencio

Knex FlameThrower (Photos) by SLDxRaPiiDZz

Make a 24X6 LED matrix

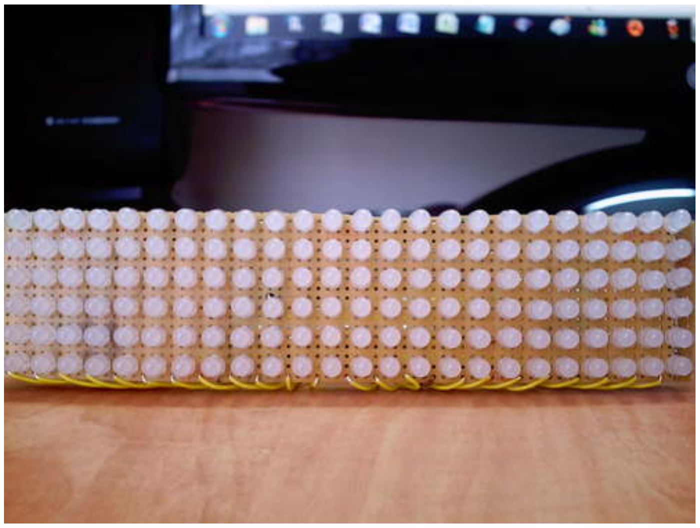
by **Syst3mX** on July 21, 2010

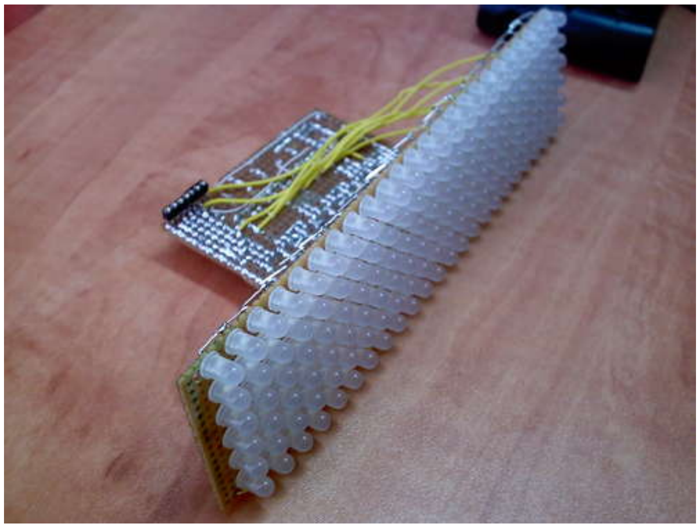
Author:Syst3mX Vadim
Electronics and LEDs what can be better ?! :D

Intro: Make a 24X6 LED matrix

After making a 8X10 matrix a lot of people asked me about expanding the matrix to some thing bigger, and some wanted to write stuff to the matrix via a PC, so one day I looked at a pile of LEDs that I had leftover from a LED cube projected and I decided to make a bigger matrix with all the things people wanted.

So what are you waiting for? Get those LEDs out and heat up your soldering iron because we are about to make a 24X6 LED matrix!





Step 1: Getting All The Right Things

So you will need the basic set of tools for this project : a soldering iron, some solder wire, a cutter, a needle nosed plier,some wire, wire striper, and some desoldering tools if you need them.

For the matrix you will:
1. 144 LEDs
2. 24 resistors(The value is determent by the type of LEDs, in my case 91 ohm)
3. 4017 decade counter
4. 6 1KOhm resistors
5. 6 2N3904 transistors
6. A long Perfboard
7. Arduino
8. 3 x 74HC595 shift register
10. some pin headers

Step 2: How it works?

The tricky behind the display is multiplexing and the idea is the same as withe the 8x10 LED matrix: It is basically a way to split information in to little peaces and send it one by one.
this way you can save a lot of pins on the Arduino and keep your program quite simple.

Now this time we have 3 shift registers which multiply the number of outputs and save lots of arduino pins.
Each shift register has 8 outputs and you only need 3 arduino pins to control almost an limited numbers of shift registers.
We also use the 4017 decade counter to scan the rows, and you can scan up to 10 rows with it because you have only 10 outputs but to control it you need only 2 pins.
The 4017 is a very useful chip and it's a good idea to know how to work with it(http://www.doctronics.co.uk/4017.htm)

Like I said the scanning is done with the 4017, by connecting one row at a time to ground and sending the right data via the shift registers to the columns.

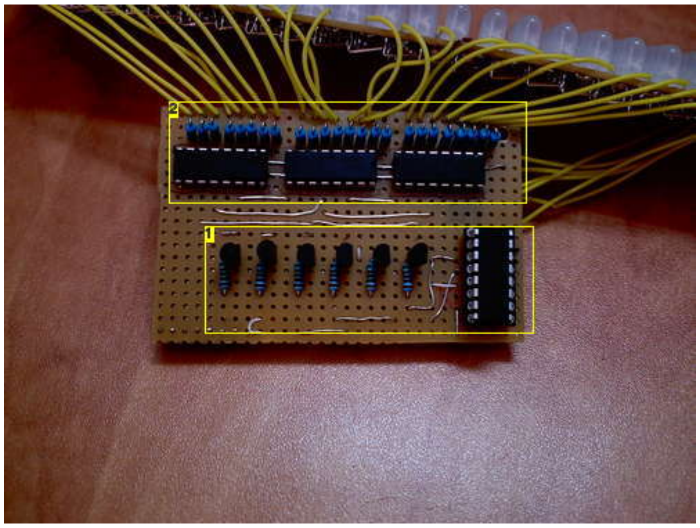


Image Notes
1. 4017 With 6 transistors for scaning the rows
2. The 3 shift registers to control the rows

Step 3: Schematics

The only thing I didn't specified in the schematics is the value of the current limiting resistors because they change from each type of LEDs, so you will need to calculate them by your self.

Now to calculate the value of the 24 resistors you can use this site :
http://led.linear1.org/1led.wiz
You should first get some specs on your LEDs, you should know their forward voltage and forward current, you can get this info from the seller. The circuit operates on 5V so your Source voltage is 5V.

Download the original file to see the schematics better.(press the "i" icon in the top left corner of the picture)

I have added a PCB layout of the control board, and i want to thanks Willard2.0 who made this layout and let me use it so thanks a lot mate!

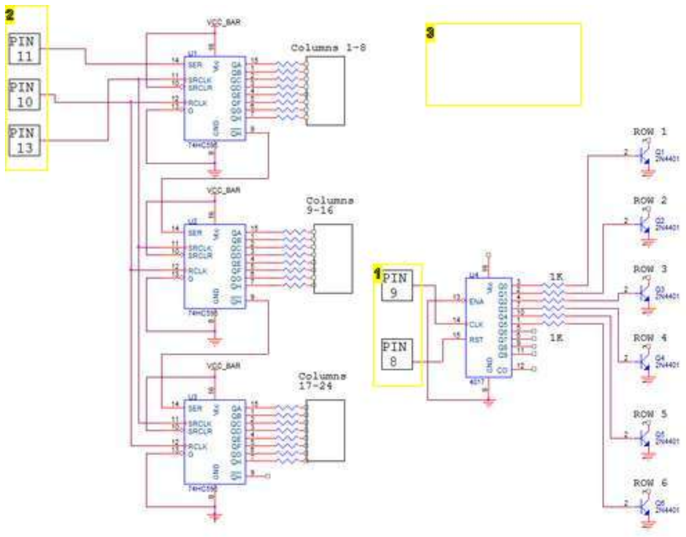


Image Notes
1. Arduino Pins
2. Arduino pins
3. Download the original file to see better.(press the "i" icon in the top left corner of the picture)

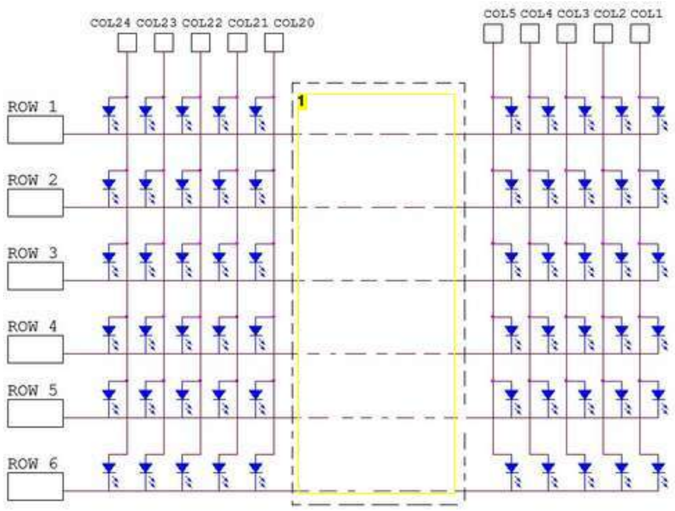


Image Notes
1. I didn't have room for the full matrix, So I hope you get the big idea

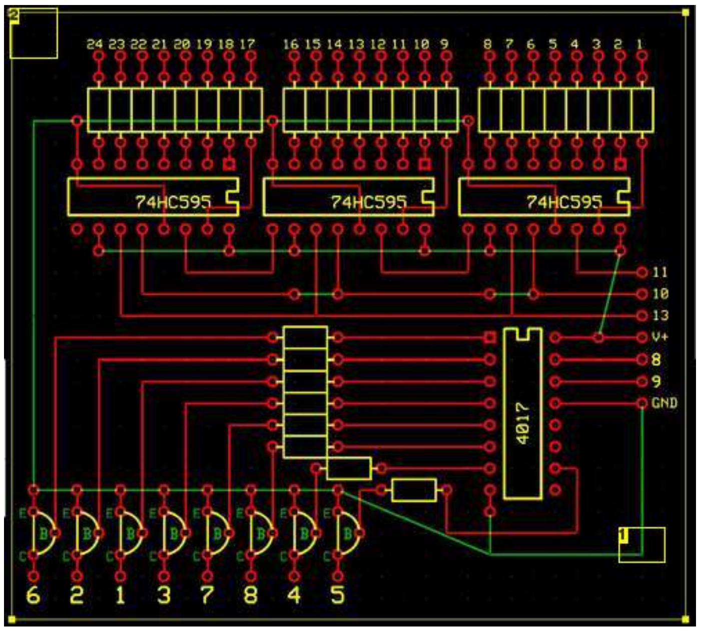
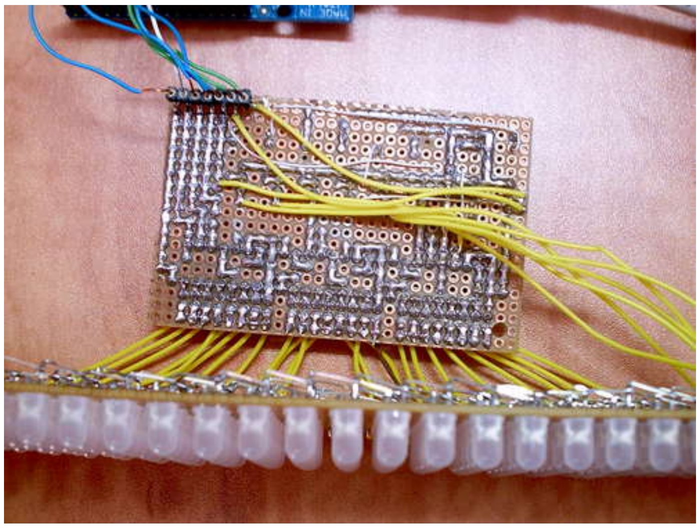


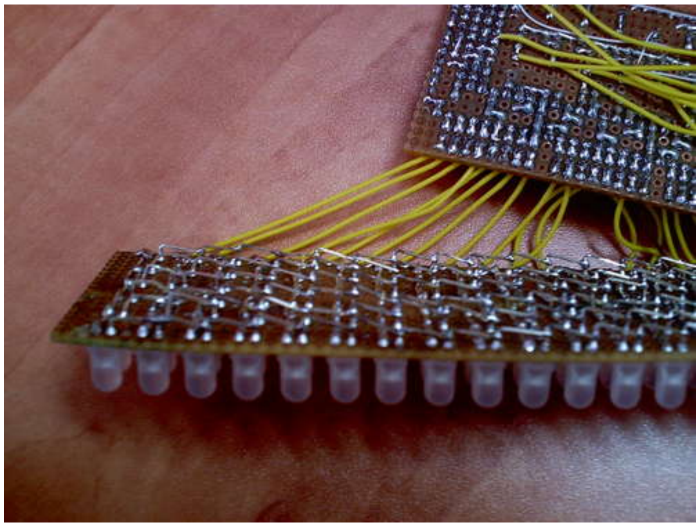
Image Notes
1. Green line are jumpers and red lines are copper traces.
2. PCB layout made by Willard2.0

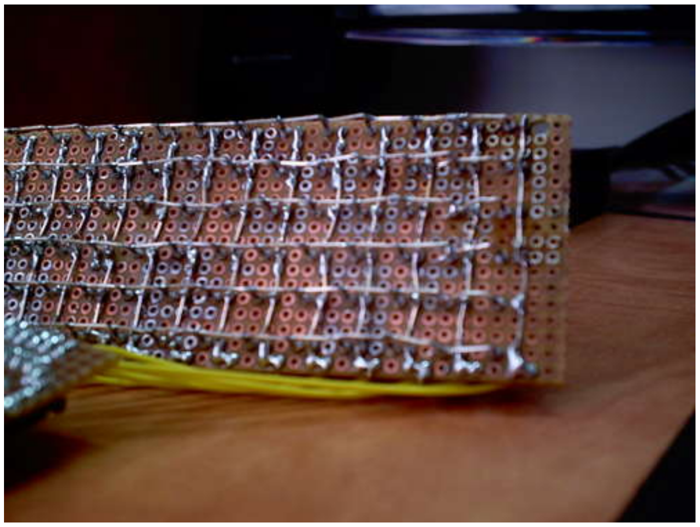


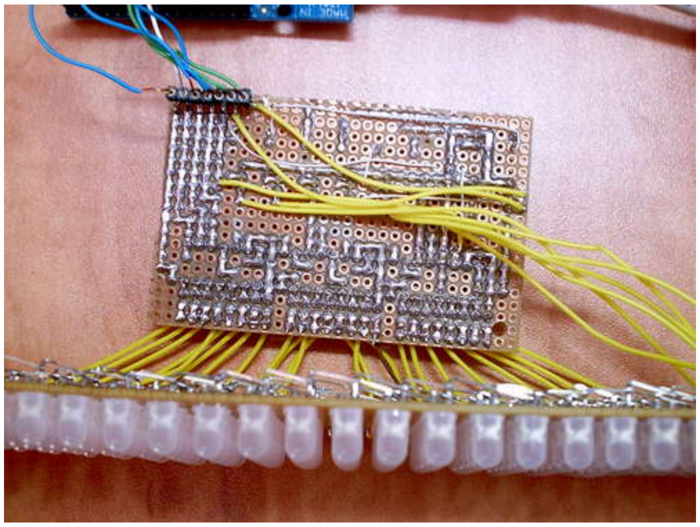
Step 4: Soldering The LEDs
Soldering 144 LEDs in a matrix formation can be a little tricky if you don't have a general idea how.
The last time I soldered a matrix I used lots of little wire jumpers which was a pain to solder, so this time I was a little more creative and came up with this way.

You need to bend the positive lead of the LED down towards the other ones and make a column, and snip off the leads you didn't use and try to make the connections as low as you can get, and you do this to all of the positive leads.
Now the negative leads are connected in a column and thats make soldering tricky because the positive rows are in the way, so you will need to make a 90 degrees bend with the negative lead and make a bridge over the positive row to the next negative lead, and so on to the next LEDs.

Now I will not explain how to solder the shift registers and all the parts because every one has hes own style and methods.



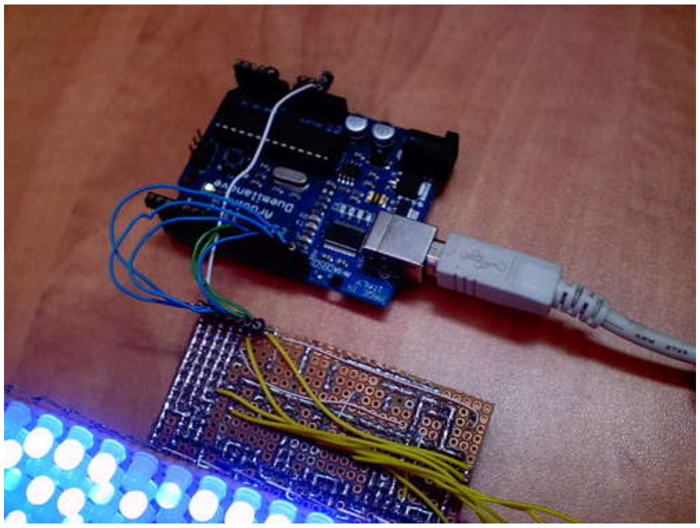




Step 5: Programming The Display
We are almost there, the only thing thats left is the program.
So far I wrote two programs for it that do pretty much the same thing.
I have added the program that gets a word or a sentence from the arduino IDE serial monitor and displays it on the matrix, my code is very basic and may be not the best in the world but it does the work, and you are free to write your own code and modify mine as you wish.

I have added an excel file so you can create your own symbols and characters.
The way it works is like so:
You create the symbol you want pixel by pixel(don't worry it's very easy) and copy the output line like so - #define {OUTPUT LINE}

I will add in the future a code for animations and a nice game of snake as soon as I have more time on my hands.



File Downloads

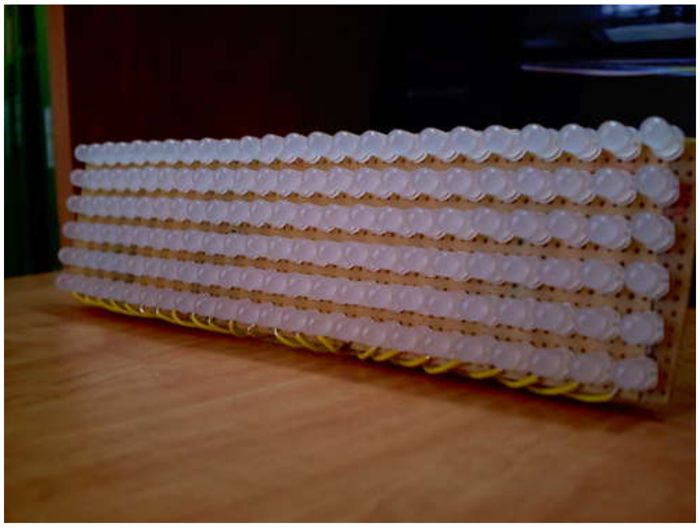
Looping text.txt (9 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Looping text.txt']

works.txt (13 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'works.txt']

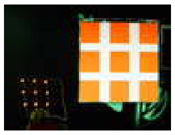
Code maker(6x24).xls (25 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Code maker(6x24).xls']

Step 6: We Are Done!

Congratulations you made yourself a 24x6 matrix and now you can display anything you like on the fly.
Now try to play with it and come up with a new program and a better interface.



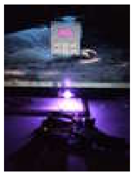
Related Instructables


GUI Controlled LED Matrix by Technochicken

Getting status through serial monitor by zaghy2zy

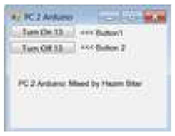
64 pixel RGB LED Display - Another Arduino Clone by madworm

Intro to Arduino by randofo


RGB's with Arduino and Processing by nazdreg2007

Make your pet dishes tweet! by thoughtfix

Controlling an Arduino with Cocoa (Mac OS X) or C# (Windows) by computergeek


Using Visual Basic 2010 to control Arduino Uno (Photos) by techbitar

Secret Knock Detecting Door Lock

by **Grathio** on October 12, 2009

Author:Grathio Grathio Labs
Creative swashbuckler. Writer for MAKE Magazine, presenter of inventions on TV, radio, magazines and newspapers. Professional problem solver. Annoyingly curious. Hacker of all things from computers to clothes to cuisine.

Intro: Secret Knock Detecting Door Lock

Protect your secret hideout from intruders with a lock that will only open when it hears the secret knock.

This started out as a bit of a joke project, but turned out to be surprisingly accurate at judging knocks. If the precision is turned all the way up it can even detect people apart, even if they give the same knock! (Though this does trigger a lot of false negatives, which is no fun if you're in a hurry.)

It's also programmable. Press the programming button and knock a new knock and it will now only open with your new knock. By default the knock is "Shave and a Haircut " but you can program it with anything, up to 20 knocks long. Use your favorite song, Morse code, whatever.

Maybe a video will explain it better:

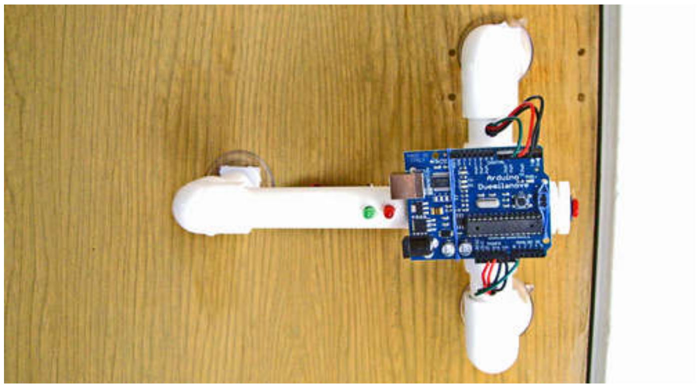
Important Notes:
(I hate to even have to say this, but since someone's going to say it, I'll say it first:)
1) This is for entertainment purposes only. Really. This decreases the security of your door by adding another way to unlock it, and it makes your unlock code known to anyone who can hear. If you put this on your door, be sure to carry your key too. The batteries might die, the suction cups might fail or you might forget your knock. Don't complain to me if someone imitates your knock and steals all your stuff, you've been warned.

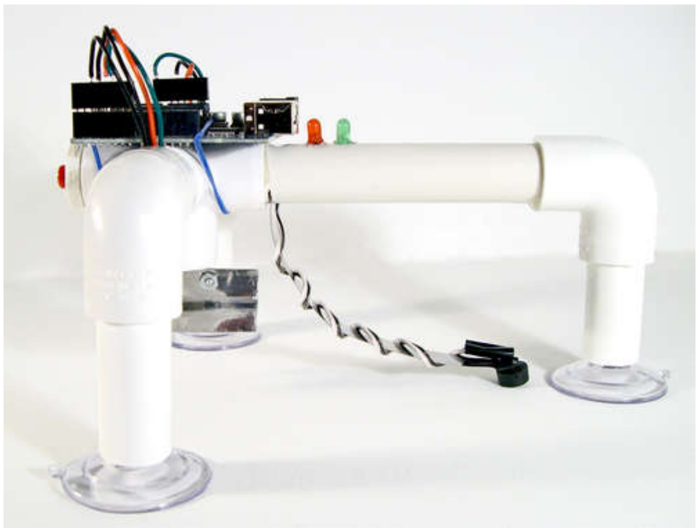
For obvious improvements to safety, security and whatever, see the final page of the Instructable.

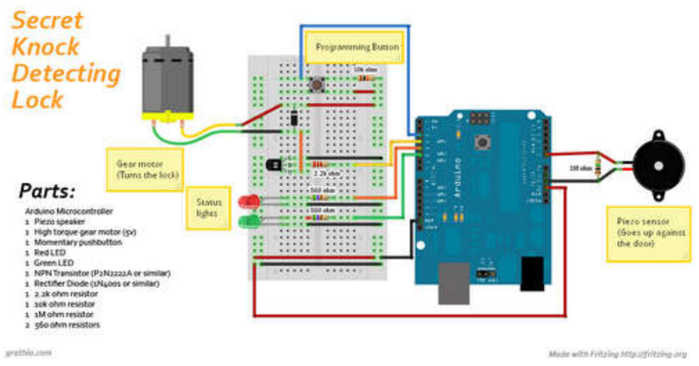
2) This is not a project for a beginner! Read through it carefully and be sure you understand it **before** you start! I will not take time to answer questions that are already in the instructions or from people who have gotten in over their head.

(If you think this project is too complex you might go here and sign up for the kit mailing list. The kits will be much more simple than this.)

Sorry about that. Now that that's out of the way, lets get to work.







Step 1: Tools, Supplies, And Skills

(If this all looks too challenging, you might consider signing up to the kit mailing list which, when available, will be much easier and a lot more simple.)

Time:

This project will take several hours to complete.

Skills:

To complete this project you should be able to do the following:
These are important! If you're not sure if you have these skills, read through the entire Instructable and make sure you understand it before starting anything!

- Basic soldering.
- Read a basic schematic.
- Basic knowledge of microcontrollers (I'll be using the Arduino.) This means you know what one is, how to upload data to it, and how to make minor changes to code.
- Improvisation. There are many ways to do this project, and you will have to make changes based on how your door and lock works.

Tools:

- Drill (ideally a drill press) and an assortment of drill bits.
- Saw capable of cutting PVC pipe. (ie: Pretty mcuh any saw.)
- Soldering iron and solder.
- Pliers.
- Screw drivers.
- Heat-shrink tubing and/or electrical tape.
- Wire stripper.
- Vice.
- Safety glasses.
- Gloves.

Other things you might find handy: a ruler/tape measure, a multimeter, a breadboard, some tape, a magic marker, sand paper, files, hot glue. And if you're like me a well stocked first aid kit.

Materials:

(The links are for example only, I don't necessarily recommend or have experience with any of these vendors. Feel free to suggest other sources in the comments.)

Electronics:

- 1 Arduino Duemilanove (Or compatible. Or really any microcontroller with at least 1 analog input and 3 digital outputs.) Buy from here, here, or here. And other places.
- 1 5v Gear reduction motor. The higher torque the better. Here's a good one. (14-16mm diameter is ideal because it fits inside of 1/2" PVC pipe.) I recommend one with at least 15oz/in (11 N-cm) of torque at 5v to turn a basic lock. [1]
- 1 Piezo speaker. (30mm) similar to this. You can use larger or smaller ones, smaller will be less sensitive.
- 1 SPST momentary pushbutton. (normally "off")
- 1 Red LED
- 1 Green LED
- 1 NPN Transistor P2N2222A like these or these (or similar).
- 1 Rectifier Diode (1N4001 or similar) this or this will do.
- 1 2.2k ohm resistor (1/4 watt)
- 1 10k ohm resistor (1/4 watt)
- 1 1M ohm resistor (1/4 watt)
- 2 560 ohm resistor (Or whatever will run your red and green LED's at 5v. How to tell.)
- 1 small piece of perf board. 5x15 holes or longer. (example)
- 1 9 volt battery clip and 9v battery. (Or any other way you can think of to get 7-12v to the Arduino. A wall adapter like this is a great option so you don't have to worry about batteries running out. 6 AA's would be another option for longer lasting power, but it will bring down the suction cups.)
- Connector wire. 20 gauge or narrower flexible wire in a number of colors for connecting the electronics together.

It's also a good idea to have a breadboard for setting up and testing the circuit before you solder it. We'll be doing this in step 3.

Case:

(These items are to make the project as pictured. Feel free to build a completely different and more functional case.)

- 20" PVC Pipe 1/2".
- 3 right angle 1/2" PVC connectors.
- 1 5-way 1/2" PVC connector. (example)
- 2 1/2" PVC end plug.
- 3 1 1/2" suction cups. (Available at hardware stores and craft centers.) NOTE: If your door is unsuitable for suction cups then replace these with three end caps and you can use adhesive strips or screws to mount the lock.
- 6" of 1/2" wide by 1/64" thick metal strip (steel, tin, copper, etc.) (available at hardware, craft, and art supply stores.)
- 4.5" of 1" wide metal sheet, 1/32" thick (steel, tin, copper, etc.) (available at hardware, craft, and art supply stores.)
- 2 3/32" x 3/8" screws with nuts. (1/8" will work too if you can't find the smaller ones.)
- 2 1.6M (metric) 16mm screws. Ideally with countersunk heads if you can find them. (For securing the motor. Check your motor specs to see what screws it needs. One motor I tried used 1.6M, the other 2M. You'll probably have to buy long ones and cut them to length.)

[1] If you have a torque meter or a torque wrench, apply it to your door lock to get an idea of what torque it will take to open your lock. Use a online conversion tool to convert between foot/pounds, N/m, etc.

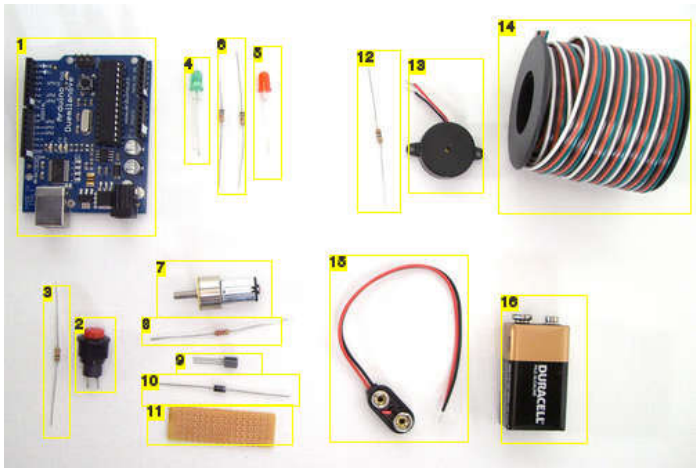


Image Notes
1. Arduino microcontroller.
2. Momentary pushbutton
3. 10K ohm resistor (brown, black, orange)
4. Green LED
5. Red LED
6. 560 ohm resistors. (Green Blue Brown)
7. Gear motor
8. 2.2K ohm resistor (red red red)
9. 2n2222 Transistor (NPN type)
10. Rectifier diode (1N4001)
11. Perf board 5x15 holes.
12. 1M ohm resistor (brown, black, green)
13. Piezo speaker
14. Some wire, 20-22 gauge. The more colors the better.
15. 9v battery connector
16. 9v battery. You can also run this project from the appropriate wall plug.

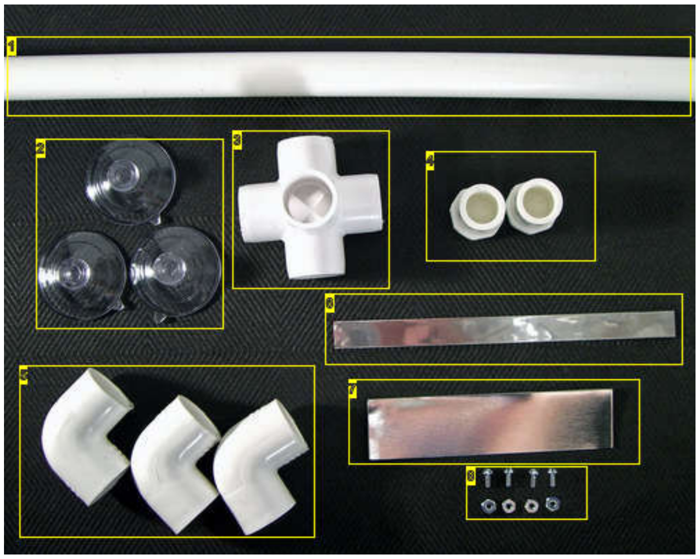


Image Notes
1. 20 inches of PVC, 1/2"
2. 1 1/2" diameter suction cups.
3. 5-way PVC connector. (Can be a little tricky to find.)
4. PVC end plugs
5. PVC right angle (90 degree) connectors
6. 1/2" wide metal strip used for a spring to press the detector to the door.
7. This metal strip will be made into the widget that connects our motor to the lock.
8. Screws and matching nuts. (Turns out you only need 2 pairs, not 4.)

Step 2: Program The Arduino

This section assumes that you know how to connect your Arduino microcontroller to you computer, compile and upload a sketch. If you don't know how to do that you probably shouldn't be doing this Instructable. But spending some time on this page and doing some of the examples and tutorials there might bring you up to speed.

We're going to upload our sketch before doing any of the electronics so we can test the electronics as we go.

#1: Download
Download the file secret_knock_detector.pde at the bottom of this section and copy it to your sketchbook. (Or view the text and cut and paste it into a new sketch.)

(Tip: If the name of the downloaded file is something like "BARS5HS13H8SW.tmp" simply rename it to secret_knock_detector.pde. and you're good to go.)

#2:
Open the sketch and compile it. It should compile properly the first go, but it's good to be sure.

#3:
Connect your Arduino and upload the sketch.

If you have any trouble, check the troubleshooting section at the Arduino site.

Code overview:
For the curious, here's a look at a few bits of code if you're interested in tinkering:
(If you're not curious, go to the next section)

about Line 28:**const int threshold = 4;**

This is the sensitivity of the knock detector. If you get a lot of noise, raise this (up to 1023), if you're having a hard time hearing knocks you can lower it (as low as 1).

about Line 29:**const int rejectValue = 25;**
about Line 30:**const int averageRejectValue = 15;**

Both of these are used to determine how accurately someone has to knock. They are percentages and should be in the range of 0-100. Lowering these means someone must have more precise timing, higher is more forgiving. **averageRejectValue** should always be lower than **rejectValue.**

Settings of about 10 and 7 make it hard for two people to knock the same knock even if they know the rhythm. But it also increases the number of false negatives. (ie: You knock correctly and it still doesn't open.)

about Line 31:**const int knockFadeTime = 150;**

This is a crude debounce timer for the knock sensor. After it hears a knock it stops listening for this many milliseconds so it doesn't count the same knock more than once. If you get a single knock counted as two then increase this timer. If it doesn't register two rapid knocks then decrease it.

about Line 32:**const int lockTurnTime = 650;**

This is now many milliseconds we run the motor to unlock the door. How long this should be depends on the design of your motor and your lock. It's okay if it runs a little bit long since I've designed a simple slip clutch into the design, but it's better for all the parts if it doesn't run too much.

about Line 34:**const int maximumKnocks = 20;**

How many knocks we record. 20 is a lot. You can increase this if your secret hideout is protected by devious drummers with good memories. Increase it too much and you'll run out of memory.

about Line 35:**const int knockComplete = 1200;**

Also known as the maximum number of milliseconds it will wait for a knock. If it doesn't hear a knock for this long it will assume it's done and check to see if the knock is any good. Increase this if you're a slow knocker. Decrease it if you're a fast knocker and are impatient to wait 1.2 seconds for your door to unlock.

about Line 39:**int secretCode[maximumKnocks] = {50, 25, 25, 50, 100, 5** .....

This is the default knock that it recognizes when you turn it on. This is weird rhythmic notation since every value is a percentage of the longest knock. If you're having a hard time getting it to recognize "shave and a hair cut" change this to **{100,100,100,0,0,0...** and a simple sequence of 3 knocks will open it.

Debugging:
about Line 51:**Serial.begin(9600);**
about Line 52: **Serial.println("Program start.");**

Uncomment these lines to see some debug info on the serial port. There are a few other lines of debugging code set throughout the rest of code that you can uncomment to see what's going on internally.

Be sure to set your serial port to the right speed.

The rest of the code is commented so you can see how it works but you probably won't need to change it if you aren't changing the design.

File Downloads

secret_knock_detector.pde (9 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'secret_knock_detector.pde']

Step 3: Lay Out And Test The Circuit

We're going to breadboard the electronics to make sure everything works. If you never make mistakes you can skip this step.

I've provided both a schematic and a layout diagram for the breadboard. Follow whichever one you're the most comfortable with.

We're going to go slowly and check as we go.

#1: Wire the Piezo Sensor
Solder a pair of 12" (30cm) leads to the Piezo speaker. Connect it between Analog pin 0 and the ground. Also attach the 1M ohm resistor between Analog pin 0 and the ground.

Test : With your Arduino plugged into your computer (via USB or Serial cable) and open the Serial Montor window. (That's the button furthest to the right at the top of the Arduino development environment.) With the Arduino powered on you should see the text "Program start." Tap the piezo speaker and you should see the text "knock starting" and "knock" each time you tap it. Stop for a second or two and you'll probably see "Secret knock failed" or "Door unlocked!"

If you don't see anything or see junk, make sure your serial port is set to 9600 baud and reset the power on the Arduino. If you're sure it's right, then try tapping Shave and a Haircut (Don't forget the two bits. See the video if you don't know it.) and see if you can get the "Door unlocked!" message.

If you get knock messages without tapping it may be too sensitive. If so you'll need to edit the sketch. Around line 27 raise the value of **threshold** . This can be raised as high as 1032 if you have a very sensitive detector.

const int threshold = 3; // Minimum signal from the piezo to register as a knock

Once you have it working the way you want it you can comment out (or delete) the lines that start with **Serial** ... We shouldn't need them any more.

#2: Wire up the LEDs
Lets wire up some LEDs so we don't have to use a serial cable to see what's going on.

Connect the red LED to digital pin 4 and green LED to digital pin 5 with their corresponding 560* ohm resistors in line.

Test : If you power the circuit the green LED should light. If not, check your connections and make sure the LED is the right way around. Every time you tap the green led should dim. After tapping the correct sequence the green led should blink a few times. Tapping the wrong sequence should blink the red one.

If none of this happens, check the polarity on your LEDs and all of your connections.

* Your LEDs might require different resistance.

#3: Wire the programming button
Solder 8" leads to the button. Connect one side of the button to +5v. The other pin on the button connect to digital pin 2 and, with a 10K resistor to the Ground.

Test : Apply power. When you press the button the red light should come on. Hold down the button and tap a simple sequence. When tapping while programming both LEDs should blink. When you're done the pattern you just tapped should repeat on both lights. After playback is complete, the new knock code is saved and the lights wil alternate red and green to tell you so.

#4: Wire in the motor
Solder 8" of leads to the motor and follow the design/schematic. Be sure to get the diode going the right way and you might want to check the pins on the transistor to be sure they match the diagram. (Some transistors might have the pins in different order.)

Test : Power the circuit. Tap the default "Shave and a Haircut" knock. The motor should run for about half a second. If not, check your connections as well as the polarity of the diode.

Extra Troubleshooting tips :

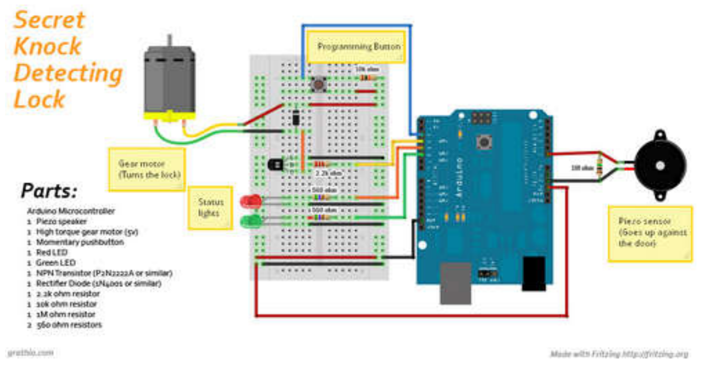
1) If the motor turns very weakly the diode might be reversed.

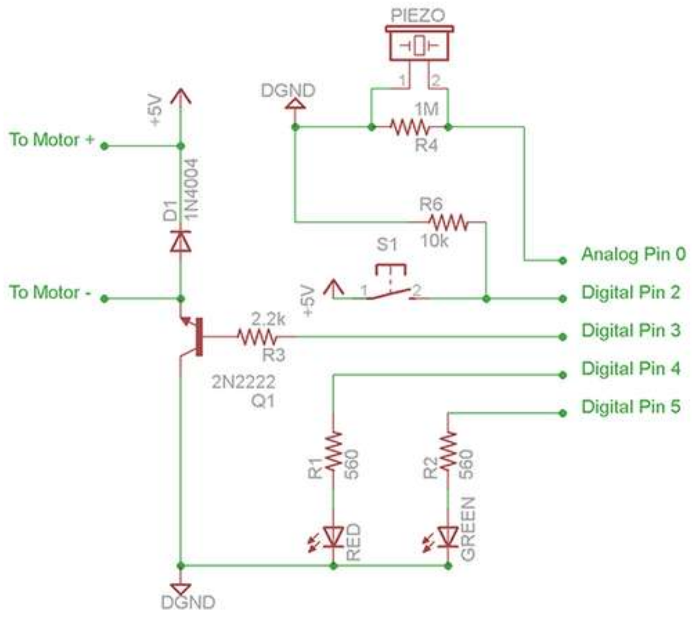
2) If you need more power on your motor make the following circuit change: Move the wire that goes from the motor to +5v to the Vin pin on the Arduino. This will supply the motor with 9v (or whatever voltage you're supplying to it.)

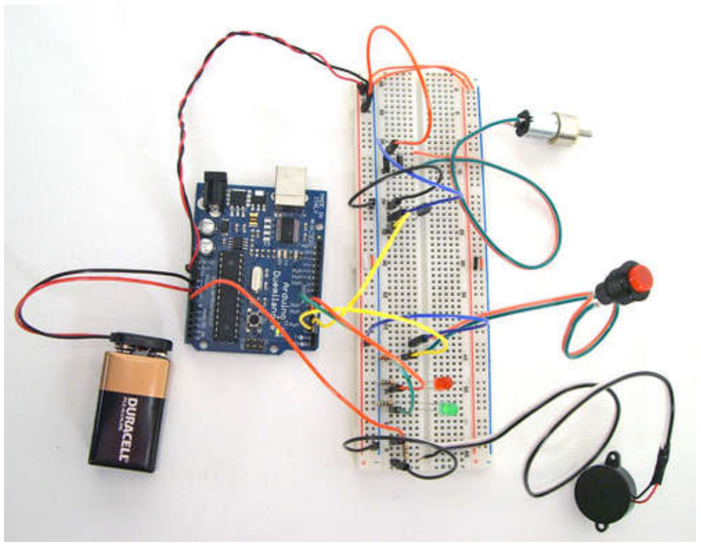
Tip : Check which way the motor turns. It should turn the same way as you turn your deadbolt lock to unlock it. If not, switch the motor's leads which should reverse the motor.

Congratulations! You have a working secret knock detector!

Now we have to put it into something more permanent that we can stick on our door.







Step 4: Prepare The Case

If you're making your own case you can skip this step. Otherwise grab your PVC and saw and lets get cracking.

Important!
We're just testing for fit here. Don't glue or fasten anything yet!

#1: The Button mount
Take one of the PVC end caps and drill a hole through the center appropriate for your push button. For my button that was 3/8". Secure the button in the hole.

Plug this into one of the 4 radial holes in the 5-way connector. (ie: not the one that points down.)

#2: The Motor mount
Take the other PVC end cap and drill a hole big enough for the shaft of your motor to pass through. You might also need to make it even bigger if your motor has a bearing that sticks out.

Depending on the design of your motor you may want to sand down the thickness of the plug so that you have more of the motor shaft to work with. Test the fit by placing the motor through the back of the plug. If it's too tight you might have to sand/file/grind the inside of the plug so it will fit.

Use a paper template to place the holes for the fastening screws, drill the holes and attach the motor to the plug. (In my case using the two 2M screws.) Countersink the screws if possible.

Plug this into the "down" facing hole on the 5-way connector.

#3: The 'arms'
Cut one piece of PVC pipe 5 inches long. We're going to call this the "long arm". Put a right angle connector on one end. Plug the other end into the 5-way connector opposite the button.

Cut two pieces of PVC pipe two inches long. We'll call these the "short arms". Half way along their length drill a 1/4" hole through one side. Put right angle connectors on one end of each arm. Plug these into the two remaining holes on the 5-way connector. You should really start to see it take shape.

#3b: A Few Extra Holes
with a pencil or marker draw a line down the center of the top and the bottom of the long arm. On the top side, make marks for two holes, one 3/4" from the 5-way, and another 1 1/2" from the 5-way. Drill a 3/16 (5mm) hole at each of these places. This is where our LEDs are going.

Also make a line along the bottom where the long arm connects to the 5-way. Using a saw, cut a short way through the pipe, from the bottom up, until there is about a 1/2" hole into the pipe. (this is where the spring for our detector will attach. Also on the bottom, drill a 1/8" hole 1/4" further along the pipe (Away from the 5-way). We will thread the sensor's wires through here.

#4: The 'legs'
These are the parts that attach to the door. You may not want to cut these yet, The length depends on the design of your door lock, the length of the shaft on your motor and the final design of the Lock Turning Clamp in the next step. All three of mine were 2 5/16" long, but you're better off cutting them long and trimming them down to size later

If they're too long the motor won't reach the lock to turn it. If they're too short the suction cups won't reach the door.

When you do cut these, hot glue the suction cups in one end and stick the other ends in the right angle connectors on the ends of our legs.

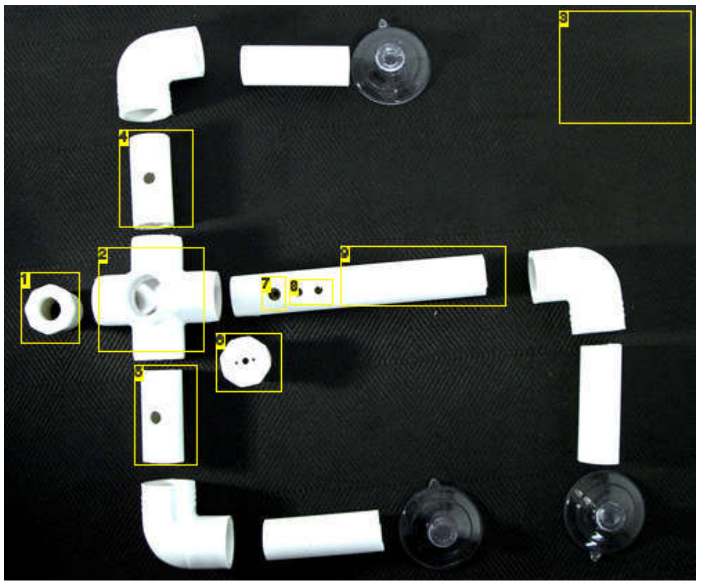


Image Notes
1. Button mount. End plug with hole drilled for the panel mount button.
2. 5-way connector. The motor goes in the center of this.
3. This is an exploded view with all of the parts laid out how they attach (though the angles are wrong on some parts.
4. Top arm. Hole drilled to let wires through.
5. The "Board Arm" This is the arm we're putting the circuit board into. Hole drilled to allow wires through.
6. Motor mount. Will go in the bottom hole on the 5-way.
7. Ignore this hole. Not sure what I was thinking.
8. holes for LEDs.
9. The "Long Arm". (It's longer because it provides more holding power against the torque of the turning lock.)

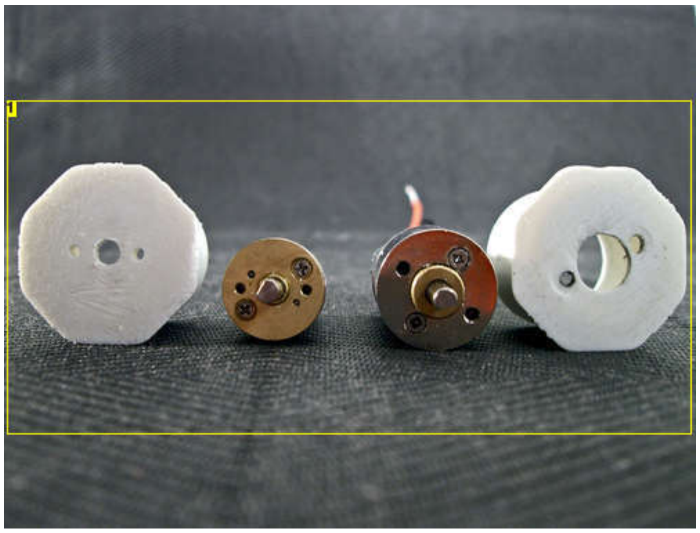


Image Notes
1. Two sample motor mounts for two different motors. The one on the left is smaller but actually more powerful. The one on the right has a larger hole to allow clearance for the bearing. The facing surface on both of these have been ground down so more of the shaft can stick out.

Step 5: Make The Lock Turning Clamp

In this section we make the all important part that connects the motor to the lock.

(This is a reasonably clunky way to do this, but it's simple and cheap. If you think of a better way, please mention it in comments.)

What we're making is a clamp that attaches to the D-shaft of our motor and fits easily over the lock latch so that it can turn the lock. It attaches securely to the motor, but there is some give in it so that it can slip if it finds its self between a rock and a hard place. (Which we prefer to wrenching the project to pieces.)

Drilling the holes:
First take the piece of metal that's 4 1/2" long and 1 1/4" tall. and cut it in half so you get two 2 1/4" pieces.

Tape them together, mark each side so you know which side is "out", and mark one of the long edges as "up". This will all help you keep everything lined up as you go.

Measure and mark the center line from top to bottom. 1/2" each side of this center line and 1/4" from the top mark holes for drilling.

Drill 1/8" holes at these marks. Marking the points with a punch, or giving it a whack with a hammer and nail will make your drilling more accurate.

The edge with the holes is the side that attaches to the motor.

Bending the metal
Measure the width of your lock latch (the narrow way) and divide by 2. This distance is how much zig we're going to bend into each piece of metal. Mark this zig distance along each strip. Bend one piece so it zigs to the left, the other so it zigs to the right. Make sure that the screw holes at the top of the pieces stay lined up and the bends don't keep the pieces from meeting at the top.

Finishing and sizing
For this part you'll need your motor, the two 1/8" screws and a couple matching nuts.

Put the screws through the holes in the top of the plates so it makes an upside down "Y" (sort of) and place the motor shaft in the top between the screws. Screw nuts on each side and tighten until it's firmly (but not really firmly) attached. The small amount of give between the metal and the shaft will let the motor spin if it meets too much resistance. (Rather than breaking something important.)

Check the other end for fit over the lock. It should fit a little loosely over the lock latch. Not so firmly that it's clamped tight, but not so loose that it can turn without turning the lock. Adjust the bend of the flanges if you need to.

After you've got the adjustments right, tighten another nut onto the end of the screws and tighten them up against the first ones. This will help lock them in place.

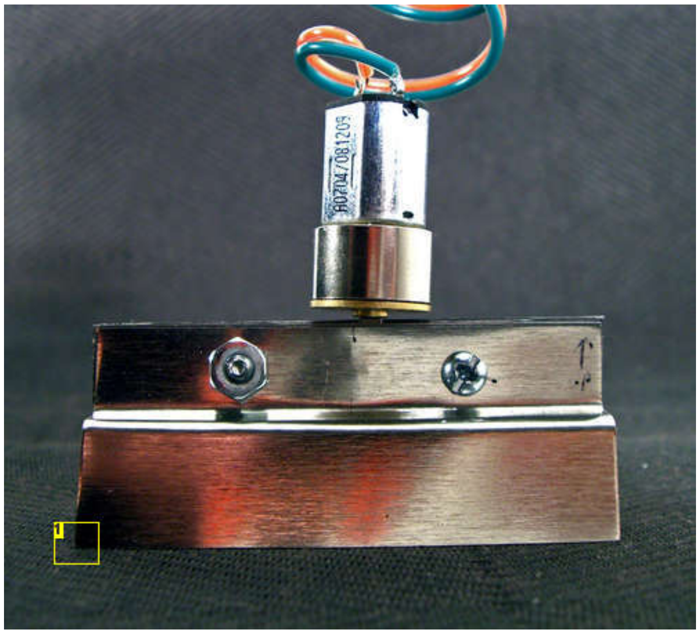


Image Notes
1. It's not a bad idea to file down the sharp corners.

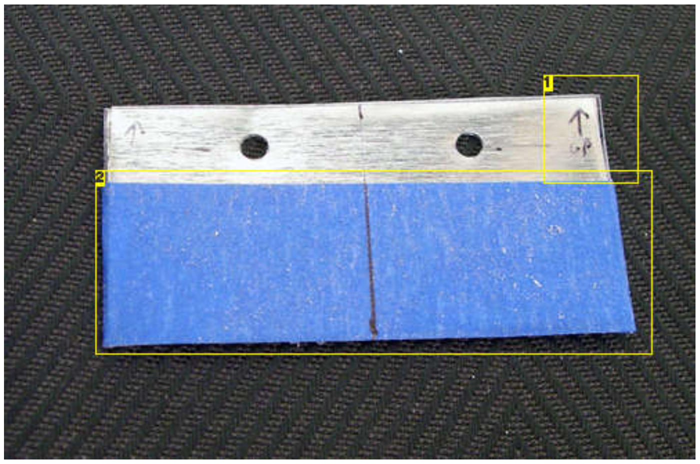


Image Notes
1. A few markings will help keep them straight when you put them back together.
2. Two pieces taped together so we get the holes lined up.

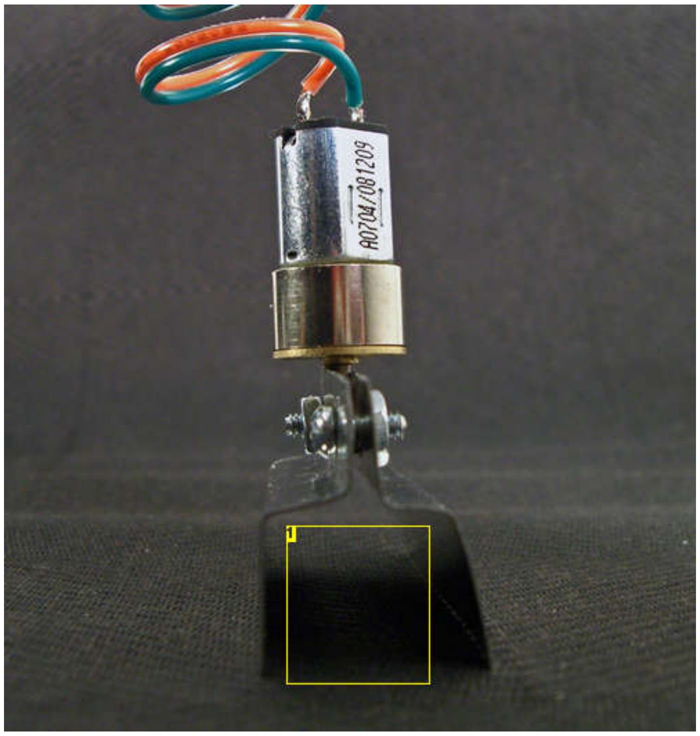


Image Notes
1. Edge view. Your lock will slot in here.

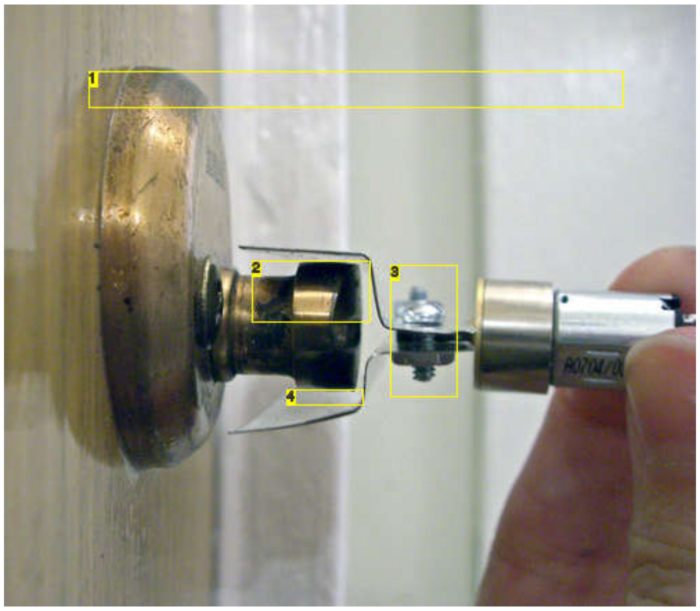


Image Notes
1. In position over the lock.
2. 1/2 of the thickness of the lock handle. This is how much we "zig" each flange.
3. Tighten the screws so they clamp down on the D-shaft.
4. Having a little bit of space is fine and will make it easier to put on.

Step 6: Make The Knock Detector Spring

In this section we're going to put our knock detector on the end of a little springy bit so it presses securely up against the door. You could just use a piece of tape or even glue or screw it straight to your door, but doing it this way keeps it portable.

#1: The parts:
You'll need

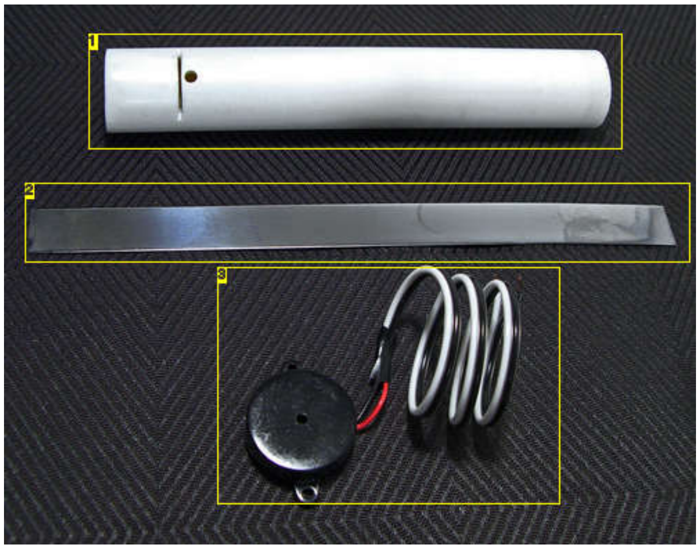
- Your strip of thin metal (which is 6" of thin 1/2").
- The piezo sensor (which should have about a foot of leads soldered on.
- The piece of PVC I've been calling the "long arm".

The PVC pipe segment, on the bottom side, should have a slot cut 3/4" from the end and a 1/8" hole just inside of it.

#2: Attach the sensor to the metal strip.
Using glue, hot glue, tape, etc and fasten the piezo sensor to one end of the metal strip. Wrap some of the remaining wire around the strip so that it stays out of the way.

(If your piezo sensor has its leads on the back then drill a hole through the strip. be sure to cover the leads with insulating tape or heat-shrink.
#3: Attach the metal strip to the PVC.
Thread the free end of the wire through the bottom hole on the PVC and then insert the free end of the metal strip in the slot. Bend the strip as shown so that the sensor faces out and down and will lay flat on the door.

The strip should stick in the slot with friction, but if not, take some pliers and bend over the end of ths strip that's inside the pipe.



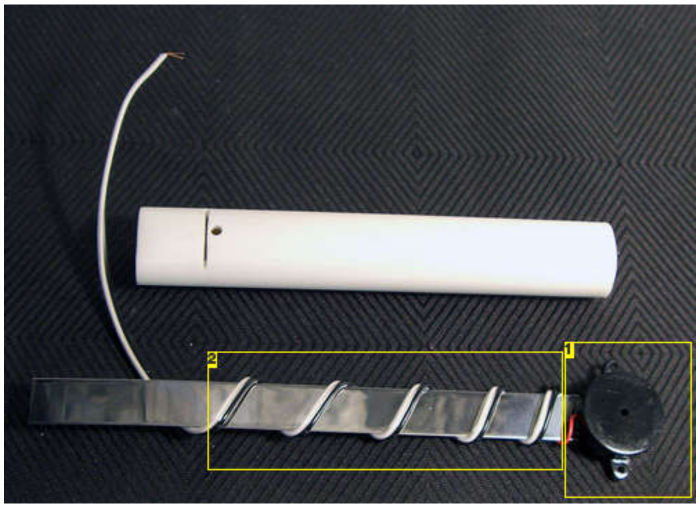


Image Notes
1. Hot glue (or otherwise attach) the sensor to the end of the metal strip. Make

Image Notes
1. The bottom side of the Long Arm with the 1/2" slot for the metal strip and a hole for the wire.
2. 4-6" flexible metal strip.
3. Piezo sensor with about a foot of wire attached.

sure the "open" end is facing out.
2. Wrap some of the remaining wire around the strap to keep it out of the way.

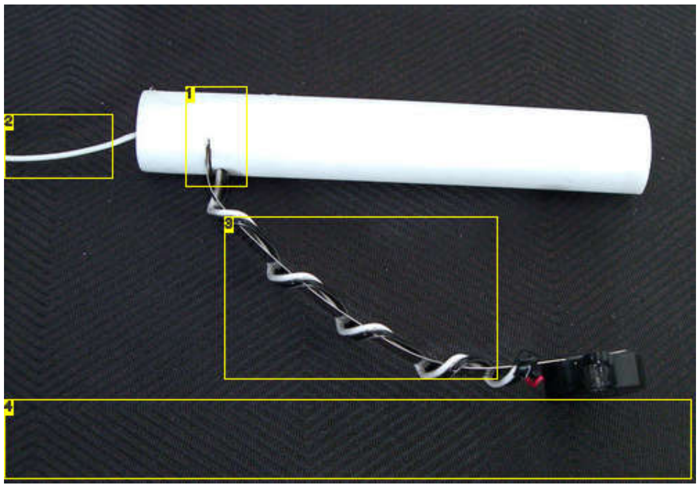


Image Notes
1. Thread the wire through the round hole and slide the free end of the metal strip into the slot.
2. The free end of our wires.
3. Bend the strip more or less like this.
4. This area is where the door will be, Try to bend it so the sensor lies more or less flat against the door.

Step 7: Soldering The Circuits

Due to the needlessly complex nature of my case, soldering and case mounting are somewhat intertwined, but I'll try to break it down so it makes sense.

I recommend that after each step you plug in and test the circuit to make sure you didn't make a mistake, moving each bit from the breadboard one at a time. Having to desolder components is no fun.

Tip : Use wires in as many colors as you can get so you can keep things straight. I also usually put labeled bits of tape on the ends of the wires to help me remember.

Tip : If you're using stranded core wire, be sure to tin the ends . It will help with your joints and make it easier to stick them into the breadboard for testing.

#1 Solder leads to the LEDs. (Hey, that almost rhymes!)
About 10" or so should work.

Okay, so much for the easy steps. After this it gets more complex because most of the components need to be threaded through various holes in our case before they're soldered. Of course if you made a different case then you don't need to worry about most of the tedium and can get right to soldering the components to the perfboard.

Tip : Mount the components as closely as possible to the perfboard. There isn't much clearance inside the pipe.

#2 Prepare the perfboard.
We're using a perfboard with 0.10" spacing where each hole has an individual copper pad. Cut the perf board to size (5x15 holes) and then sand/file/grind off some of the edges so it fits easily into the 1/2" PCV pipe.

For future reference we're calling the side with the copper "back" and the side with the components the "front".

#3: The ground line
Since the ground is the most common terminal in the project we're going to run a ground line across the back for the components to connect to. To make this I too a 9 inch piece of solid core wire that I'm using as my ground wire and stripped about an inch off one end. The soldered between hole 1 and hole 10 (see the attached diagram). Then I'll tack the other gronded components to it as the come through the board. (You can also just bridge the connections with solder, but I hate doing that because it can get messy. My soldering is messy enough.)

The other end of this wire will go to a Ground pin on the Arduino. (This is a good time to label the other end with a piece of tape.)

#4: The +5v line.
There are also a couple points where we want to supply +5v. This is the same idea as the ground line but we only need about half an inch stripped.

The other end of this will connect to the +5v pin on the Arduino.

#5: The LEDs.
Solder the LED's resistors (560 ohm by default) in place as shown.

You have two choices of how to deal with the LED's. You can mount them on top (the easy way) or you can mount them from the bottom, which looks better, but is a pain because 1/2" PVC doesn't give you much room to work. If you mount them from the top, be sure to thread the leads through the holes before soldering.

Thread all 4 leads from the leds out through the near end of the 'long arm' through the 5-way and through the 'board arm".

Solder the cathode (-) lead (the short one) from each LED as indicated. The anode will connect to digital pin 4 (red) and 5 (green) on the Arduino. (Thread the Ardunio leads through the "short arm" of the 5-way.)

#7: The knock sensor.
Solder the speaker's 1M ohm resistor in place on the board.

Make sure you have the speaker mounted firmly at the end of the spring and the wire is wound a few times around it to keep it out of the way. Thread the wire through the long leg, through the 5-way and into the short arm that we're keeping the circuit board.

Solder one end of these leads to each side of the 1M resistor. Then solder a 8" lead from the ungrounded end of the resistor. This will go to Analog pin 0.

#6: The button.
Solder the 10K ohm resistor in place as shown.

Fasten the button through the hole on the end plug, then put the plug on the 5-way connector and thread both wires through to the 'board arm' hole.

Solder one lead from the switch to the resistor. The other end to the +5v wire.

Solder a length of wire from the resistor according to the diagram and label it "Digital 2".

#8: The motor.
Nearly done.

Solder the diode, transistor and resistor in place. (Make sure you get the direction right on the diode. And the transistor for that matter.)

To the free end of the 2.2k ohm resistor solder a 8" lead that will go to digital pin 3.

Put the motor in place in the bottom hole of the 5-way connector, thread the leads trough and solder them in place on either side of the diode, making sure you've got the motor wires in the right order so when it runs it will turn to unlock.

#9: The Arduino pins
Connect the labeled wires to their appropriate places on the Arduino.

Test :
Wait, you don't need to do this, right? You've been testing as we go, haven't you?

Plug some power into the Arduino and make everything works. Especially make sure that the motor spins in the right direction to unlock your lock.

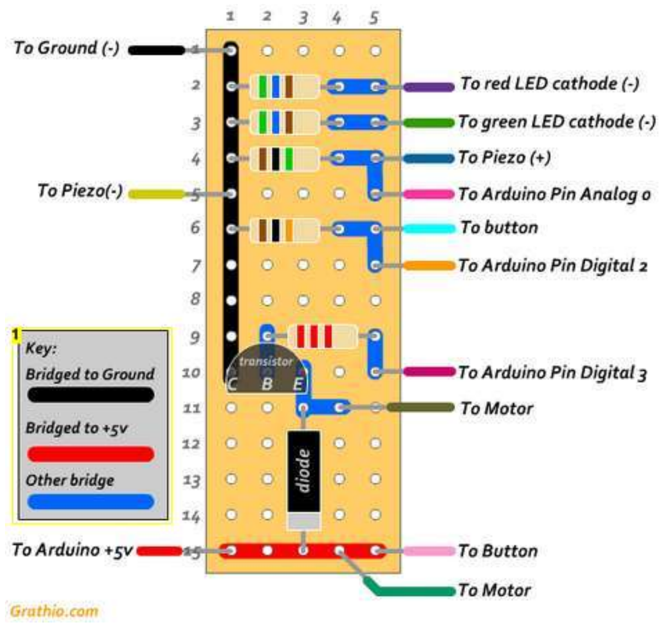


Image Notes
1. Check the pin order on your transistor. Some times they're reversed.

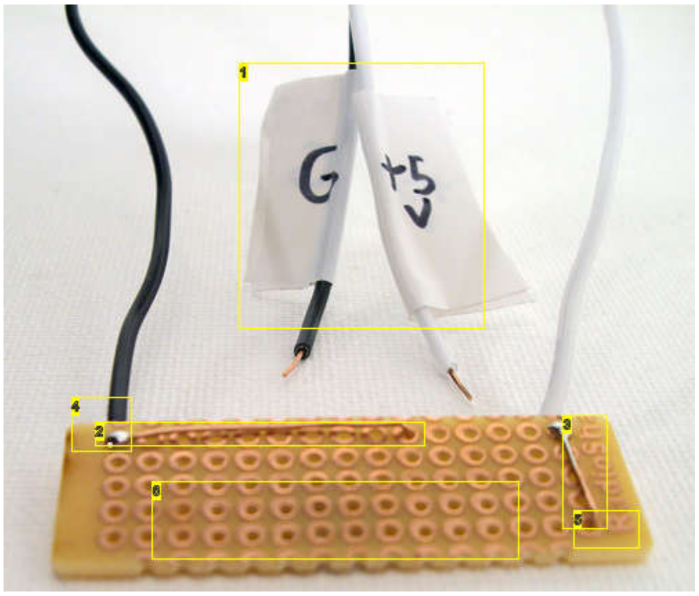


Image Notes
1. Tape labels are a good idea for this project if you're making the PVC pipe case. Misrouting wires is a real possibility.
2. The ground wire. We'll tack other components to this as we add them to the board.
3. The +5v wire. We'll tack other wires to this as we add them to the board.
4. In the soldering diagram this hole is hole #1,1
5. On the soldering diagram this hole is #5,15.
6. This is the bottom of the board, compared to the soldering diagram.

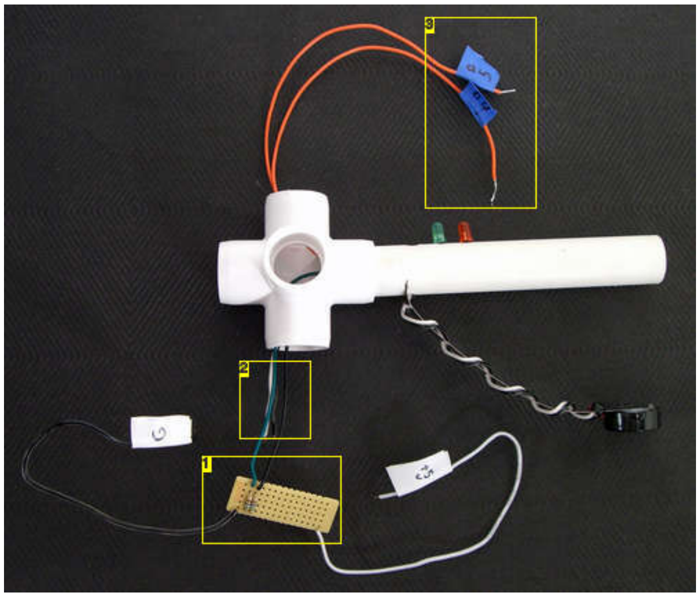


Image Notes
1. #5: The solder the resistors in place to the ground line, and the short leads of the LEDs to the resistors. Pass the leads through the Board Arm hole of the 5-way.
2. While we're threading wires we have threaded both the Piezo sensor wires through for the next step.
3. The + leads for the LEDs come out up here. And we mark them with tape so we don't confuse them later.

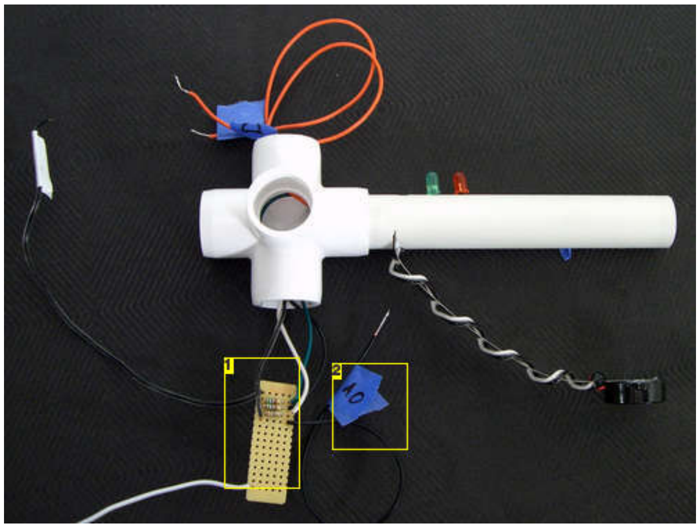


Image Notes
1. The1M resistor soldered in with the piezo sensor attached either side.
2. A lead attached that will go to Analog 0. (We don't need to thread this anywhere for right now.

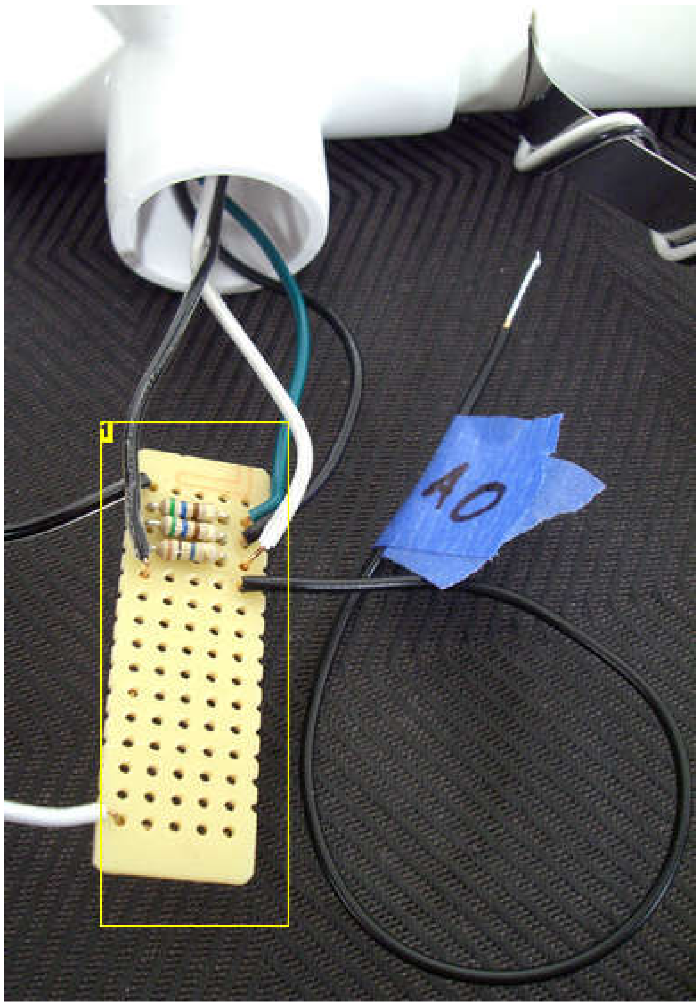
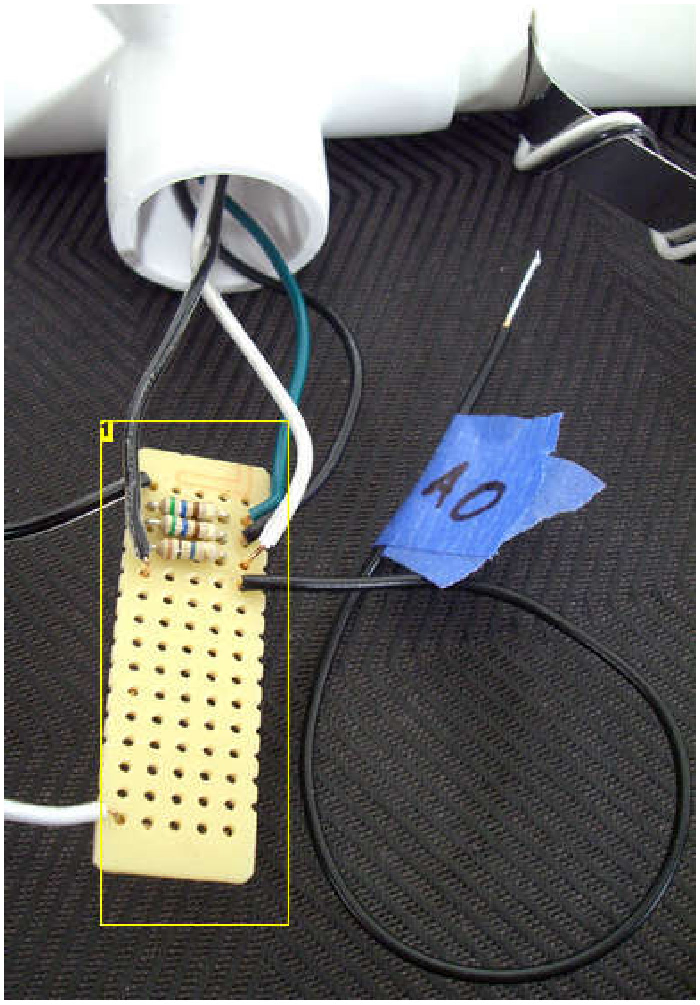


Image Notes
1. Closer look at the resistors and their connections. This is what it should look like after #7, above.



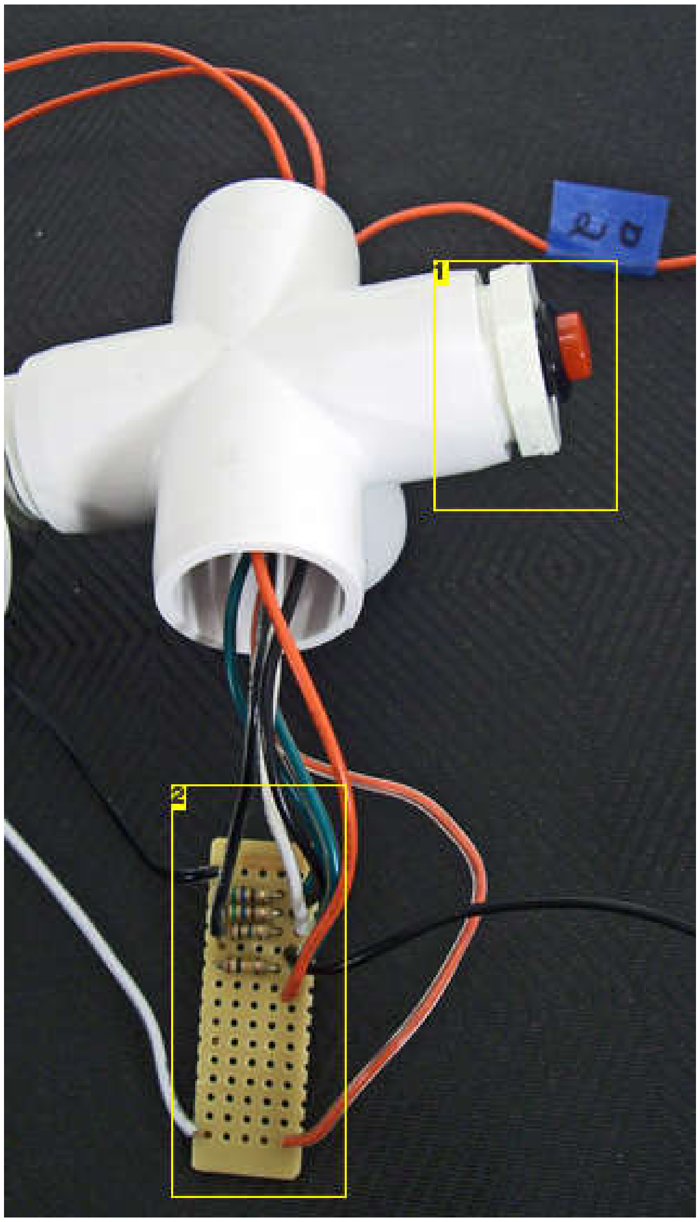


Image Notes
1. The button in place with the wires fed through the 5-way.
2. The resistor and button connections.

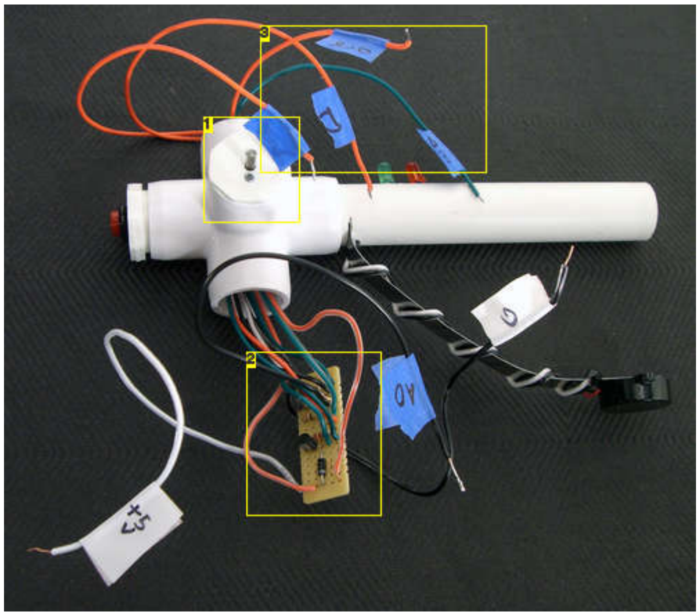


Image Notes
1. The motor in place.
2. The final circuit with the wires fed through.
3. All the connections that will go to the Digital side of the Arduino we thread through to the Short Arm side.

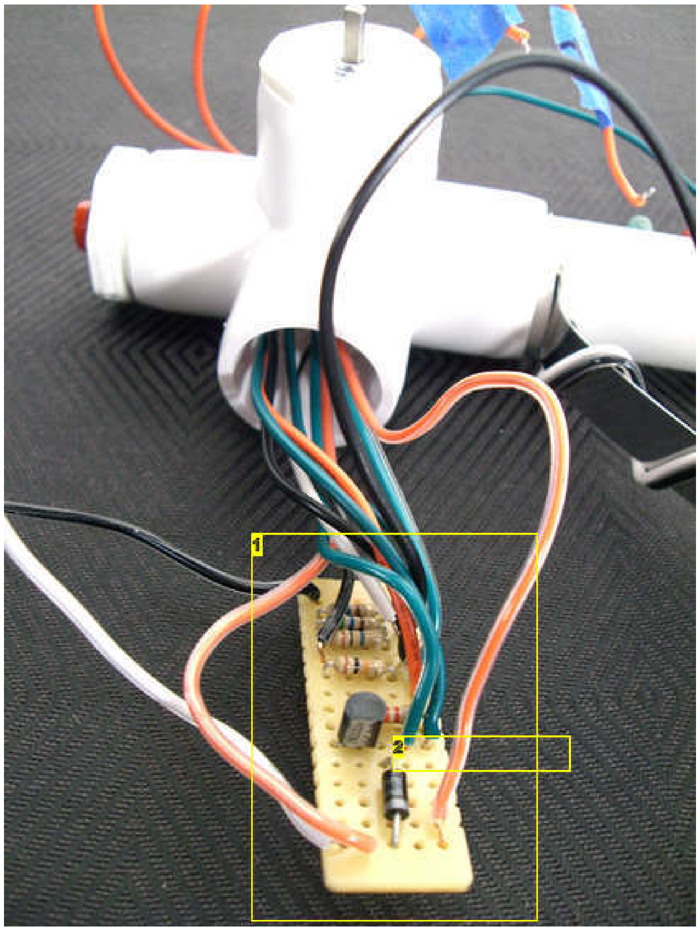


Image Notes
1. Close up of the completed board.
2. Embarrassing note: the motor connection (the green connection on the left) is not in the same hole as the diagram. It's connected the same, just in the wrong hole.

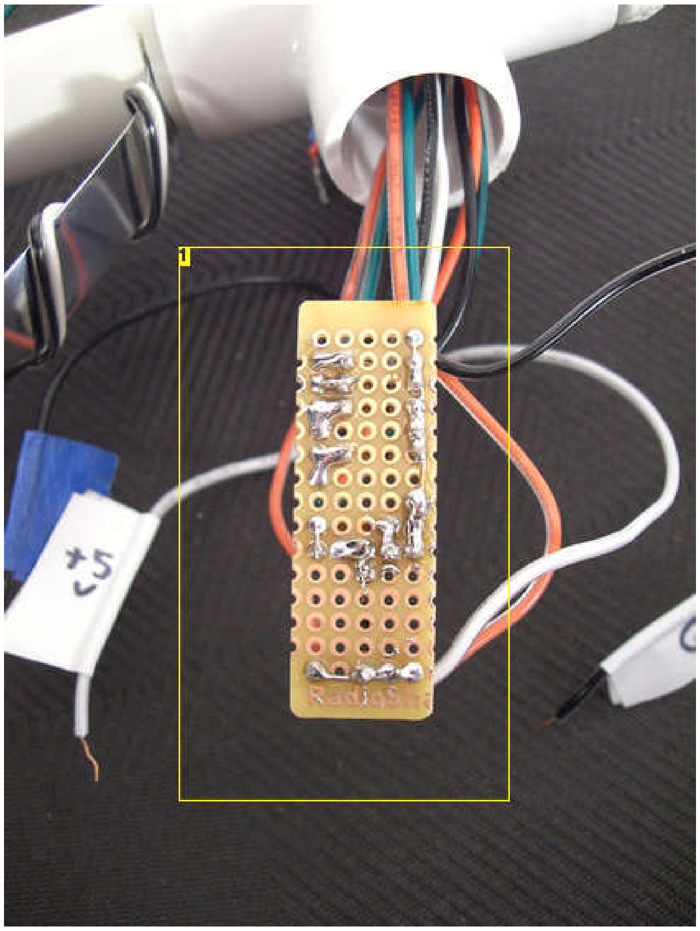


Image Notes
1. Here's the back of the board so you can see the bridged connections. And a lot of messy soldering.

Step 8: Assembling The Case

All of your components should be in place and the circuit should be working. We're almost there, all we have to do is cram all that wire and the circuits into the pipe.

#1 Getting the wires to the correct side .
If you had the wires connected to the Arduino, unplug them.

The wire to Analog 0 and to Ground and +5v will come out of the Board Arm, so we don't do anything with those yet.

The other wires to Digital 2,3,4 and 5 thread through the 5-way to where the Short Arm will go.

#2: The Short Arm
Speaking of the short arm... Pull the wires for Digital 2,3,4 and 5 through the hole in the middle of the Short Arm pipe.

#3: The Long Arm.
The long arm has the LEDs and the sensor in it. Using needle nose pliers, (or a bit of coat hanger with a small hook on the end, or a crochet hook) pull up the slack on these wires as you plug it into the 5-way.

#4 The circuit board.
The circuit board should be the first part put in place inside the pipe for the Board Arm. Thread the wires for Ground, +5v and, Analog 0 through the small hole on top of the arm.

Now make a tight bundle of the wires around the circuit board putting even pressure on it, being careful not to bend, break or spindle the thing. Gently slide it into the Board Arm. If you have a lot of extra wire lengths you might want to push it out the far side about half an inch so there's more room for wire on the inside.

When it's in place, plug this short arm into the 5-way.

#5 The Motor .
The motor should already be in place in the bottom of the 5-way. But if it's not nows the time to put it there.

#6: The Button .
The button should also be in place, but if not, put it in. If you have a bunch of extra wire getting jammed up inside the 5-way, you can try pulling some of it (gently!) to the button side of the 5-way since it doesn't take up much space.

#7: The legs and suction cups.
The arms should all be plugged in. Attach 90" turns to the ends of the 3 arms, and plug the legs in to the other end.

Suction cups should fit snug into the bottom of the legs. If not, some hot glue will get them into shape.

(If you're not using suction cups then this is where you use your alternate solution.)

#8: The Arduino and battery
Yes, this is ugly as sin. I works, but... yeah. If you come up with anything better, you're welcome to it.

Stick the Arduino onto the top of the frame. I used lengths of insulated wire. It worked...

Attach the battery in a similar way somewhere where it can power the Arduino. Again, I used insulated solid core wires. At one point I used rubber bands which also worked just fine.

Tape? Yes, that would work too.

Plug in the wires in where they labels say they should go. Might as well test again it since it's all hooked up.

Whew! Now we're ready to attach it to the door!

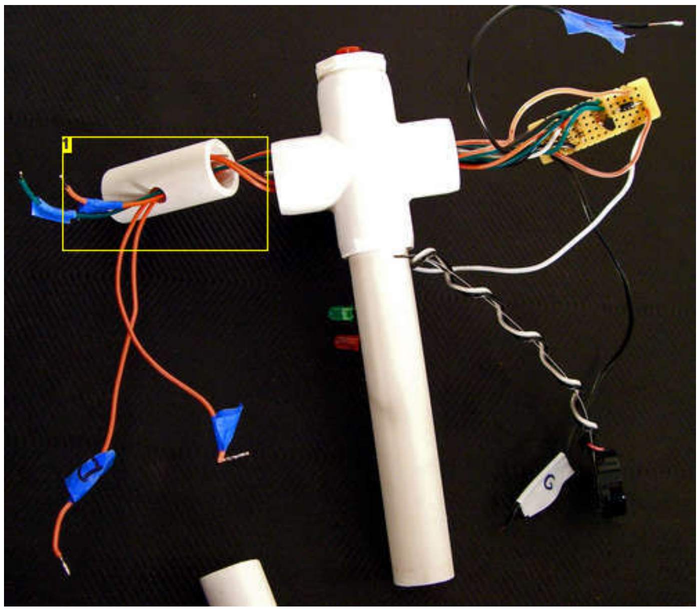


Image Notes
1. The digital pin wires threaded through the Short Arm

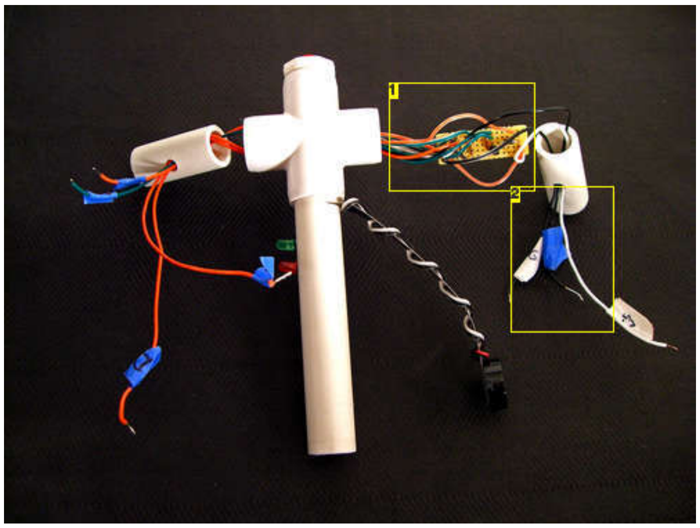


Image Notes
1. Circuit board getting ready to be crammed into the Board Arm.
2. The wires for Analog 0, Ground and +5v threaded through the hole in the Board Arm.

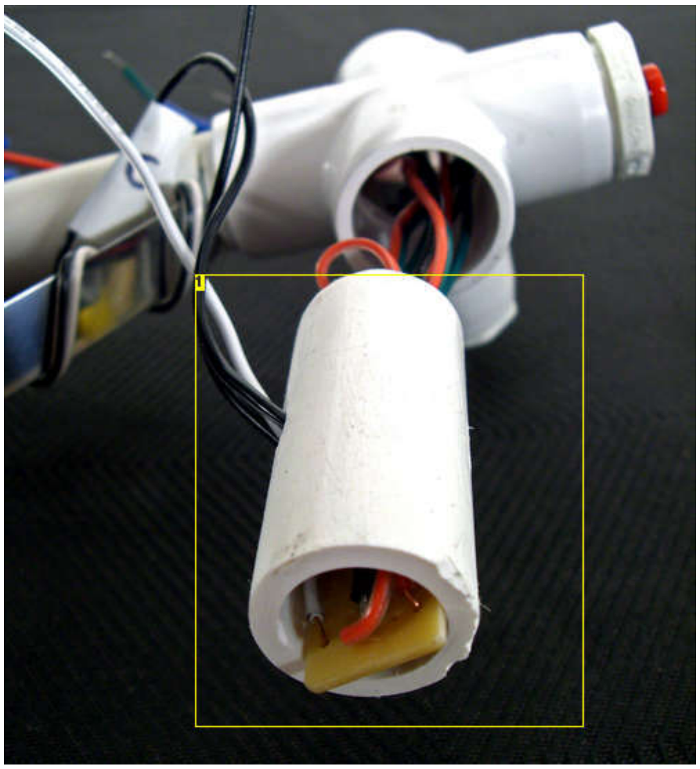


Image Notes
1. The circuit board has been pushed into the Board Arm. Having a little sticking out the far side is fine, it'll be hidden in the right angle adapter.

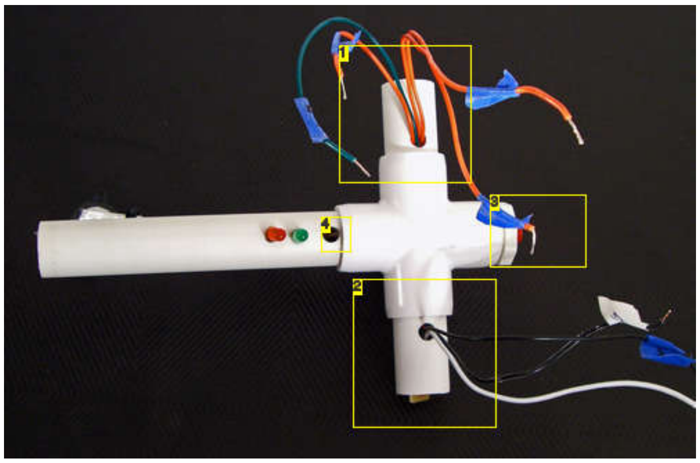


Image Notes
1. Short Arm plugged in.
2. Board Arm plugged in.
3. Button plugged in.
4. Once again, ignore this hole.

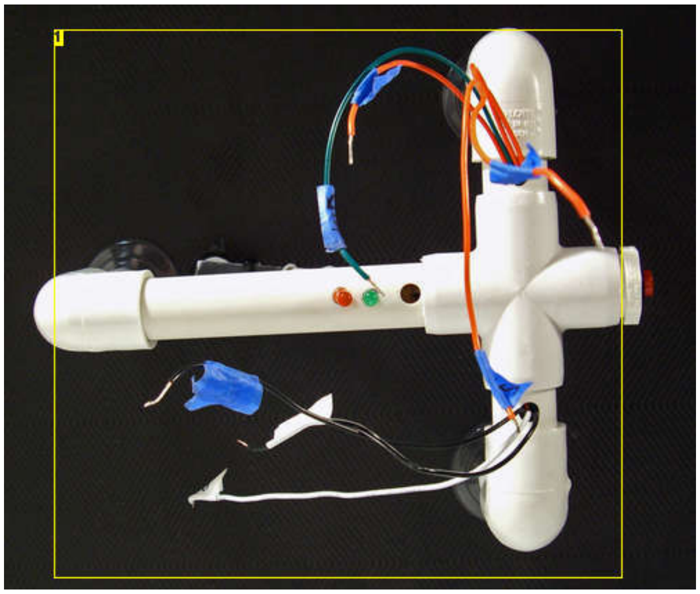


Image Notes
1. Legs and suction cups in place. at the end of the arms.

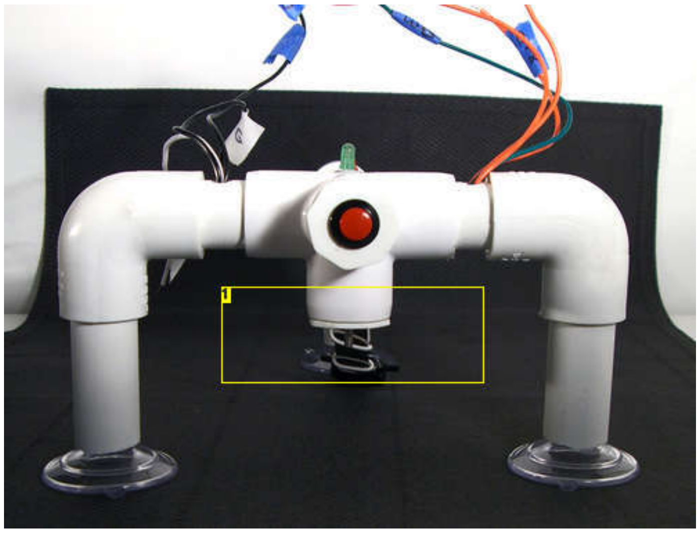


Image Notes
1. Side view. We haven't attached the lock clamp yet.

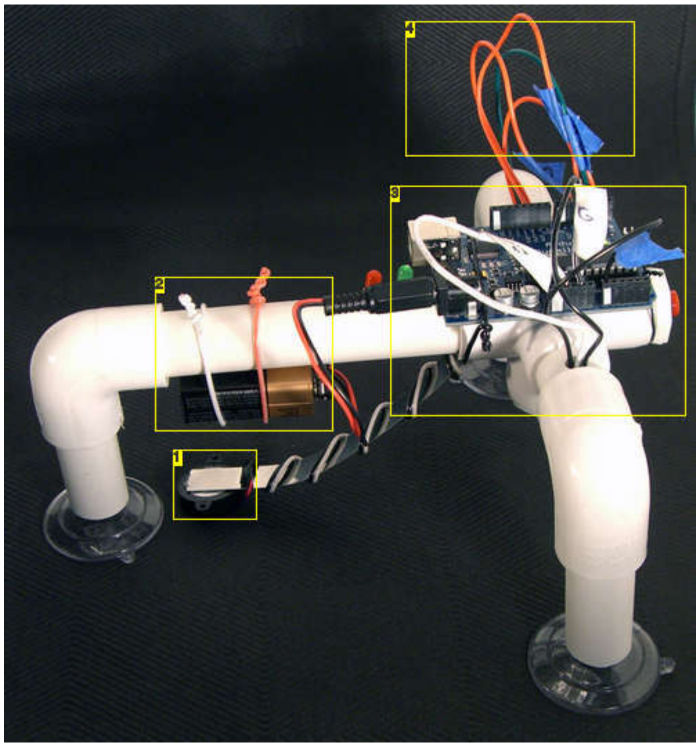


Image Notes
1. The knock sensor is arranged so it will press flatly against the door.
2. Battery attached.
3. Arduino attached and wires plugged.
4. Here is where a neat person would shorten the leads.

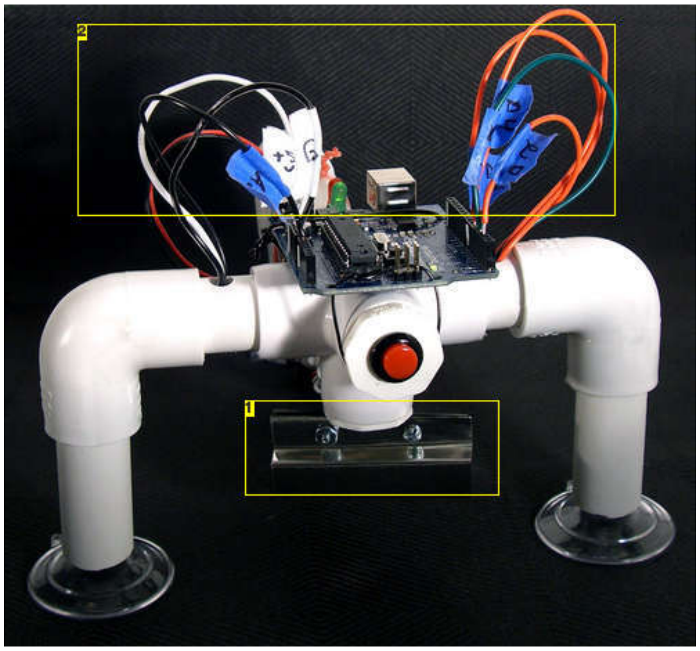


Image Notes
1. Side view. We've attached the lock clamp. Now we're ready to test it on the door!
2. Shortening the wires and removing the labels would make it look nicer, but you'll have to do that on your own time. I have an Instructable to finish!

Step 9: Mounting, Testing, and Use

First lets make sure our legs are the right length. Be sure the lock clamp is attached to your motor and the legs are in place. Attempt to fasten it to the door so the clamp fits over your lock. If you're lucky the legs will be the right length and you're ready to go! If you're unlucky the legs are too short and you'll have to cut new ones. But chances are your legs will be too long. Either grind/file/sand/cut them to the right length so the clamp fits right over the deadbolt lock handle.

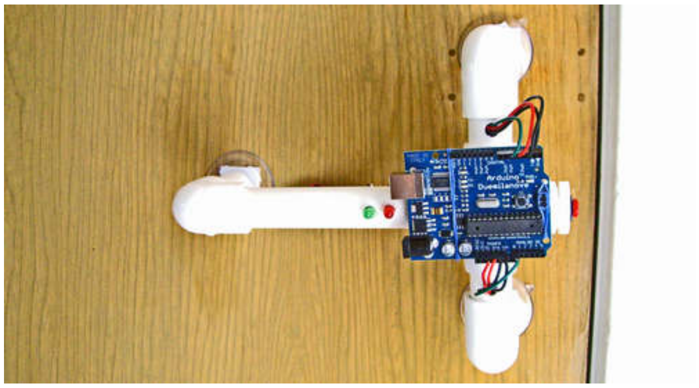
You don't have to use suction cups and in fact may doors are immune to their sucking. Two other options are to put PVC end caps on the legs and then secure them to the door with double sided foam tape (like this) or with screws through the caps, if you don't mind putting holes in your door.

Now that's its on you can give it a test. Do the first test from the inside. Lock your door and power it up. When the green light is on give it the old Shave and a Haircut and it should unlock!

Now program in a less obvious knock (or not) and your tree house will finally be safe from that smelly kid!

If the motor doesn't turn far enough to unlock your door you'll need to update the sketch to run the motor longer. (See Step 2!)

For additional Troubleshooting : See the bottom of **Step 2** . It also includes a bunch of other tweaks that might help you.



Step 10: Epilog: Changes And Improvements

There are many ways to improve or change this project if you're feeling ambitious. Here are a few to get you started, feel free to add more in the comments.

- Add an H-Bridge to the circuit so it can lock and unlock the door.
- Make it work in silent mode by removing the knock sensor and attach a capacitance (touch) sensor to the doorknob and record sequences of touches.
- Use a servo to unlock the door rather than this hacked together gear-motor+slip transmission.
- Add a potentiometer to adjust the knock sensitivity and other values.
- Build it into an actual door knocker.
- Use a more economical microcontroller and enable sleep mode for better battery life.
- Make the whole package small enough to fit inside the door.
- Store several knocks so several people can have their own 'private' knocks.
- Add a real-time clock and using different knocks for different days of the week.
- Add a knocker to provide feedback through the door. It could then offer a challenge-response security where the door starts a knock sequence and the user has to finish it correctly.
- Remove the knock sensor and record pushes of the doorbell or other hidden button.
- Remove the knock sensor and put a photosensor in the peephole, send the open code through the peephole with a keychain flashlight.

And here's a zero-technology solution to the "Yeah, but someone'll overhear your secret knocks!" problem: Scream while knocking. No one will overhear the knock over the racket you're making.

Did you build this?

Post a photo (or better yet a video!) photo of it mounted on a door will earn a Master Of Secret Knocks patch*!

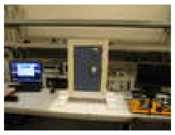
*As long as I have patches left to hand out. Which I do.

Masters of Secret Knocks:

- josiasfilho added a servo and locking ability.
- Jorad unlatches his door and added feedback in the peephole.
- Crimson-Deity added a pushbutton.
- bserrato added unlocking and a bluetooth camera to photograph people who give an incorrect knock.

Image Notes
1. Post a picture (or video!) of your secret knock detector in comments and get this fancy Master of Secret Knocks patch

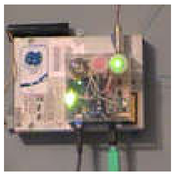
Related Instructables


Secret Knock Detecting Door Lock (Photos) by vinny03

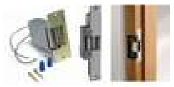
How to Break Down a Door by drums787

Magnetic Combination Lock Picture Safe by pastprimitive

Very Simple Arduino Electric Lock by RKlenka


Nintendo Keyless Entry System by action_owl

A bit of safe cracking... by killerjackalope


Easy Bluetooth Enabled Door Lock With Arduino + Android by Collin Amedee

Knock Block by jkestner

turn signal biking jacket

by **leahbuechley** on June 21, 2008

Intro: Turn signal biking jacket

This tutorial will show you how to build a jacket with turn signals that will let people know where you're headed when you're on your bike. We'll use conductive thread and sewable electronics so your jacket will be soft and wearable and washable when you're done. Enjoy!

A version of this tutorial is also on my website.

Image Notes
1. the first jacket I made

Step 1: Supplies

.
Get your supplies. You need:

-- LilyPad Arduino main board
-- FTDI connector
-- mini USB cable
-- LilyPad power supply
-- 16 LilyPad LEDs (note: these aren't available from SparkFun yet, but will be soon)
-- 2 push button switches
-- a spool of 4-ply conductive thread
-- a digital multimeter with a beeping continuity tester. This is the one I have.
-- a garment or a piece of fabric to work on
-- a needle or two, a fabric marker or piece of chalk, puffy fabric paint, a bottle of fabric glue, and a ruler
(Available at your local fabric shop or Joann Stores .)
-- a pair of scissors
-- double sided tape (optional)
-- a sewing machine (optional)

disclosure: I designed the LilyPad, so I'll make some $ if you buy one.

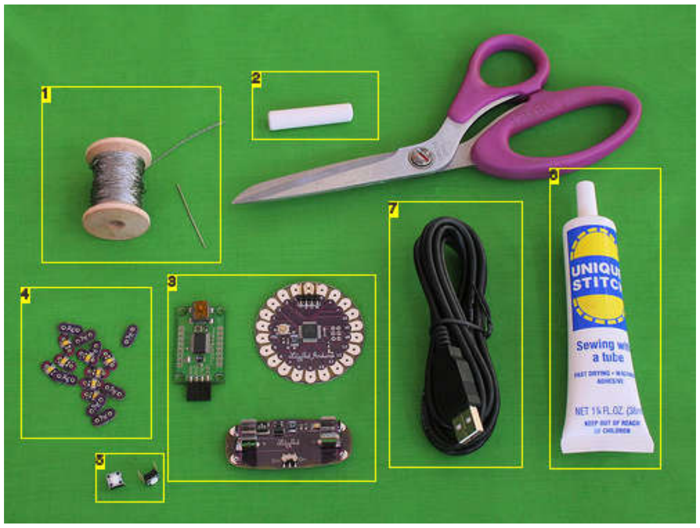


Image Notes
1. conductive thread and needle
2. chalk for drawing on fabric
3. LilyPad Arduino main board, power supply and USB link
4. LilyPad LEDs
5. switches
6. fabric glue
7. mini USB cable (like the one for your camera)

Step 2: Design

.
Plan the aesthetic and electrical layout of your piece.

Decide where each component is going to go and figure out how you will sew them together with as few thread crossings as possible. Make a sketch of your design that you can refer to as you work. The photos below show the sketches for my jacket. Stitching for power (+) is shown in red, ground (-) in black, LEDs in green, and switch inputs in purple.

Important note about the power supply

As you design, plan to keep your power supply and LilyPad main board close to each other. If they are too far apart, you are likely to have problems with your LilyPad resetting or just not working at all.

Why? Conductive thread has non-trivial resistance. (The 4-ply silver-coated thread from SparkFun that comes with the LilyPad starter kit has about 14 ohms/foot.) Depending on what modules you're using in your construction, your LilyPad can draw up to 50 milliamps (mA) of current, or .05 Amps. Ohm's law says that the voltage drop across a conductive material--the amount of voltage that you lose as electricity moves through the material--is equal to the resistance of the conductive material times the amount of current that is flowing through it.

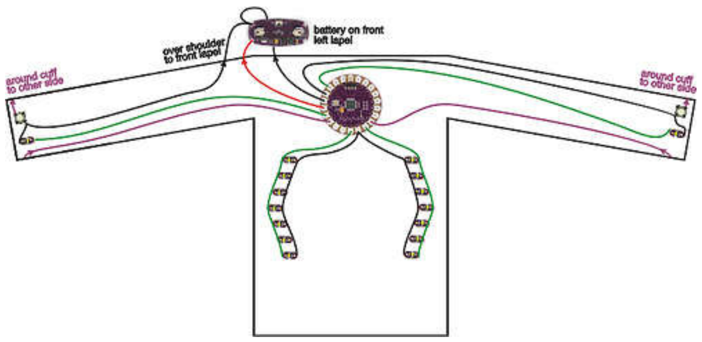
For example, if your LilyPad is a foot away from the power supply, the total resistance of the conductive material that attaches your LilyPad to your power supply is about 28 ohms. (14 Ohms in the conductive thread that leads from the negative terminal of the power supply to the negative petal on the LilyPad and 14 Ohms in the conductive thread that ties the positive terminals together). This means we can expect a drop of 1.4 Volts (28 Ohms * .05 Amps.) This means that while 5 Volts is coming out of the power supply, the LilyPad will only be getting 3.6 Volts (5 Volts - 1.4 Volts). Once the voltage at the LilyPad drops below about 3.3 Volts, it will reset. The resistance of the traces from + on the power supply to + on the LilyPad and - on the power supply to - on the LilyPad should be at most 10 Ohms. Plan the distance accordingly.

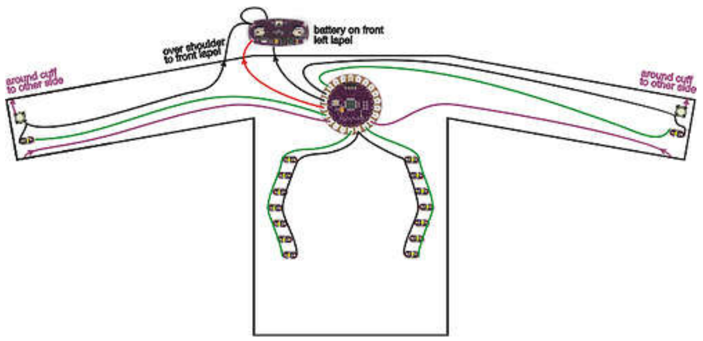
If all of this was confusing, don't worry! Just keep the LilyPad and power supply close to each other in your design.

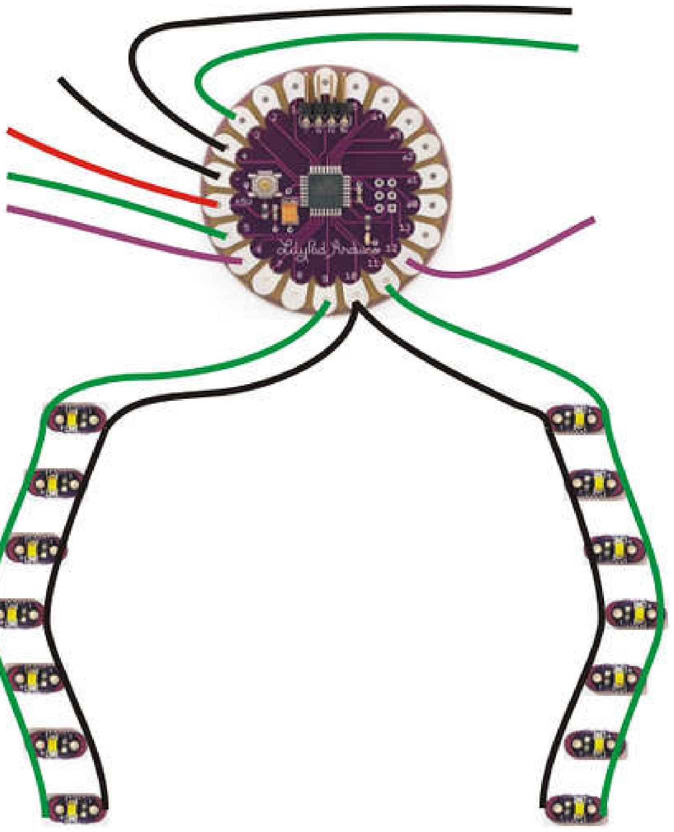
Transfer the sketch to your garment.

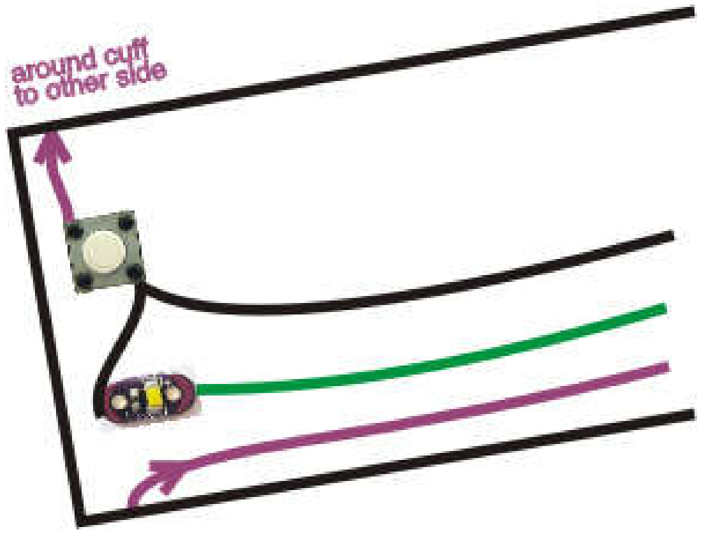
Use chalk or some other non-permanent marker to transfer your design to the garment. If you want, use a ruler to make sure everything is straight and symmetrical.

Use double sided tape to temporarily attach LIlyPad pieces to your garment. This will give you a good sense of what your final piece will look like. It will also keep everything in place and, as long as the tape sticks, make your sewing easier.


around cuff to other side
over shoulder to front lapel
battery on front left lapel
around cuff to other side


battery on front left lapel
over shoulder to front lapel
around cuff to other side
around cuff to other side




around cuff to other side

Step 3: Sew your power supply and LilyPad to your jacket

.

First, trim the leads off of the back of the power supply

Get out your LilyPad power supply piece and trim the metal parts that are sticking out the back of it. Small clippers like the ones shown in the photo work well, but you can also use scissors.

Stabilize your battery on the fabric.

Generally, you want to do everything you can to keep the power supply from moving around on the fabric. I recommend gluing or sewing the battery down before starting on the rest of the project. You may also want to glue or sew something underneath the power supply to help prevent it from pulling on the fabric and bouncing around as you move.

If you are working on a thin or stretch piece of fabric--first of all, reconsider this choice! It's much easier to work on a heavy piece of non-stretchy fabric. If you are determined to forge ahead with a delicate fabric, choose the location for your power supply wisely. It's the heaviest electronic module, so put it somewhere where it will not distort the fabric too badly. definitely glue or sew something underneath the power supply.

Sew the + petal of the power supply down to your garment.

If you are new to sewing, check out this great introduction before you start for info on how to thread a needle, tie knots and make stitches. Cut a 3-4 foot length of conductive thread. Thread your needle, pulling enough of the thread through the needle that it will not fall out easily. Tie a knot at the end of the longer length of thread. Do not cut the thread too close to the knot or it will quickly unravel.

Coming from the back of the fabric to the front, poke the needle into the fabric right next to the + petal on the power supply and then, from the front of the fabric, pull it through. The knot at the end of the thread will keep the thread from pulling out of the fabric. Now make a stitch going into the hole in the hole in the + petal on the power supply. Do this several more times, looping around from the back of the fabric to the front, going through the + petal each time.

Pay special attention to this stitching. It is the most important connection that you'll sew in your project. You want to make sure you get excellent contact between the petals on the power supply and your conductive thread. Go through the hole several times (at least 5) with your stitching. Keep sewing until you can't get your needle through anymore. Do not cut your thread, just proceed to the next step.

Sew from the battery to the LilyPad.

Once you've sewn the + petal of the battery down, make small neat stitches to the + petal of your LilyPad. I used a jacket with a fleece lining and stitched only through the inner fleece lining so that no stitches were visible on the outside of the jacket.

Sew the + petal of your LilyPad down, finishing the connection.

When you reach the LilyPad, sew the + petal down to the fabric with the conductive thread. Just like you were with the battery petal, you want to be extra careful to get a robust connection here. This stitching is making the electrical connection between your power supply and LilyPad.

When you are done with this attachment, sew away from the LilyPad about an inch along your stitching, tie a knot, and cut your thread about an inch away from the knot so that your knot won't come untied.

Put fabric glue on each of your knots to keep them from unraveling.

Once the glue dries, trim the thread close to each knot.

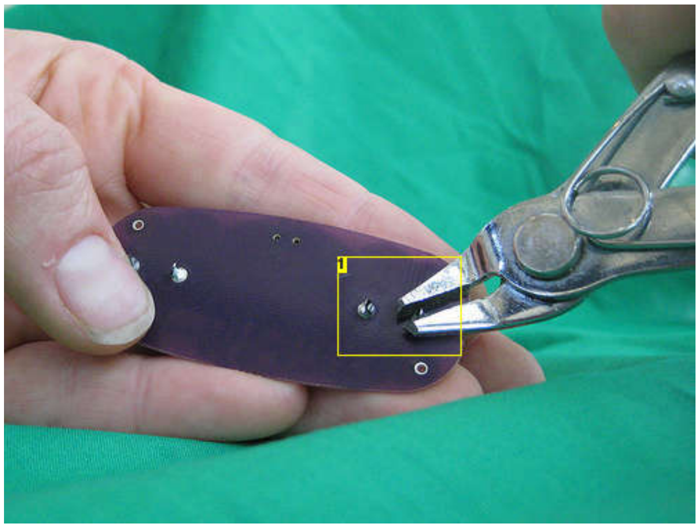
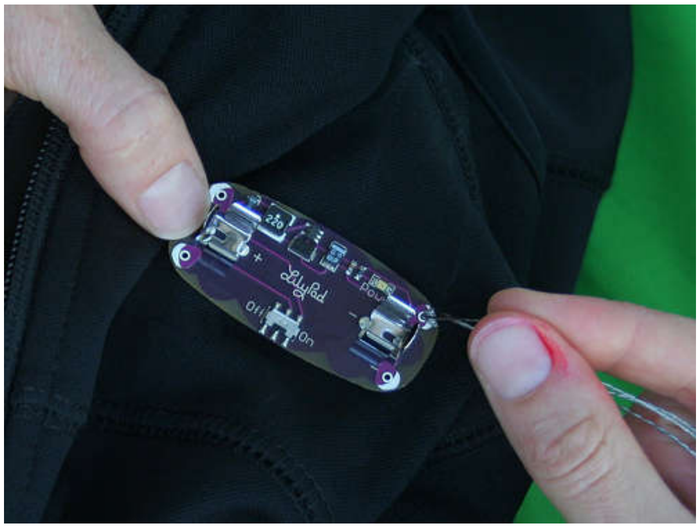


Image Notes
1. trimming the battery posts off the power supply.



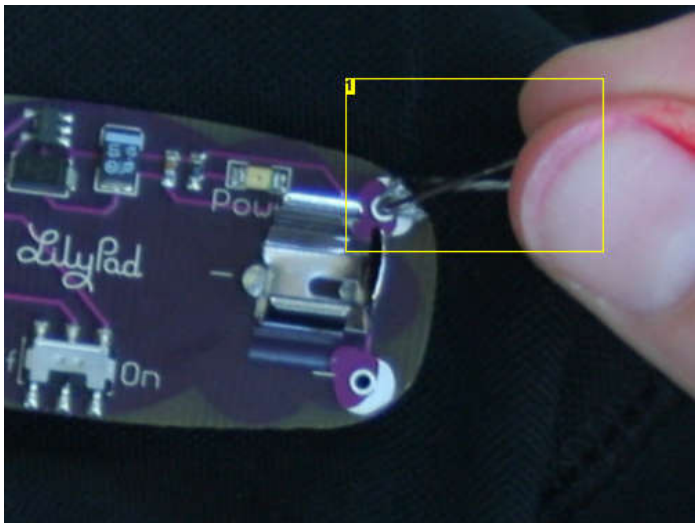
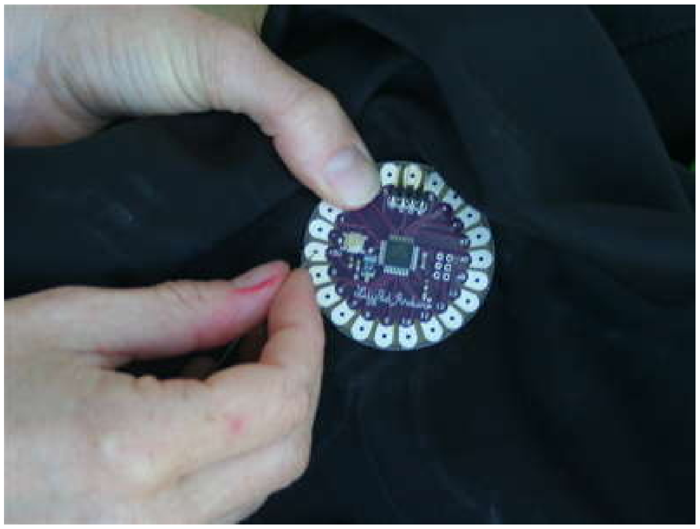
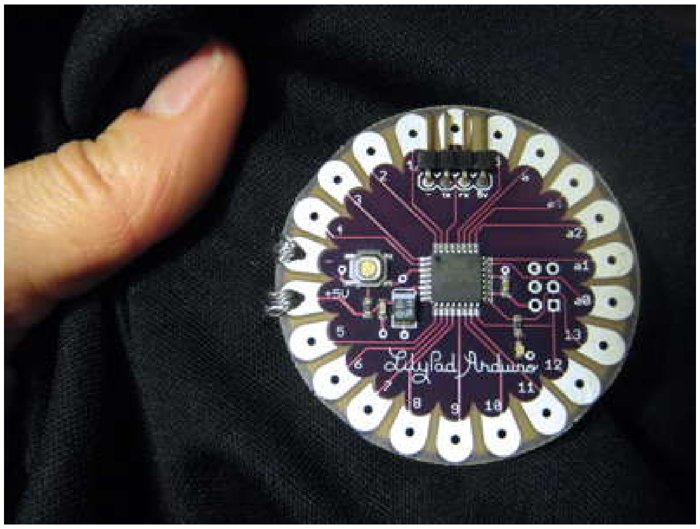


Image Notes
1. sewing on the + petal of the power supply. notice how I'm sewing through the hole from the front instead of the back, which is much harder.





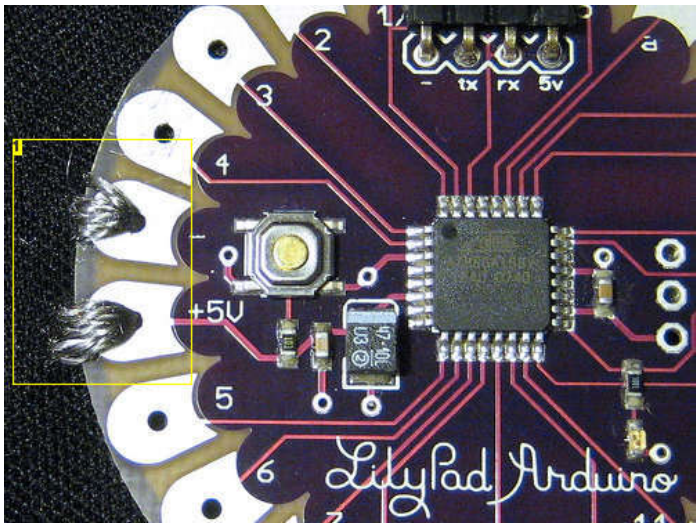
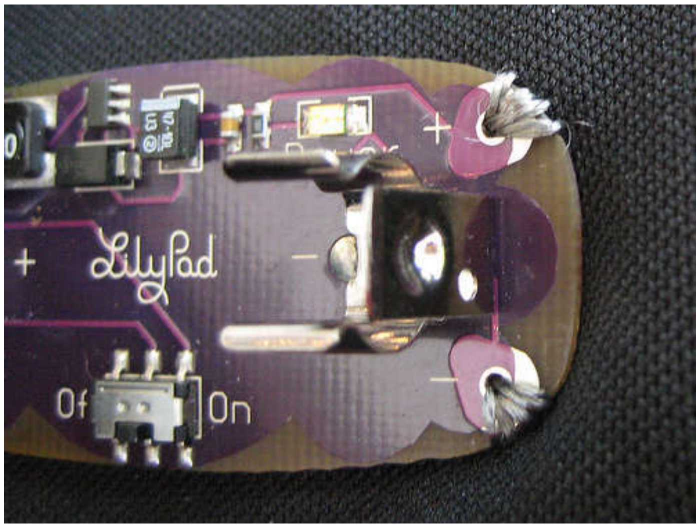


Image Notes
1. notice how dense my stitching is here. this is what your stitches should look like.



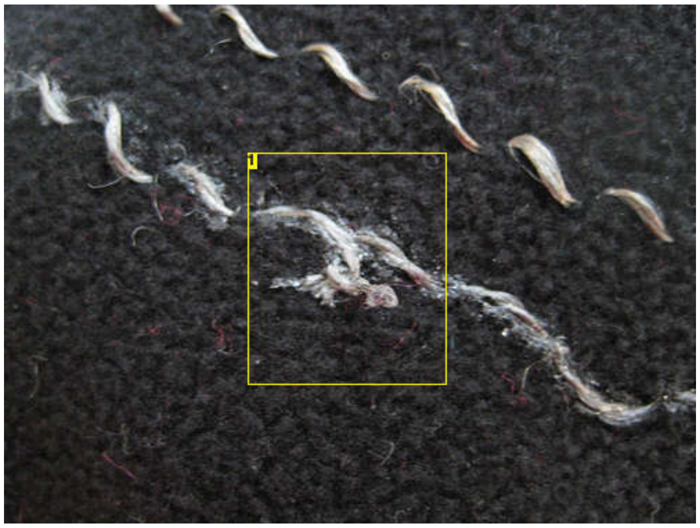


Image Notes
1. a glued and trimmed knot. knots without glue will come unraveled quickly.

Step 4: Test your stitching

.

Measure the resistance of your stitching.

Get out your multimeter and put it on the resistance measuring setting. Measure from power supply + to LilyPad + and power supply - to LilyPad -. If the resistance of either of these traces is greater than 10 ohms, reinforce your stitching with more conductive thread. If you're not sure how to measure resistance, check out this tutorial .

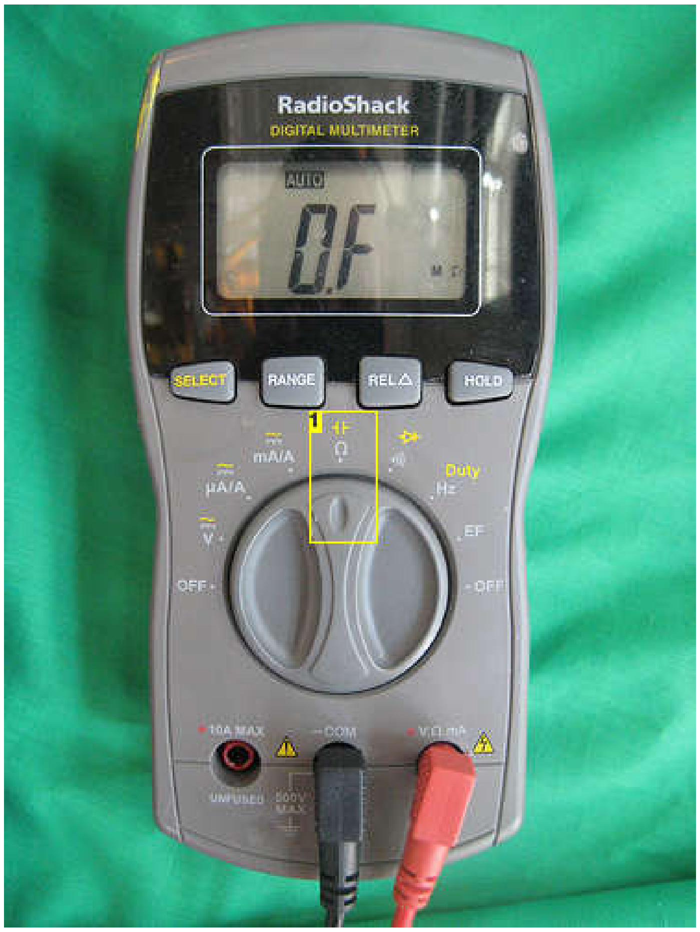
Put a AAA battery into the power supply and flip the power supply switch to the on position. The red light on the power supply should turn on. If it doesn't and you're sure you flipped the switch, quickly remove the battery and check for a short between your + and - stitches. (Most likely there is a piece of thread that's touching both the - and + stitching somewhere.) You can test for a short between + and - by using the beeping continuity tester on your multimeter. See this tutorial for information on how to use the continuity tester.

Also check the resistance between the + and - stitching. If the resistance is less than 10K Ohms or so, you've got a mini-short (probably a fine conductive thread hair that is touching both + and -) that you need to find and correct.

If the power supply does turn on, look at your LilyPad. It should blink quickly each time you press its switch. Once these connections are working properly, turn off the power supply and remove the battery.

Insulate your power and ground stitching

So, your jacket is now full of uninsulated conductive stitches. This is fine when a body is inside of it. A body will prevent sewn traces from contacting each other. But when the jacket is off of a person and you bend or fold it, traces will touch each other and short out. To fix this problem, cover your traces with puffy fabric paint (or another insulator like a satin stitch in regular thread). But, you don't want to cover traces until you're sure that everything works! So, use good judgment in when to coat traces.



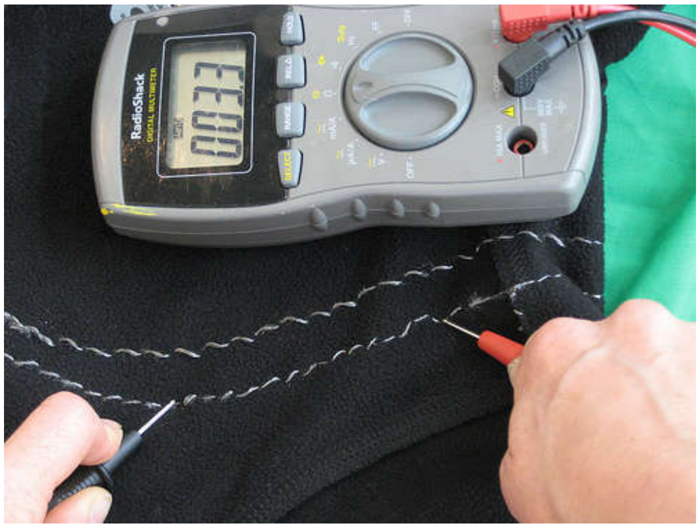


Image Notes
1. this is the resistance measuring setting

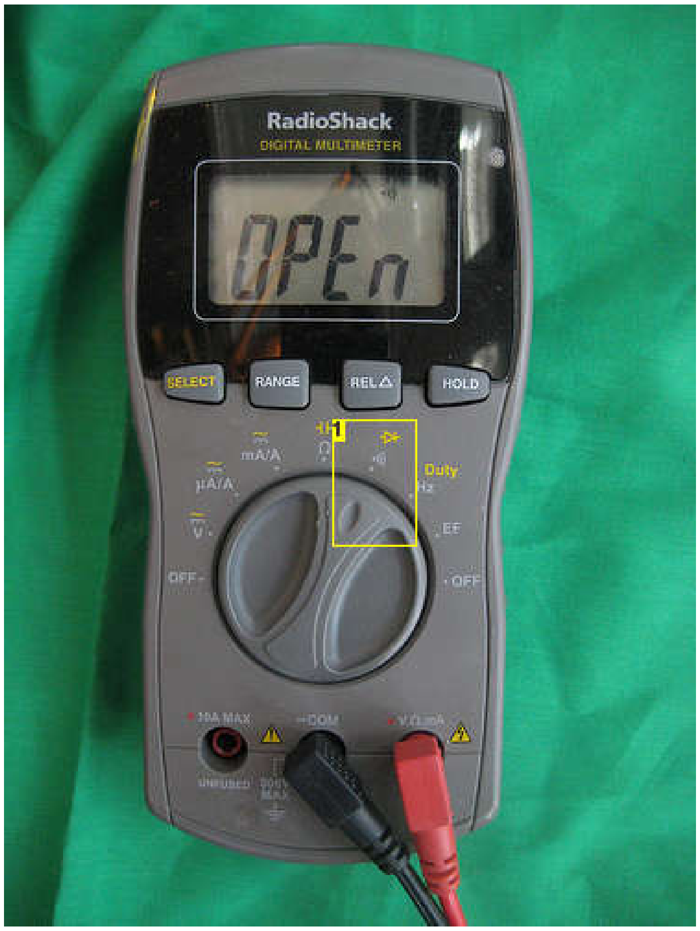


Image Notes
1. this is the continuity testing setting

Step 5: Sew on your turn signal LEDs

.
Sew in your left and right signals.

Using the same techniques you used to sew the power supply to the LilyPad, attach all of the + petals of the lights for the left turn signal together and to a petal on the LilyPad (petal 9 for me) and all of the + petals for the right signal together and to another LilyPad petal (11 for me). Attach all of the - petals of the lights together and then to either the - petal on the LilyPad or another LilyPad petal (petal 10 for me). Refer back to my design sketches if any of this is confusing.

Remember to seal each of your knots with fabric glue to keep them from unraveling. Be careful to avoid shorts; don't let one sewn trace touch another. In this case, the - traces for the LEDs are all connected, but you want to make sure that the + traces for the left and right signals do not touch the - trace or each other.

Test your turn signals.

Load a program onto your LilyPad that blinks each turn signal to make sure all of your sewing is correct.

Note, if you don't know how to program the LilyPad, work through a few of these introductory tutorials before proceeding.

Here's my test program:

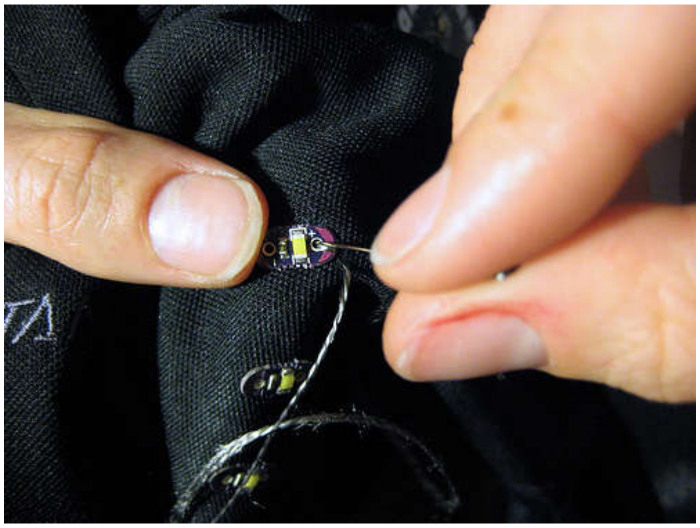
```
int ledPin = 13; // the LED on the LilyPad
int leftSignal = 9; // my left turn signal is attached to petal 9
int rightSignal = 11; // my right turn signal is attached to petal 11
int signalLow = 10; // the - sides of my signals are attached to petal 10

void setup()
{
pinMode(ledPin, OUTPUT); // sets the ledPin to be an output
pinMode(leftSignal, OUTPUT); // sets the leftSignal petal to be an output
pinMode(rightSignal, OUTPUT); // sets the rightSignal petal to be an output
pinMode(signalLow, OUTPUT); // sets the signalLow petal to be an output
digitalWrite(signalLow, LOW); // sets the signalLOW petal to LOW (-)
}

void loop() // run over and over again
{
delay(1000); // wait for 1 second
digitalWrite(leftSignal, LOW); // turn the left signal off
delay(1000); // wait for 1 second
digitalWrite(rightSignal, HIGH); // turn the right signal on
delay(1000); // wait for 1 second
digitalWrite(rightSignal, LOW); // turn the right signal off
delay(1000); // wait for 1 second
}
```

If your layout is the same as mine, you can just copy and paste this program into your Arduino window.

If your turn signals don't work, use your multimeter (and the instructions from the last step) to test for shorts or bad connections and make sure that your program matches your physical layout.

insulate your turn signal stitches

Cover your traces with puffy fabric paint. Remember, you don't want to cover traces until you're sure that everything works! Use good judgment in when to coat traces.



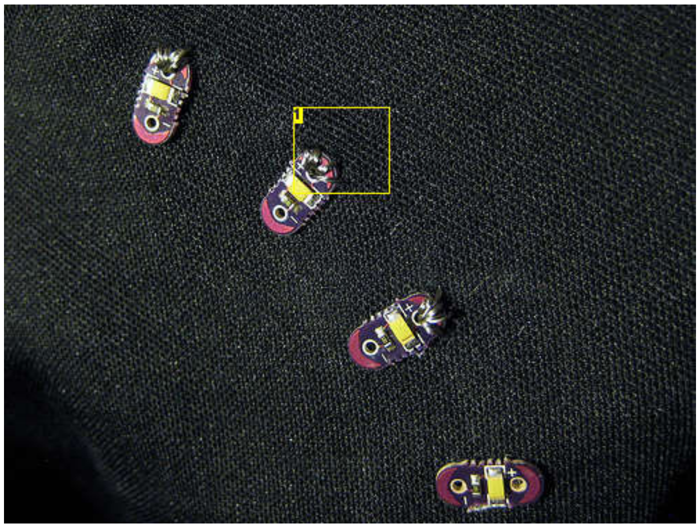


Image Notes
1. stitching in process, outside view: 3 + petals are sewn together

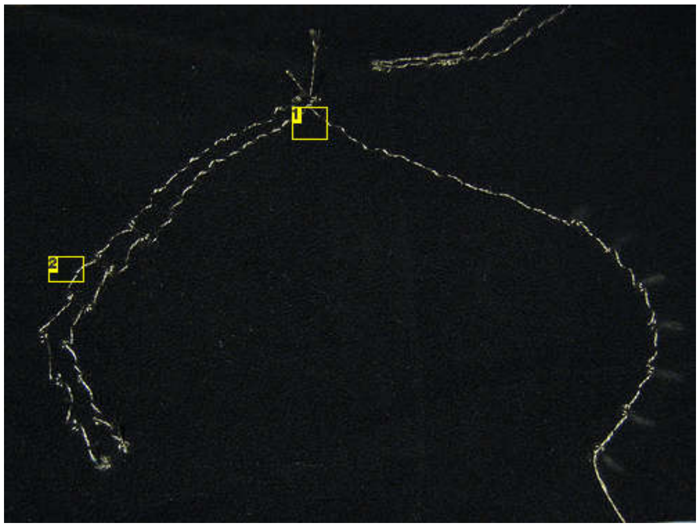


Image Notes
1. these 2 middle traces are the negative (-) traces for all of my turn signal LEDs. these traces are attached to petal 10 on my LilyPad
2. this is the stitching for the positive (+) leads of my right turn signal LEDs. (Since this is an inside view, everything is reversed.) These traces lead to petal 11 on my LilyPad.

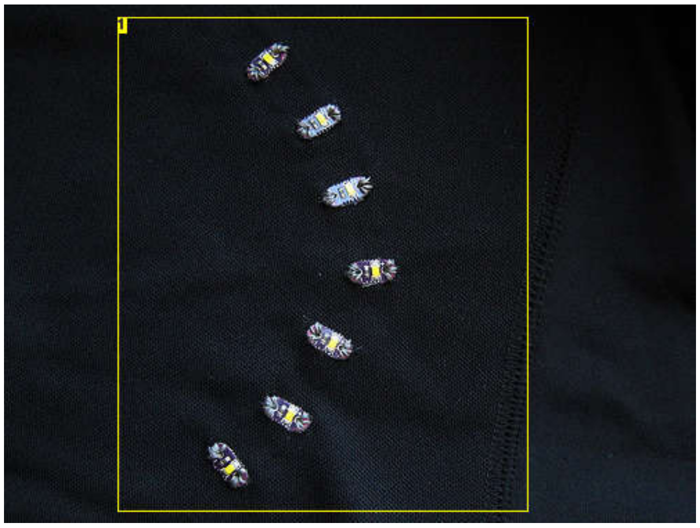


Image Notes
1. my finished right turn signal. notice how my stitching doesn't come through to the outside of the garment.

Step 6: Sew in your control switches

.

Place your switches

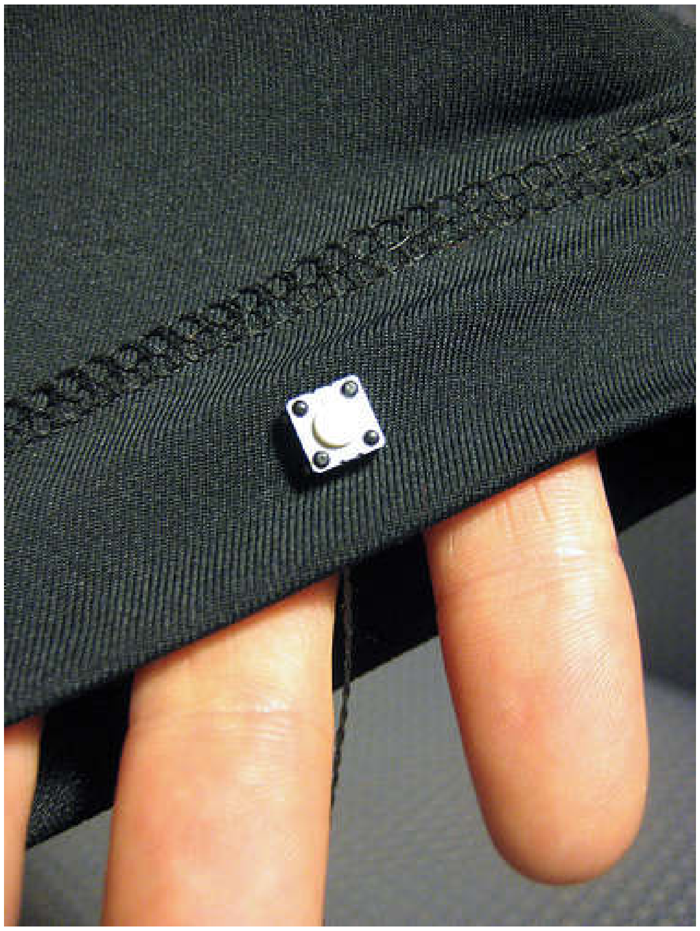
Find a spot for your switches where they'll be easy to press when you're riding your bike. I mounted mine on the underside of my wrists. I found a good spot by trying out different places. Check out the photos to see what I mean.

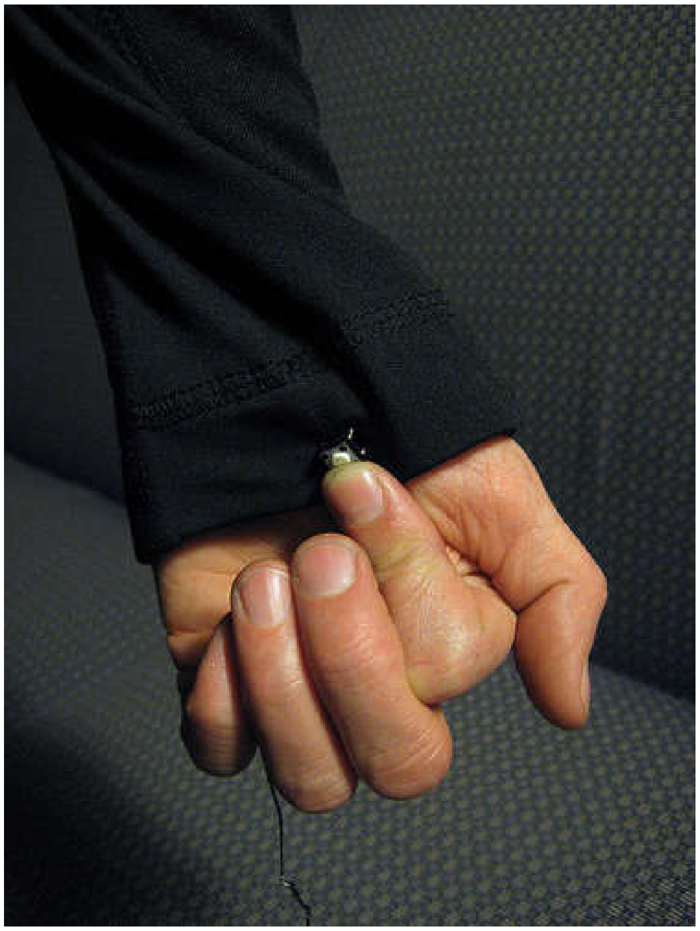
Once you've found a good position, push the legs of the switch through the fabric and bend them over on the inside of the fabric.

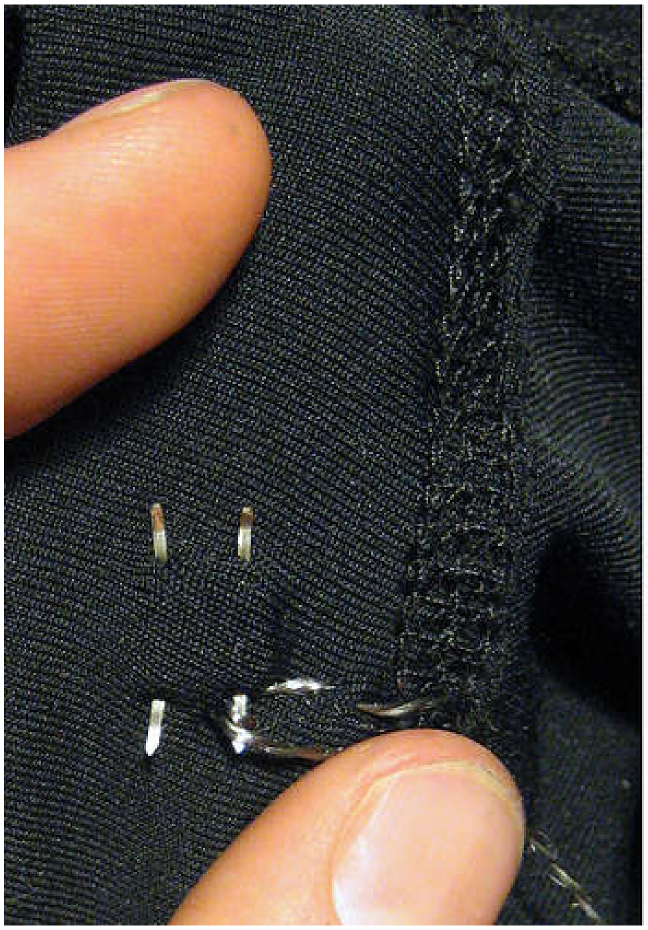
Sew in your switches.

Sew your switches into the garment. Sew 1 leg to the switch input petal on the LilyPad and another leg, one that is diagonally across from the first , to ground or another LilyPad petal. I used petal 6 for the switch input on the left side and petal 12 for switch input on the right side. I used - for the - connection on the left side, but petal 4 for the - connection on the right side. Refer back to my design drawings if any of this is confusing.

When you're done sewing, go back and reinforce the switch connections with glue. You don't want your switches to fall out of their stitching.







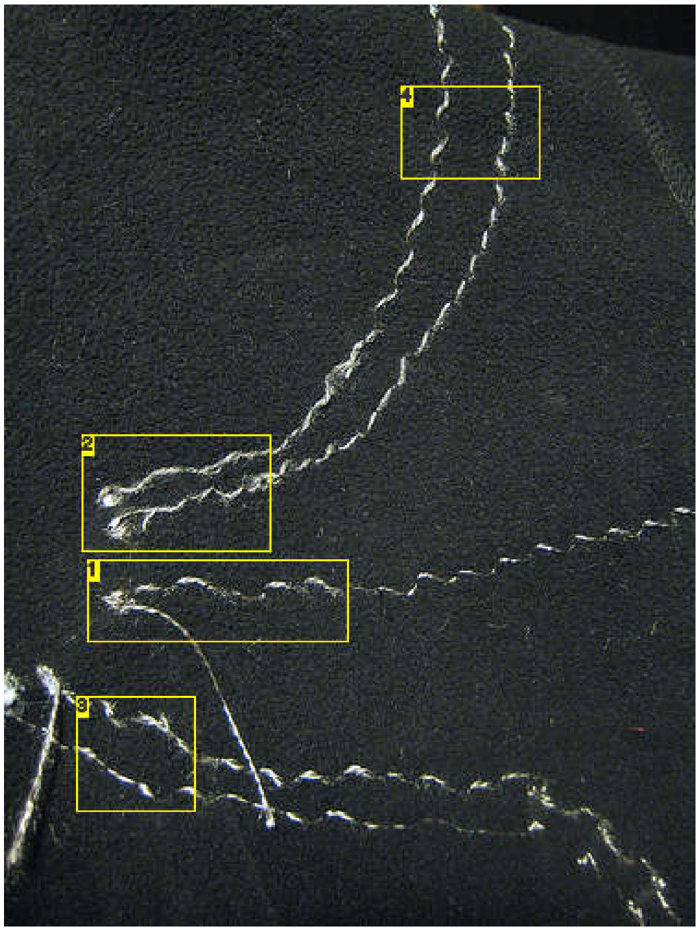
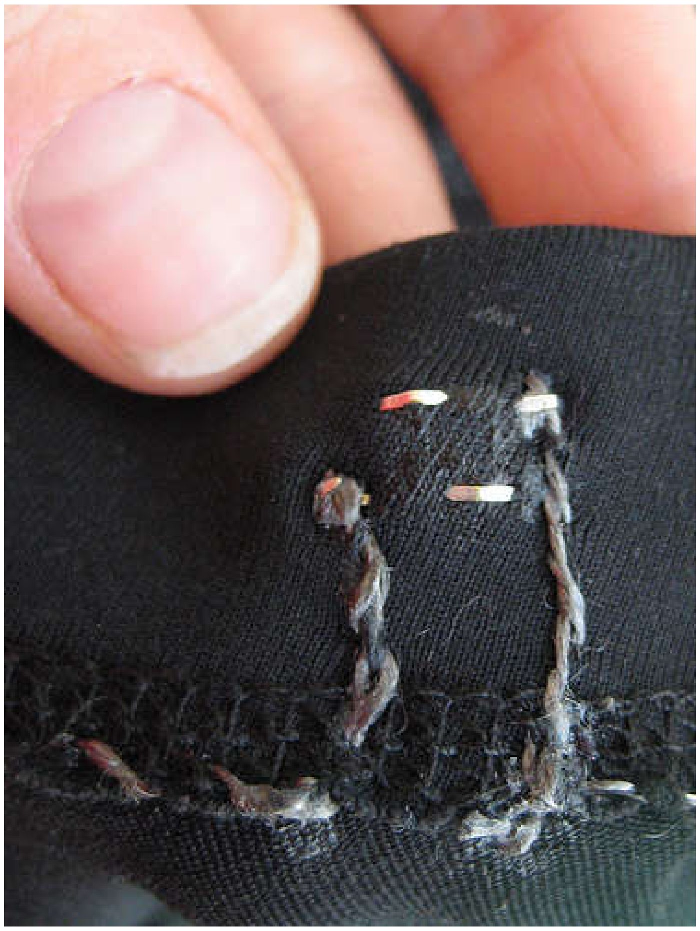


Image Notes
1. the first trace from my left switch is finished. this is the switch input trace that is tied to petal 6 on the LilyPad. I just have to glue and trim the knot.
2. these are the stitches that lead from the power supply to the LilyPad.
3. these are my left turn signal stitches. I have a knot to glue and trim on these too.
4. you might have noticed that I didn't insulate my traces. you too can leave them uninsulated, but be aware of shorts from folding/bending whenever you're not wearing the jacket! especially when you are programming and troubleshooting.



Step 7: Sew in your indicator LEDs

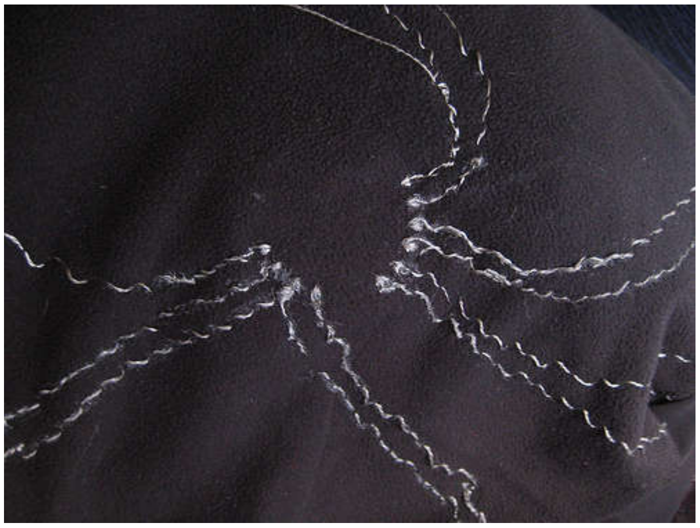
.

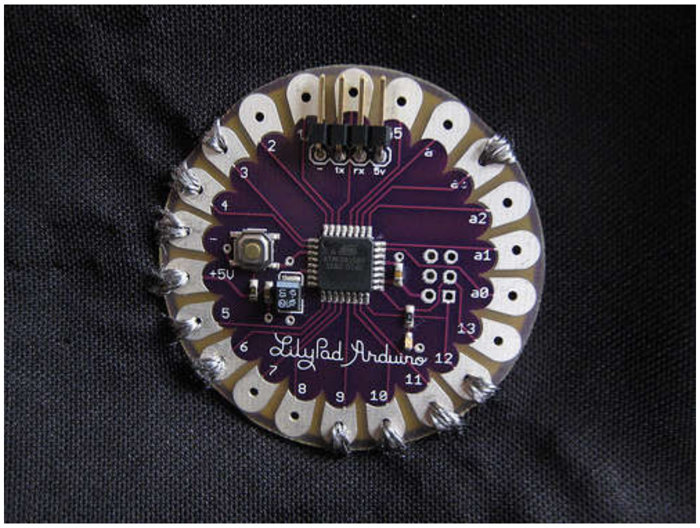
Sew a single LED onto the sleeve of each arm.

These will give you essential feedback about which turn signal is on. They'll flash to tell you what the back of your jacket is doing, so make sure they're in a visible spot. Sew the + petals of each LED to a LilyPad petal and the - petals of each LED to the - side of the switch (the - trace you sewed in the last step). I used petal 5 for the LED + on the left side and petal 3 for the LED + on the right side. Again, refer back to my design drawings if any of this is confusing.

As always, remember to glue and trim knots and be careful not to create any shorts.

Once you sew both wrist LEDs, you're done with the sewing phase of the project! Now, on to programming...





Step 8: Program your jacket

.

Decide on the behavior you want.

I wanted the left switch to turn on the left turn signal for 15 seconds or so, and the right switch to do the same thing for the right signal. Pressing a switch when the corresponding turn signal is on should turn the signal off. Pressing both switches at the same time should put the jacket into nighttime flashing mode. The wrist mounted LEDs should provide feedback about the current state of the jacket. Here's the code I wrote to get that behavior.

Program your jacket

To program your garment, copy and paste my code into an Arduino window and load it onto the LilyPad. You may have to make some small adjustments first depending on where you attached lights and switches. Play with delays to customize your blinking patterns. Follow my LilyPad introduction instructions if you need more information on how to program the LilyPad or how to make sense of my code.

Plug your battery back in and see if it works and...go biking!

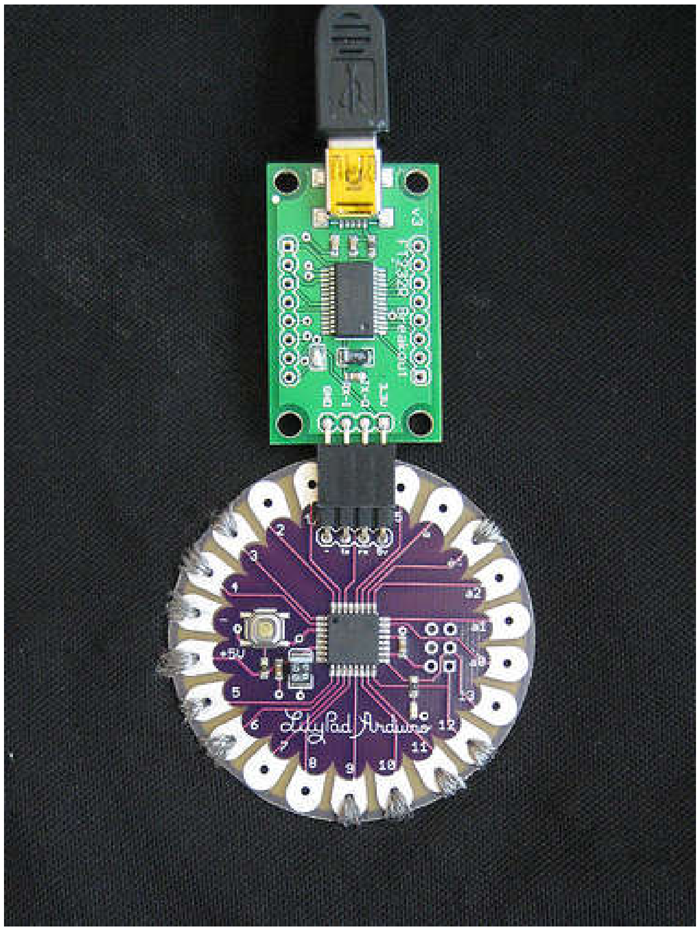
Insulate the rest of your traces

Cover the rest of your traces with puffy fabric paint. Again, don't coat anything until you're sure it works.

About washing

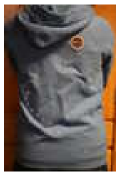
Your creation is washable. Remove the battery and wash the garment by hand with a gentle detergent.

Note: silver coated threads will corrode over time and their resistance will gradually increase with washing and wear. To limit the effects of corrosion, insulate and protect your traces with puffy fabric paint or some other insulator. You can also revive exposed corroded traces with silver polish. Try this on a non-visible area first to see what it does to your fabric!


FT232R Breakout
v3
LilyPad Arduino
+5V

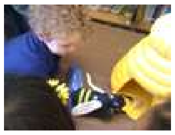
Related Instructables

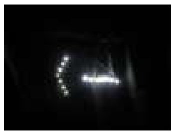
Programmable LilyPad EL-Wire Dress by quasiben


soundie: a musical touch-sensitive light-up hoodie by kanjun

EL Driver Board by quasiben

LilyPad Arduino Blinking Bike Safety Patch by bekathwia


Interactive Bee Game by quasiben


Latch-Modified Turn-Signal Jacket by quasiben

Light for life: Glowing button cycling jacket by kempton

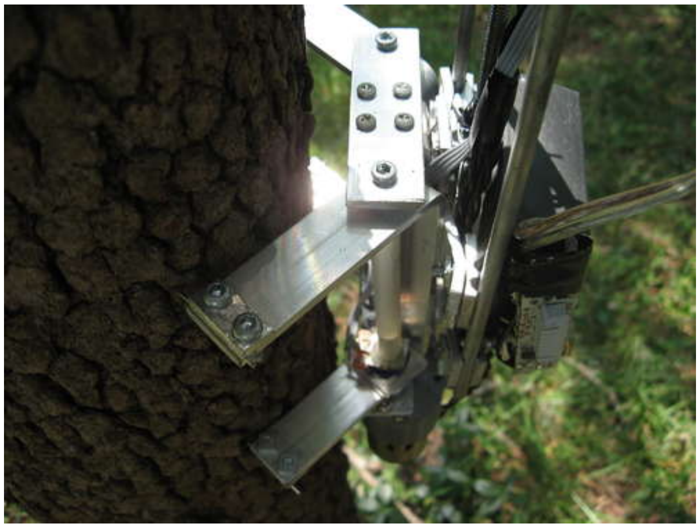
The Motivational Moody Workout T-Shirt by Lingon

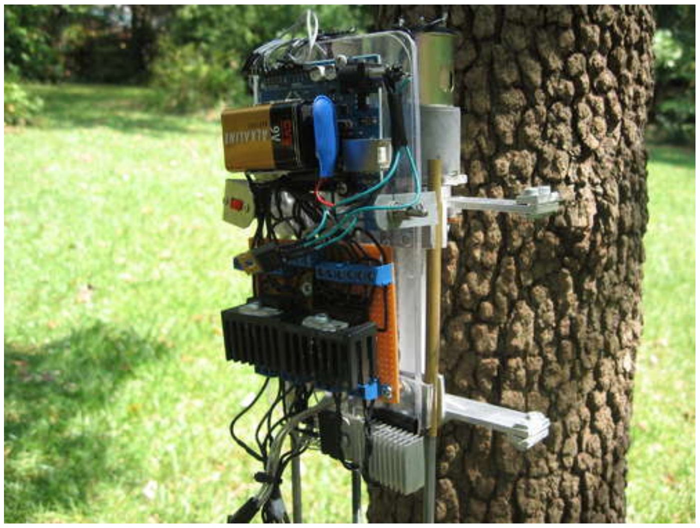
Tree Climbing Robot

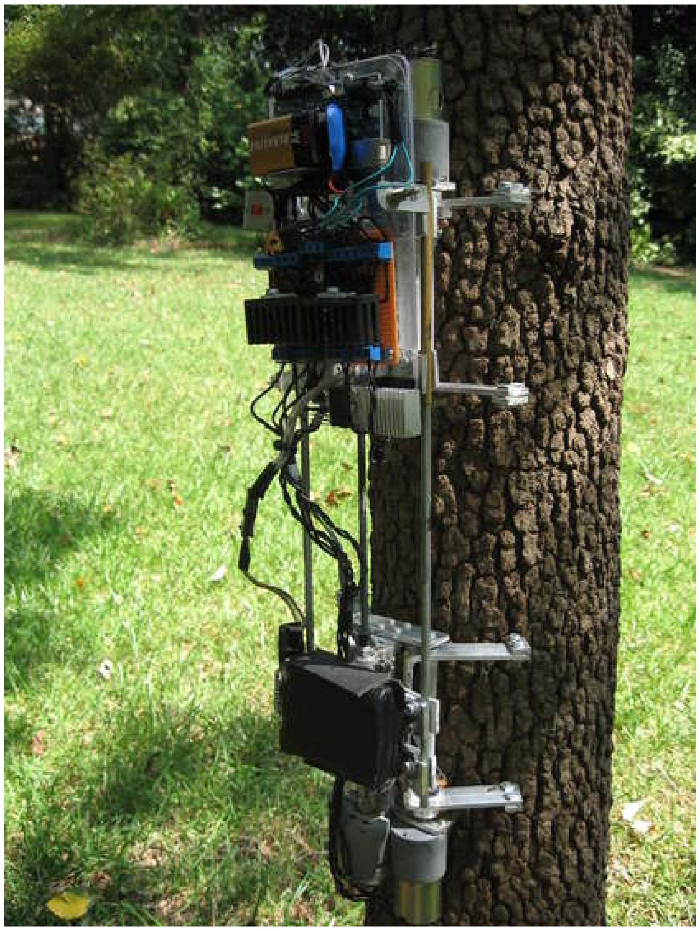
by Technochicken on August 17, 2011

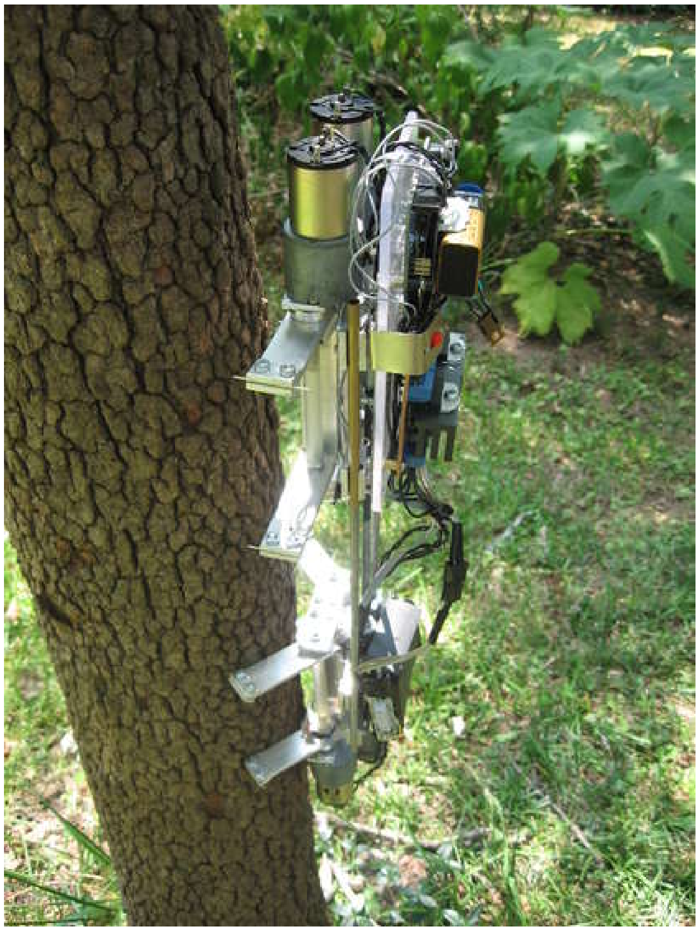
Intro: Tree Climbing Robot

After I got comfortable programming and building with an Arduino, I decided to build a robot. I did not have any particular type in mind, so I wracked my brain (and the internet) for cool robot ideas. Eventually, somehow the idea popped into my head to build a robot that could climb trees. At first I dismissed the idea as beyond my skill level, but after further thought, and some time in Sketchup, I decided to take a shot at the challenge. This is the result of my efforts.







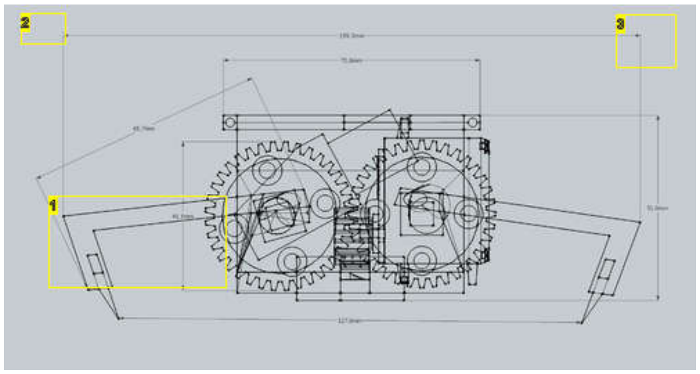


Step 1: Design

I started out by creating a basic design in Sketchup. The robot was to consist of two segments, joined by a spine which could be extended or retracted. Each segment would have four legs with very sharp points as feet. To climb, the legs on the top segment would pinch together and the sharp feet would dig into the bark, securing the robot. Then the spine would be retracted, pulling up the bottom segment. The legs on the bottom segment would then grip the tree, and the top segment would release. Finally, the spine would extend, pushing the top segment upwards, and the process would repeat. The climbing sequence is somewhat similar to the way an inchworm climbs.

In my original design (show in the images above), all four legs in each segment were controlled by one highly geared down motor. I decided to ditch this idea for a few reasons. Firstly, I could not find the type of spur gear needed to mesh the legs together. Also, with all the legs linked together, the robot would have a hard time gripping uneven surfaces. Finally, I decided that the robot would be much easier to build if the motors drove the legs directly.

The other significant change I made from my original design was the way the spine worked. In my model, I used a rack and pinion type gearing system to extend and contract the spine. However, I could not find the necessary parts to build such a system, so I ended up using a threaded rod coupled to a motor to actuate the spine.



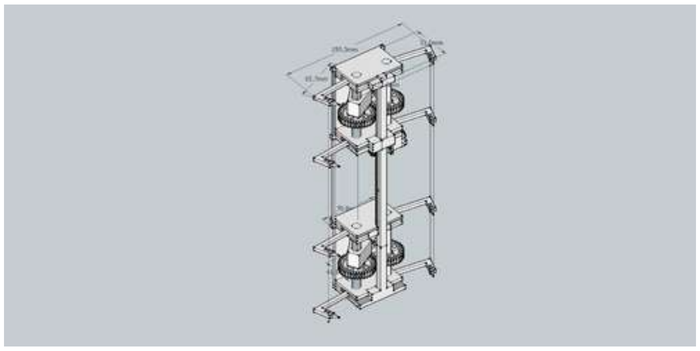


Image Notes
1. The strange leg shape came from resizing the legs all at once in Sketchup.
2. Front on

3. To make these types of images in Sketchup, turn X-ray on and turn perspective off.

Step 2: Tools and Materials

Microcontroller

- Arduino Uno (any will work)

Motor Controller

- 3X L298HN - these can be gotten for free as samples from ST
- 2.5" x 3.125" Perf Board
- Terminal Strips
- 22AWG Solid core wire
- 3X Aluminum heatsinks (I cut in half an old northbridge heatsink)
- Thermal paste

Power

- 9V Battery (to power the Arduino)
- Approximately 12V LiPo or Li-ion battery (I modified a laptop battery, so I did not even need to buy a charger)
- 5V regulator (To regulate power to the motor controller logic circutry)
- 9V Battery clip
- Barrel connector (Must fit the Arduino power connector)

Other Electronics

- 4X 7 RPM Gear Motor (These power two legs each)
- 4X Thin linear trim pots (Rotation sensors for the legs)
- DPDT Toggle switch (Power switch)
- SPDT Slide switch (User input)
- 2X Mini Snap Action Switch (Limit switch)
- 3 10K resistors (Pull down)
- Headers
- Signal Wire (Old IDE cables work really well, and let you organize your wires easily)
- Heat Shrink Tubing

Hardware

- 12' 3/4" x 1/8" Aluminum Bar (These come in 6' lengths at my local hardware store)
- 6" x 3" acrylic sheet (Electronics are mounted to this)
- 6x Standoffs with screws
- 1' Threaded rod and corresponding 1/2" nut
- 2X 1' x 3/16" steel rod
- 1' x 3/16" I.D. Brass Tubing
- 4X 5mm Aluminum Universal Mounting Hub
- Pack of large T Pins
- 4X 3/32 screws (to mount the motors)
- An assortment of 4/40 screws and nuts
- Assorted hex screws and nuts
- 4X Bic pens (I used the plastic shafts to fix the pots on the legs in place)
- 4X Locknuts
- 5 Minute epoxy
- Sheet metal scraps (For spacing and mounting things. Bits of Meccano work well)
- Stick on Velcro (For holding on the batteries)
- Hard Drive reading head bearing
- 3/4" Plastic angle
- Electrical Tape
- Zip Ties

Tools

- Electric Drill / Drill press (As well as a lot of bits)
- Hacksaw
- Soldering Iron
- Pliers
- Allen wrench
- Assorted screwdrivers
- Wire Strippers
- C Clamp (These can be used to make nice 90 degree bends in the aluminum)
- Ruler
- Files

Nonessential

- Bench PSU
- Multimeter
- Breadboard

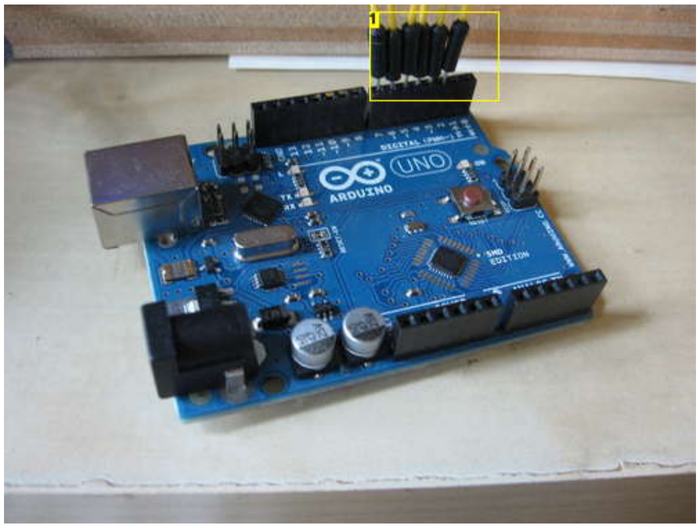


Image Notes
1. Still hooked up to my LED matrix

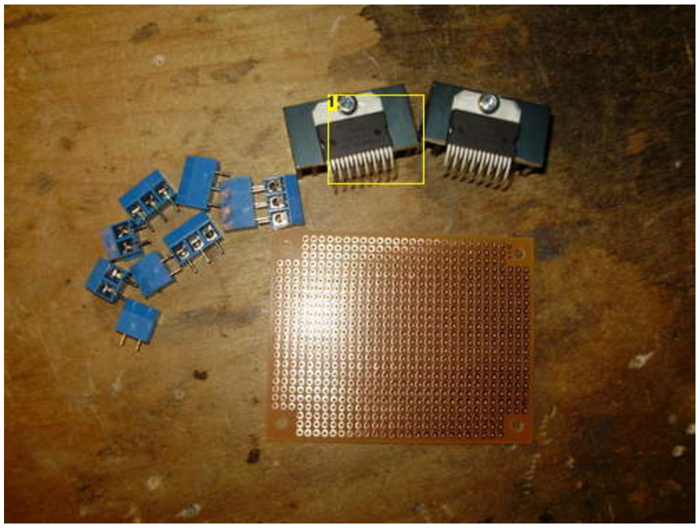


Image Notes
1. L298HN mounted to heatsink

Image Notes
1. 400:1 gearbox for lots of torque

Step 3: Motor Controller

The motor controller I built for this robot is based off the L298HN Dual Full Bridge chip. To use the chip, I followed the guide here . To start out, I placed all the components on a piece of perf board, to figure out the layout. With this chip, each motor requires three inputs to work: an enable signal and two input signals. The enable signal is used to control the motor speed with PWM, but since I did not need to control PWM, I just wired all the enable pins in parallel to a 5V line when I hooked the controller up to the Arduino. Once I figured out the layout, I soldered all the components in place, and made connections with 22AWG solid core wire. Finally, I spread some thermal paste on the back of the L298's, and screwed on the heatsinks. The particular heatsinks I used were made by cutting in half a northbridge heatsink from a computer motherboard, and drilling and tapping a hole for the screw. They are probably much larger than needed, but there is no harm in having over sized heatsinks. A higher resolution image of the labeled board can be found here .

When finished, this motor controller should be able to bidirectionally control 4 DC motors at up to 2A each (probably 2A continuously, because of the size of the heatsinks). As you may notice, this leaves me one motor short. My original design used a servo to actuate the spine, but I had to change my design to using a DC motor. To power it, I wired my third L298 chip to a molex connector (so I can disconnect the motor) and soldered on wires for all the connections. It does not look as pretty as my controller on a circuit board, but it works.

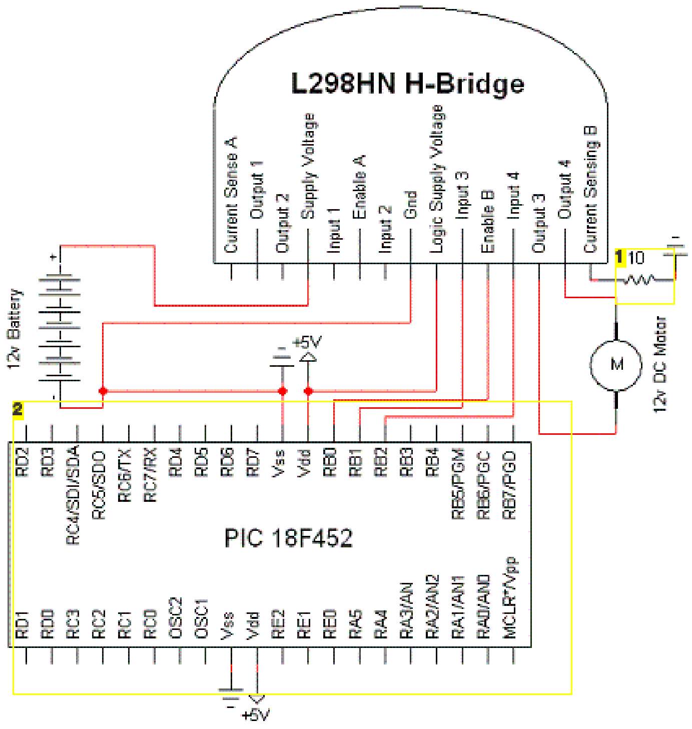


Image Notes
1. I removed this resistor, and wired the Current Sense directly to ground
2. Ignore this bit

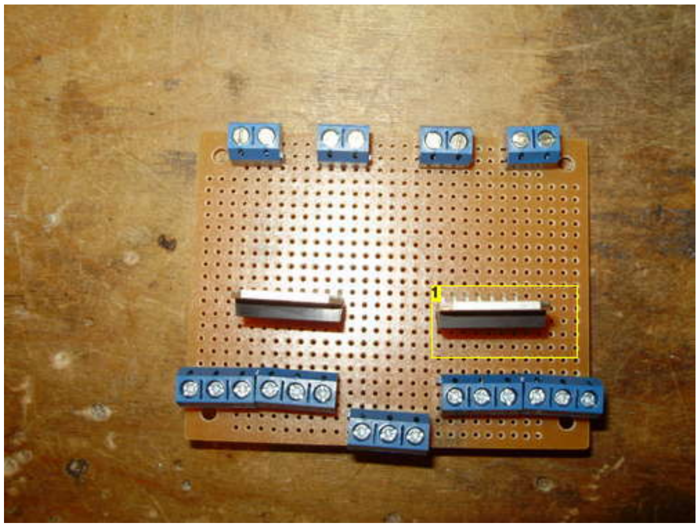


Image Notes
1. The pins on the L298HN must be manually bent for the chip to stand upright

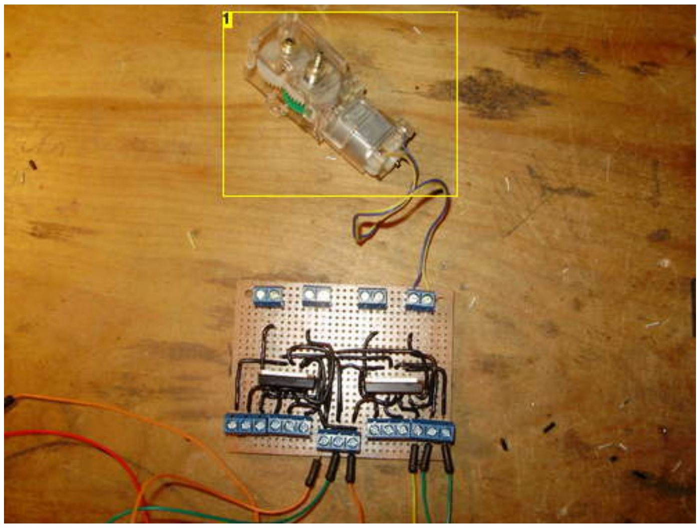
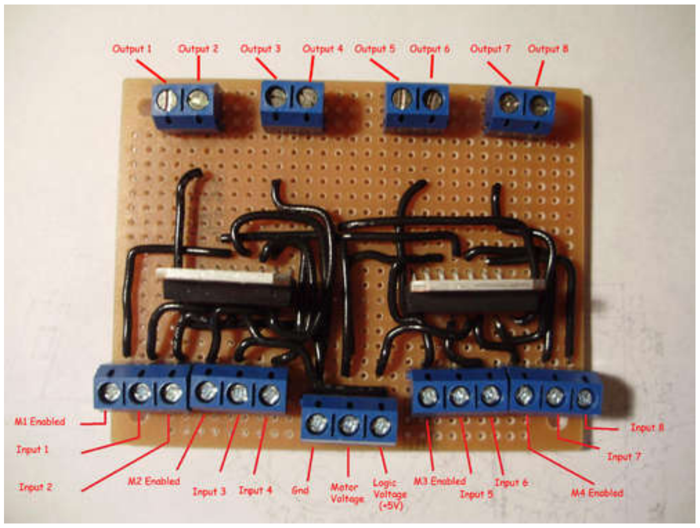
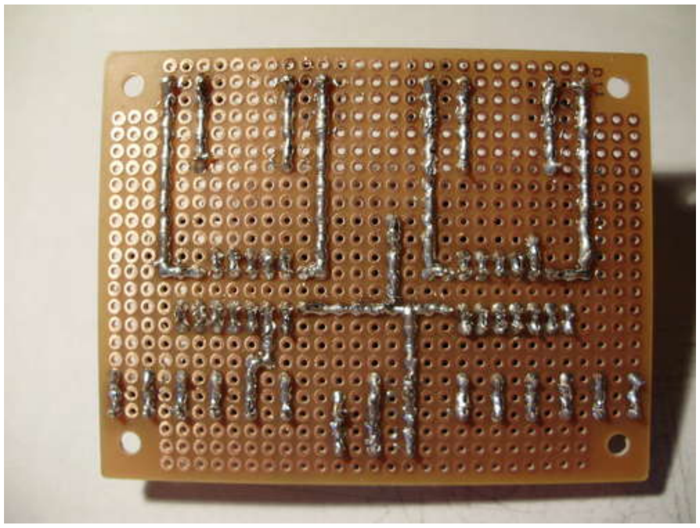
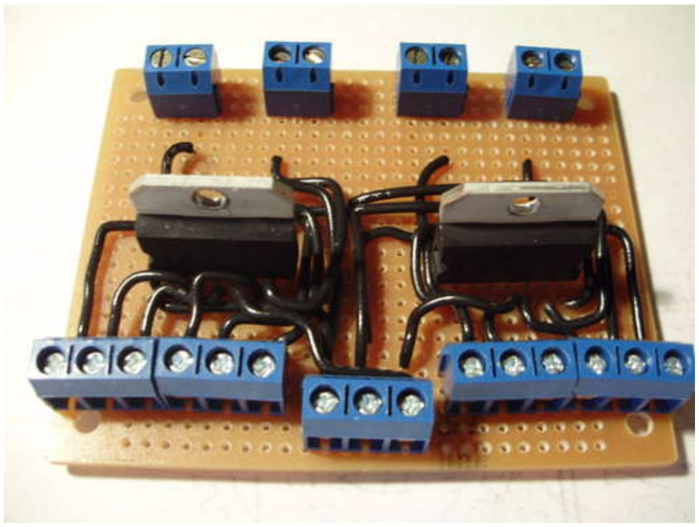
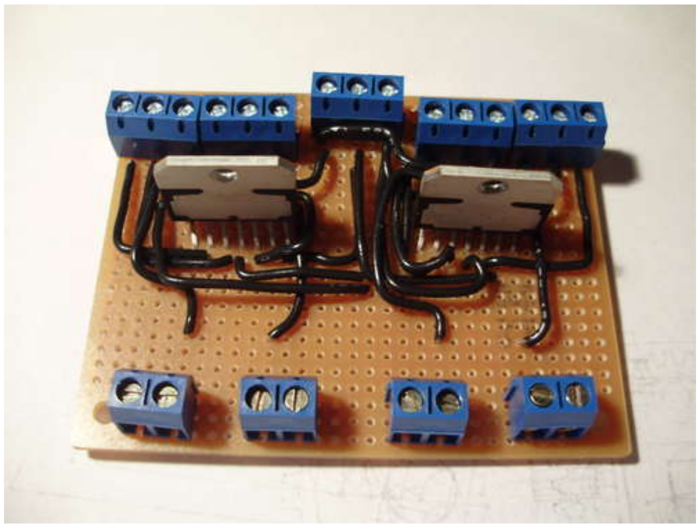
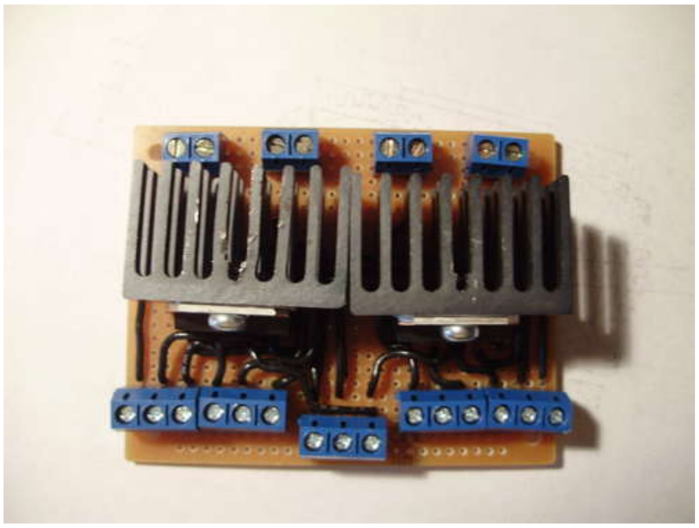
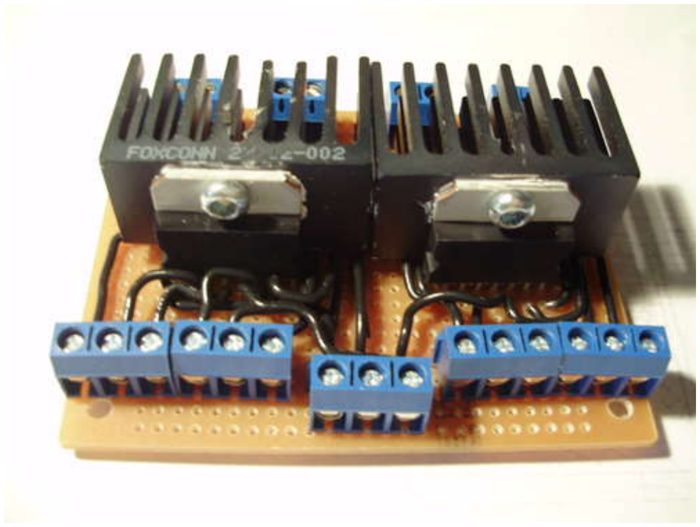
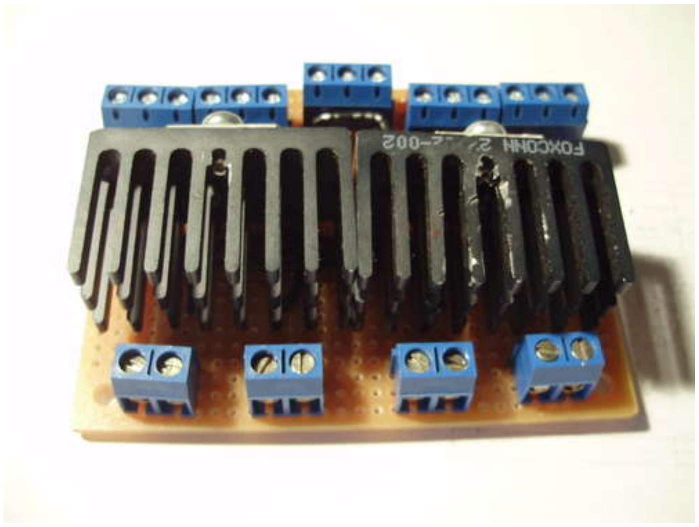
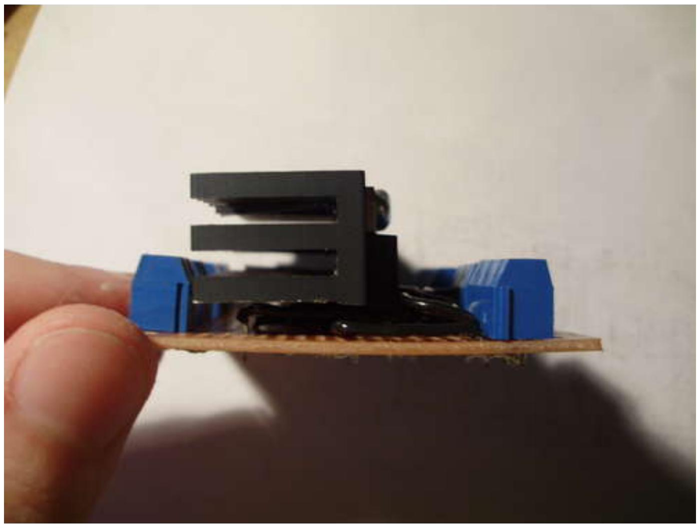


Image Notes
1. Testing with a cheap gearbox








FOXCONN

FOXCONN



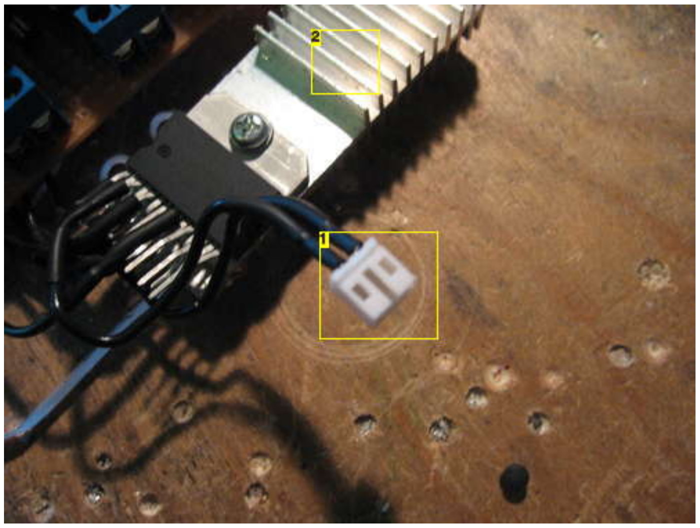


Image Notes
1. This goes to the 5th motor
2. This heatsink is part of one from an old ATX power supply.

Step 4: Power

The robot's power is supplied by two different sources. The Arduino and the motor controller logic circuitry are powered by a 9V battery, while the motors are powered by an approximately 12V Li-Ion battery pack.

I wanted to avoid having to buy an expensive LiPo/Li-Ion battery pack and charger, so I searched through my piles of electronic junk for a device with an appropriate battery. I settled on the battery from a 12" iBook laptop. The battery was 10.8V and 50Wh, but it was a little large and heavy for my needs. To fix this, I tore it open and had a look at the internals. I found that the battery was comprised of six 3.7 volt cells. These cells were organized in pairs of two wired in paralleled. The three pairs were then wired in a series, making a total 11.1V. To shrink the pack but keep the voltage, I simply removed one cell from each pair. The final battery pack had only half the capacity and half the discharge rate of the original (now only 2C), but the full voltage. I then wrapped the cells together with electrical tape so they would hold their shape, and soldered a quick-disconnect connector to the battery leads.

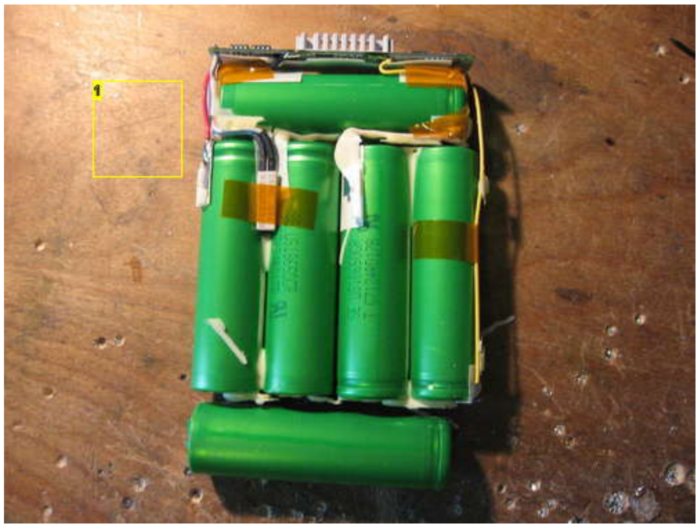


Image Notes
1. Casing removed

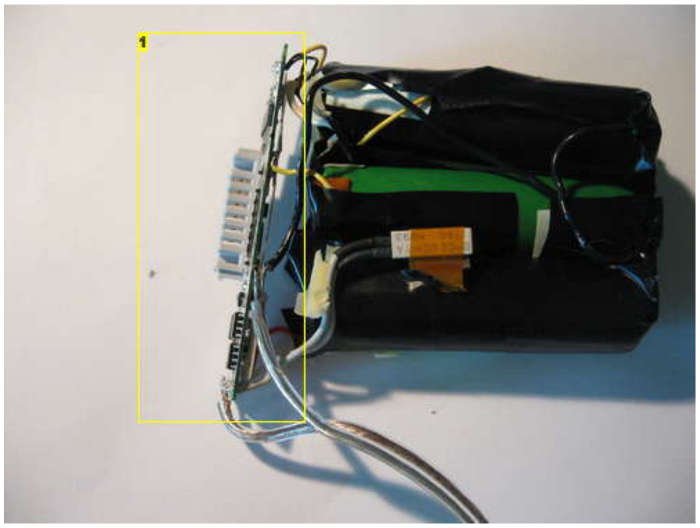


Image Notes
1. I left his bit so I could still charge the battery in the laptop

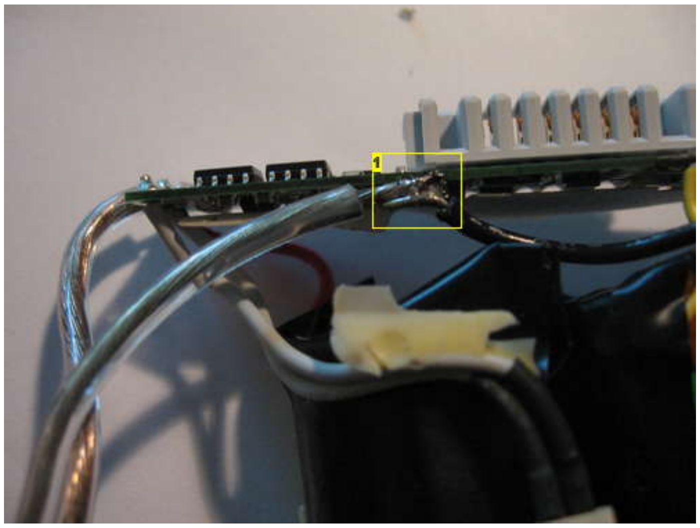


Image Notes
1. -

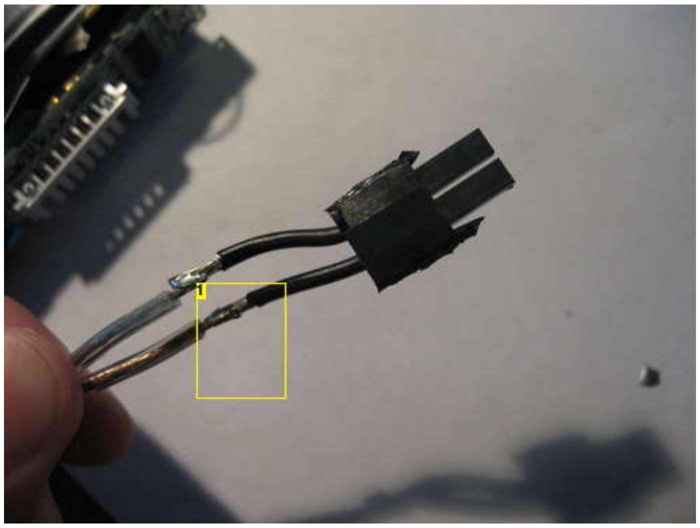
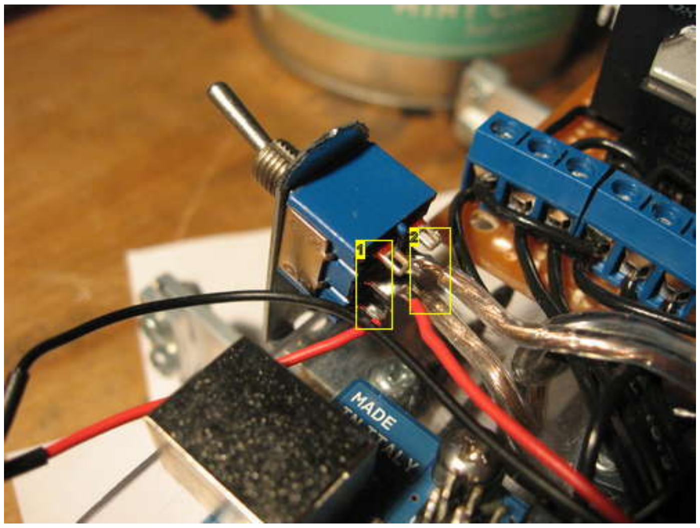


Image Notes
1. This was covered up later

Step 5: Power, cont.

The Arduino and the logic circuitry for the motor controller are both powered by a 9V battery. While the Arduino can take 9V input, the logic circuitry requires 5V, so I wired a 5V regulator in paralleled to the 9V going to the Arduino. Now, why did I not just take advantage of the Arduino's internal 5V regulator? Well, basically I ran out of pins, and I did not want to overdraw the Arduino. In addition to the regulator, I soldered a barrel power connector to the 9V end of the circuit, to fit into the Arduino. Finally, I added a DPDT toggle switch to break the 12V battery circuit as well as the 9V battery circuit.



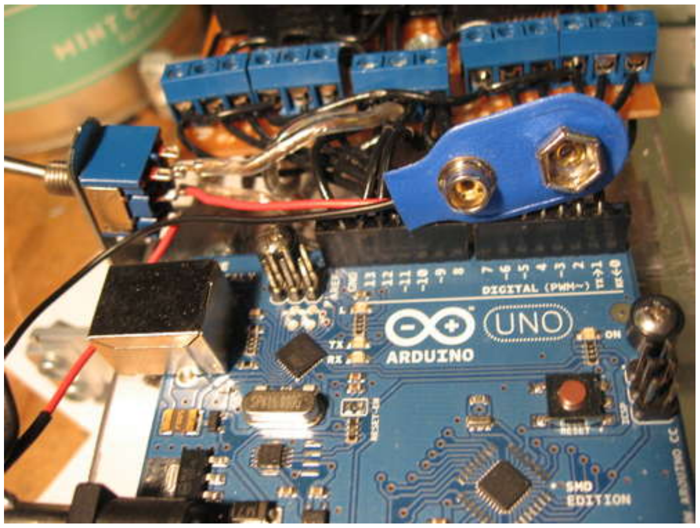


Image Notes
1. 9V breaker
2. 12V breaker

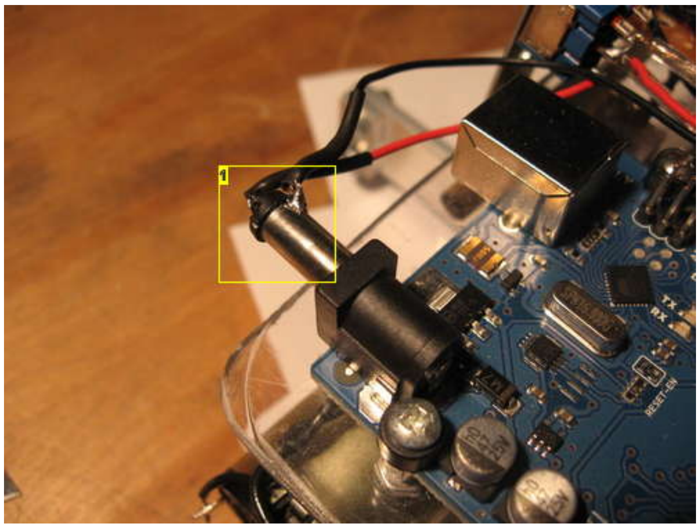


Image Notes
1. These barrel connectors can be scavenged from old wall worts

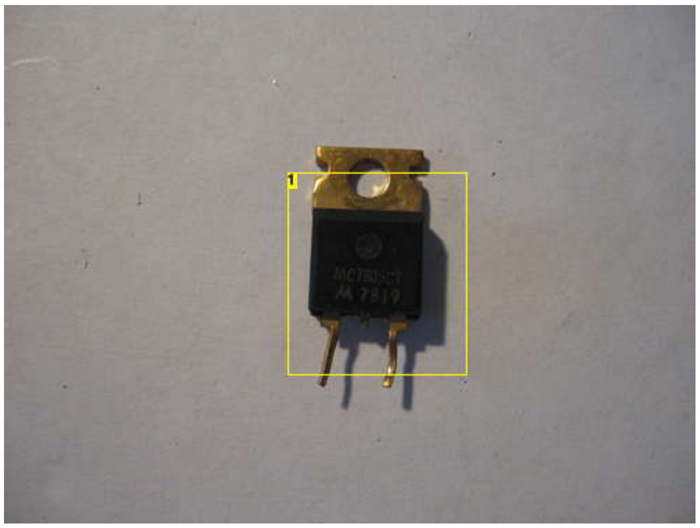
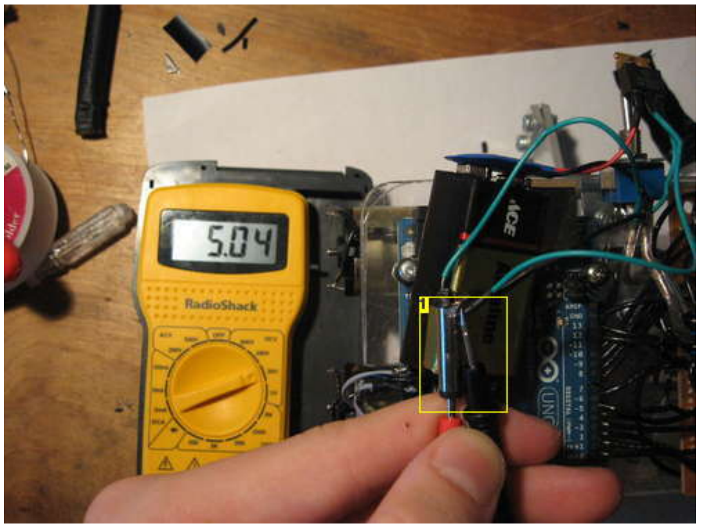


Image Notes
1. 5V regulator, up to 1A



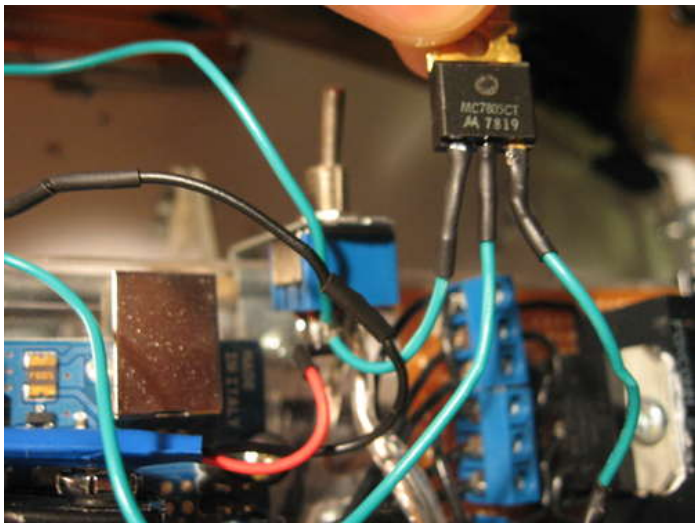


Image Notes
1. Ignore how this is wired, I did it wrong the first time around.

Step 6: Legs
The legs are some of the most important parts of this robot, because their design determines whether or not the robot can grip onto trees. I decided to have four pairs of legs, each pair controlled by one motor.

To make the legs, I cut four 8.5" lengths of the aluminum bar. I marked the segments 2.5" from each end. At those marks, I bent the aluminum at a right angle, to make a "U" shape. If you do not have a bending brace (which I don't) you can get a clean bend by clamping the aluminum with a c-clamp right on the mark, and pushing the unclamped end against a solid surface, like a work bench.

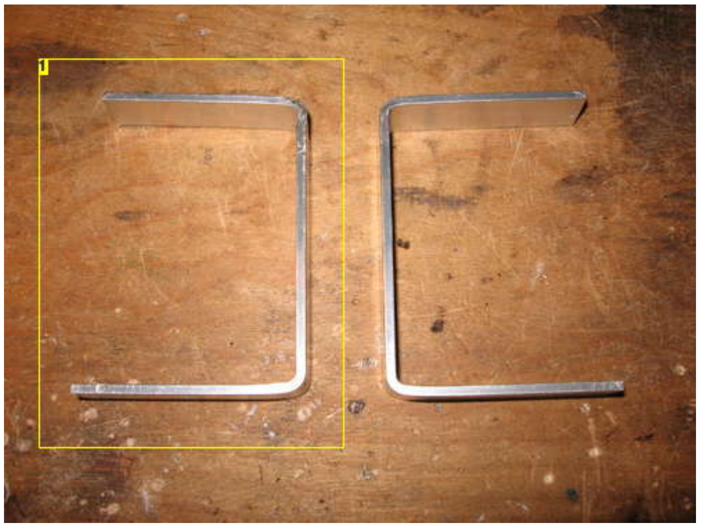


Image Notes
1. End product

Step 7: Feet
To grip the tree, the robot has very sharp feet at the end of its legs (where else?). The feet are made from jumbo-sized T pins, which you can get at your local fabric store. To fasten them to the legs, I made some clamps out of aluminum. I cut 8 3/8" or so lengths of aluminum, and filed a thin groove lengthwise into each of them, for the pins to fit into. Then I drilled a pair of holes into the aluminum, and corresponding holes into the ends of the legs. The clamps were then bolted down to the legs, with the pins inserted in the grooves. I left about 3/8" of an inch of the pins extending form the legs, but the length can be adjusted by loosening the bolts.

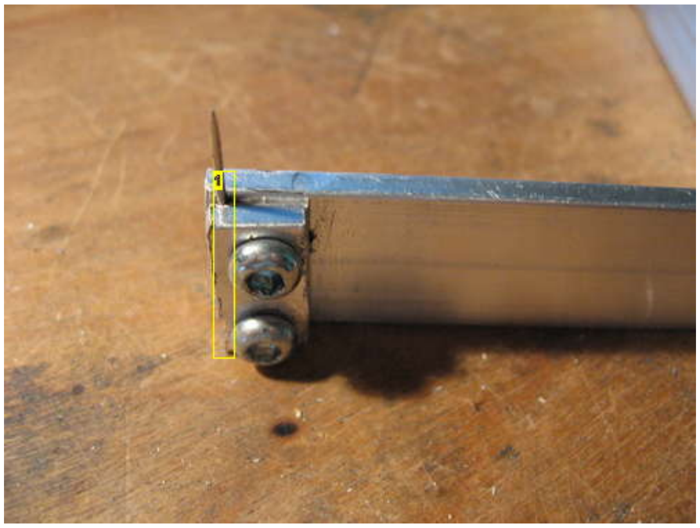


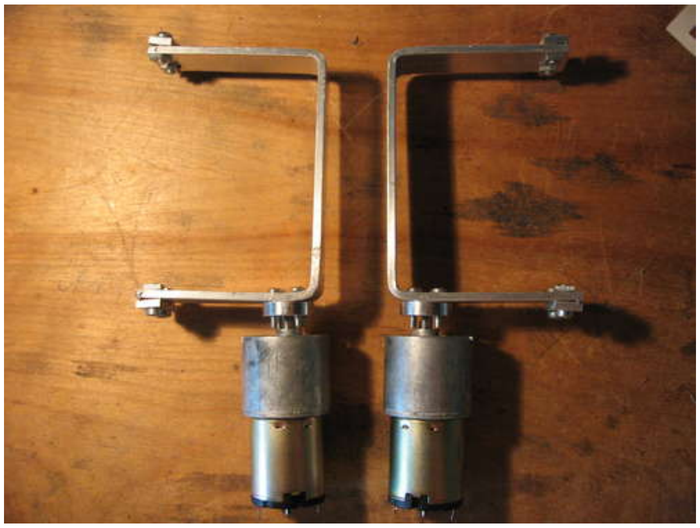
Image Notes
1. It is difficult to see from this picture, but there is a small groove down the clamp which holds the pin at a right angle to the leg.

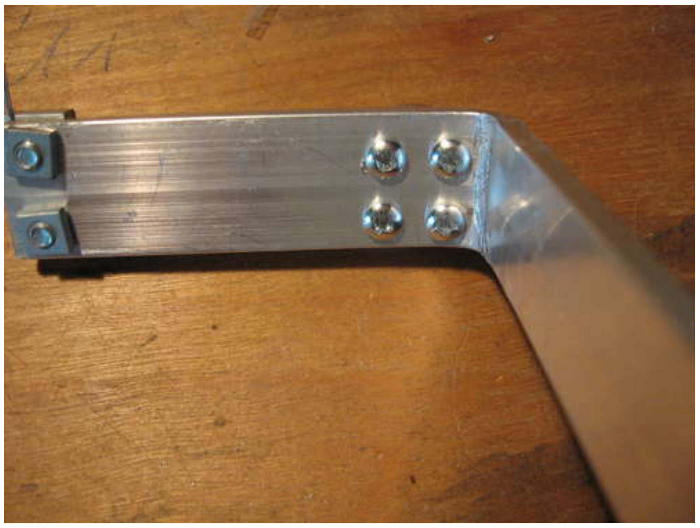
Image Notes
1. This is approximately how the legs will be positioned on the robot

Step 8: Motor Hubs

The next step is to couple the legs to the motors. I found these handy 5mm mounting hubs , which were perfect for the job. I drilled four holes in a square on one side of each pair of legs, and screwed the hub to the legs with 4/40 screws. To fix the motors to the legs, you simply line up the flat side of the motor shaft with the screw in the hub, and tighten the screw.

Image Notes
1. I ended up switching these out for shorter screws





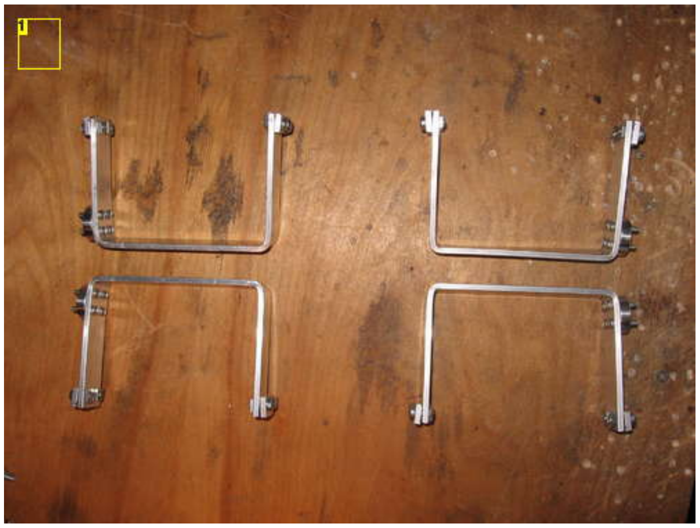
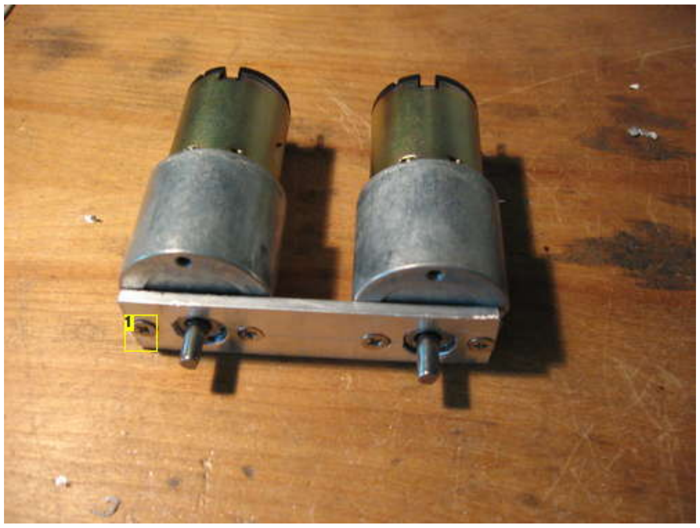


Image Notes
1. Approximate final layout of the legs

Step 9: Building the Frame

With the legs finished, the next step was to build a frame to hold the motors and legs together and in place. I started the frame by making a plate out of aluminum to hold the motors together. I drilled the plate to fit the screw holes of the motors and the gearbox shaft. The motors are held in place by 3/32 screws.

Next, I made a matching plate for the opposite side of the leg assembly. This plate holds the legs straight while they turn. I drilled holes through the legs, opposite to the motor hubs. Then I bolted the legs through the plate with washers and a locknut to hold them in place and let them spin freely on the bolt.



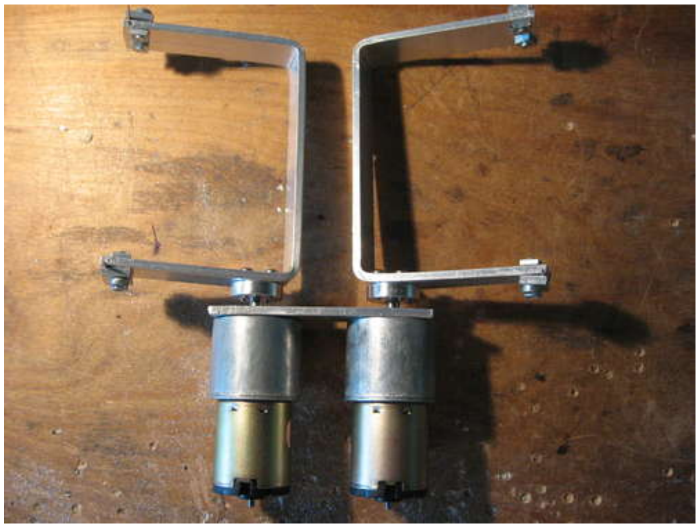
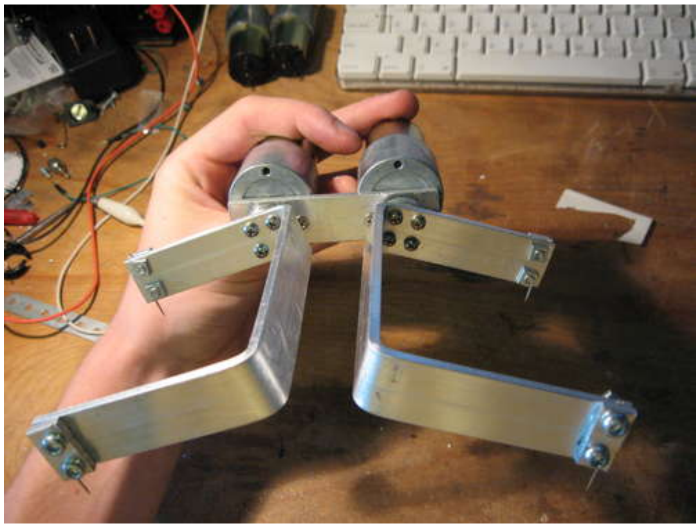
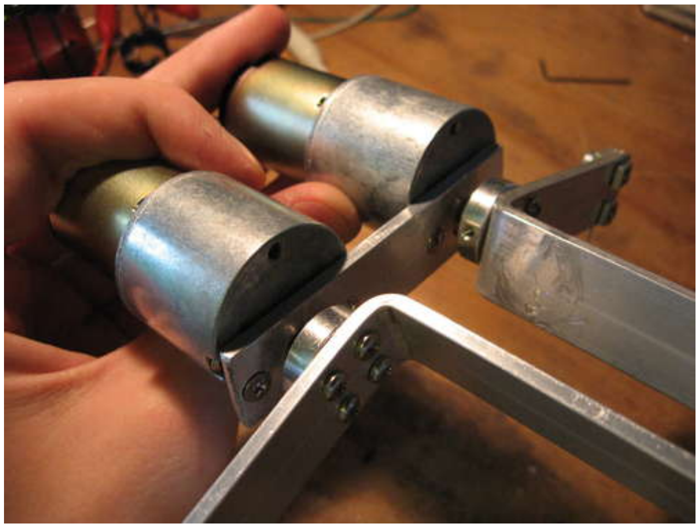
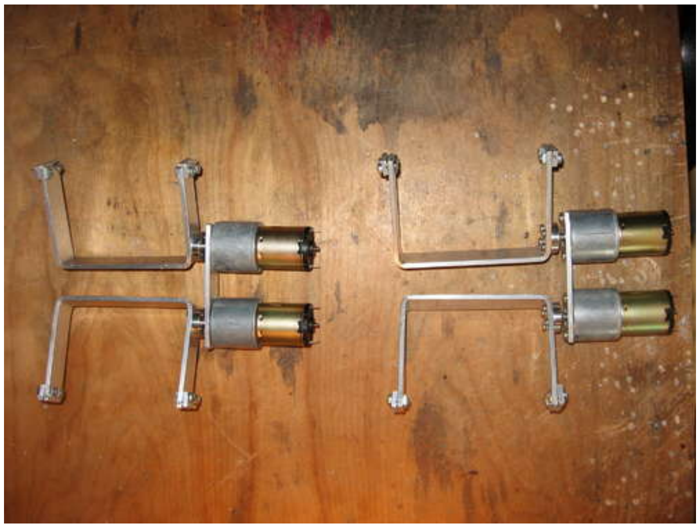


Image Notes
1. I countersunk the screws, because the heads were beveled.







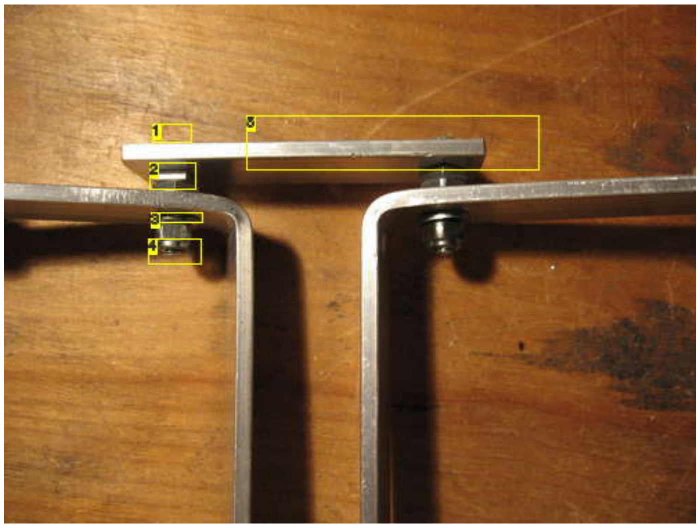
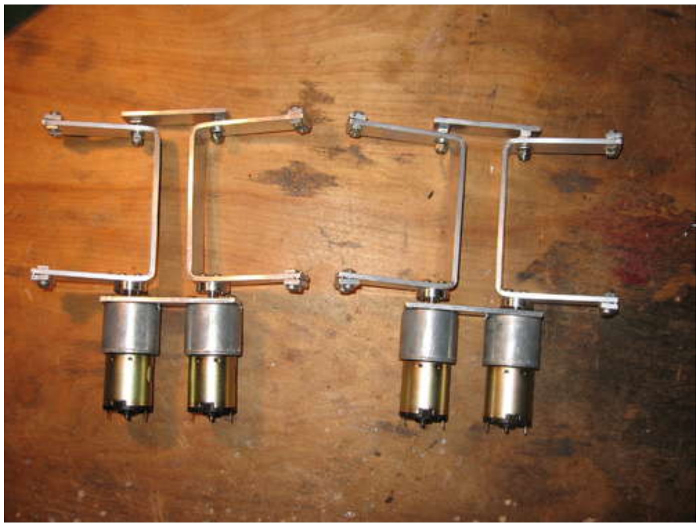


Image Notes
1. Bolt head
2. Nuts fasten the bolt to the plate
3. Washer
4. Locknut holds the washer on
5. Second plate



Step 10: Frame, cont.

Next, I made a piece out of aluminum to fix the two opposite plate together. This piece sits between the pair of legs on each assembly, and is the primary structural support of each segment. As well as holding the robot together, it provides a place to mount the electronics and other components later on.

I bent the aluminum at right angles using a c-clamp, and drilled four holes in each end. I drilled matching holes in the motor plate and the opposite plate in each leg assembly, and then bolted everything together with 4/40 screws.

Once both segments of the robot were built and structurally sound, I could test their tree-gripping ability by hooking the motors up directly to a battery. Fortunately, they worked quite well, or I would have had nothing else to share.

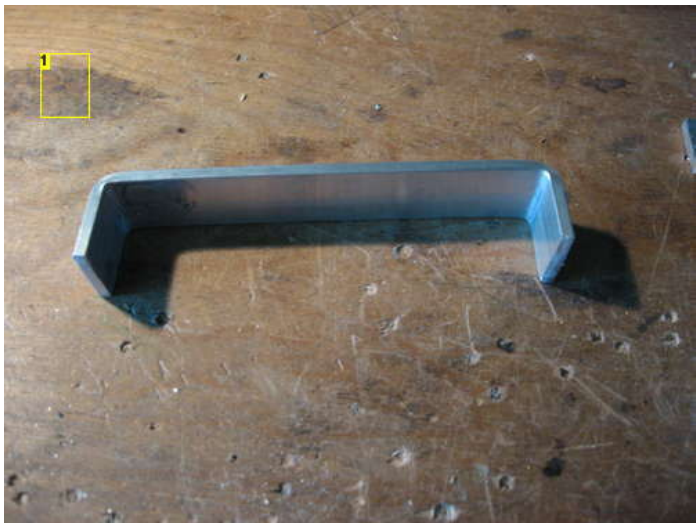
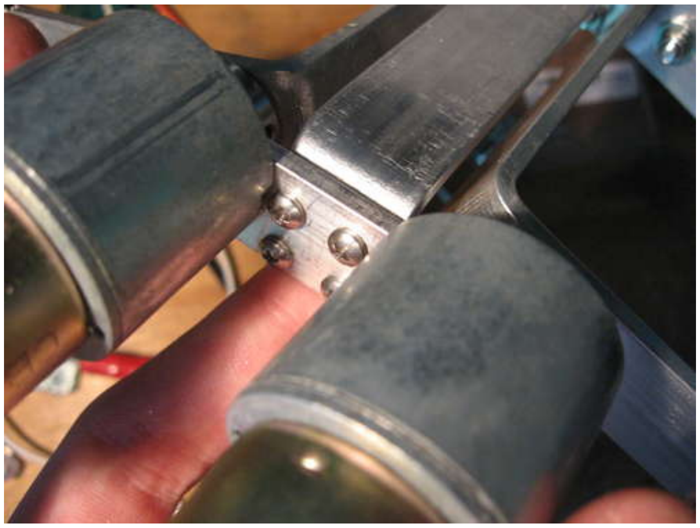
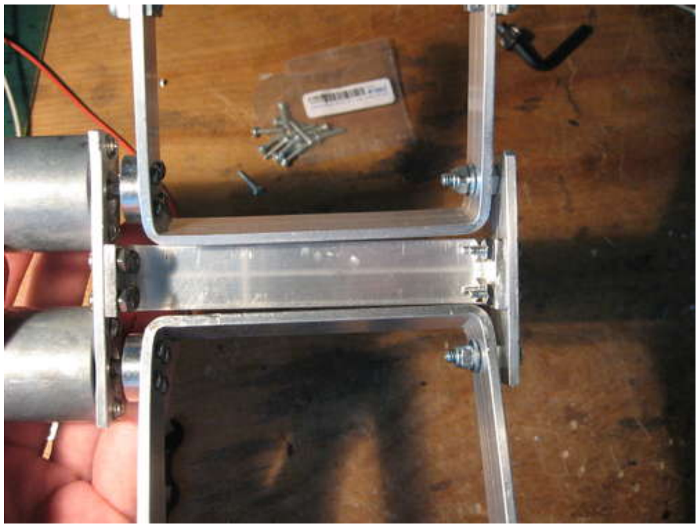
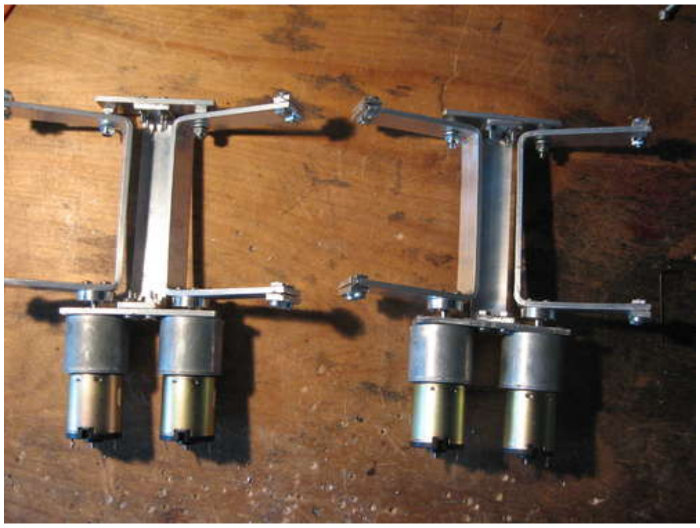


Image Notes
1. 2 of these







Step 11: Electronics Platform

To hold the electronics, I cut an approximately 6" x 3" piece of acrylic. I drilled six holes in it and screwed standoffs into the holes, to support the Arduino and the motor controller. Then I drilled four holes in the top of the leg assembly, and bolted the acrylic to the assembly, with spacers to lift it up above the motors. Finally, I screwed the Arduino and motor controller into the standoffs.

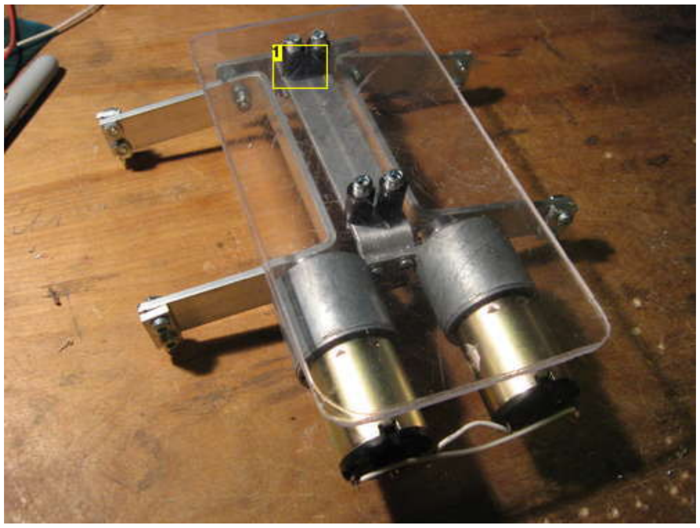


Image Notes
1. I actually eventually had to ditch these spacers. Read on.

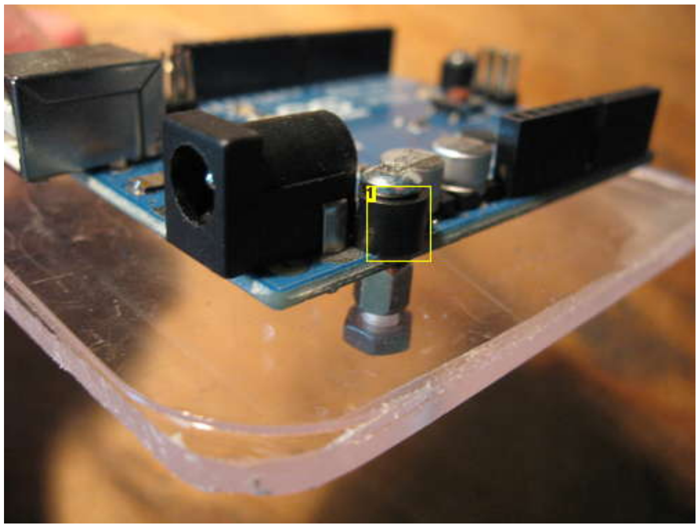
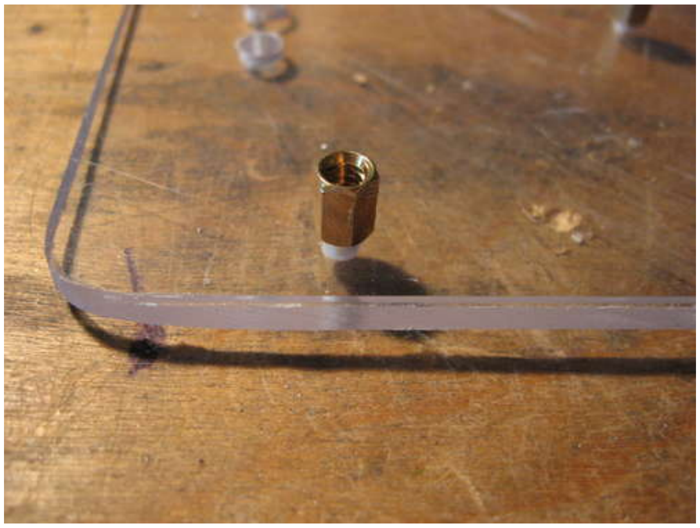
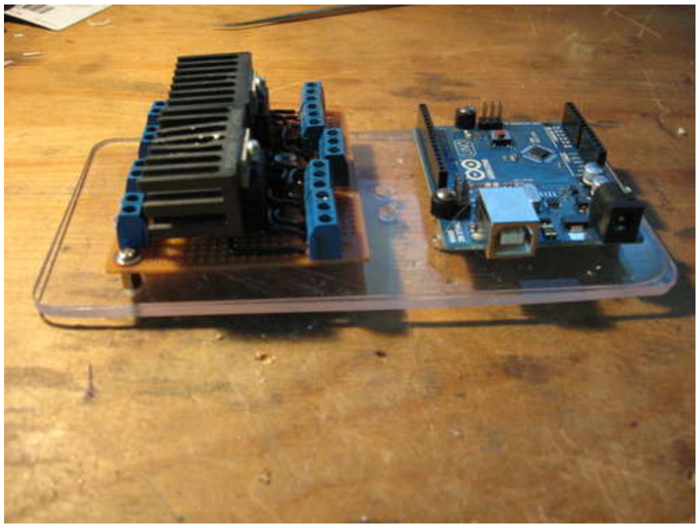
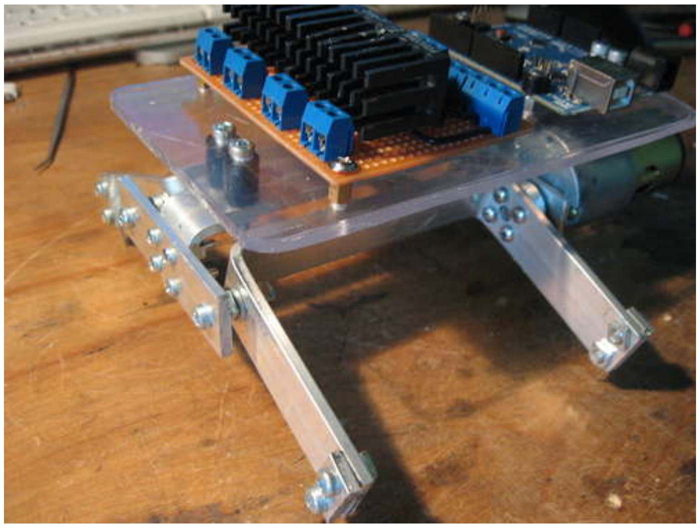


Image Notes
1. I used a spacer so the head would not short anything







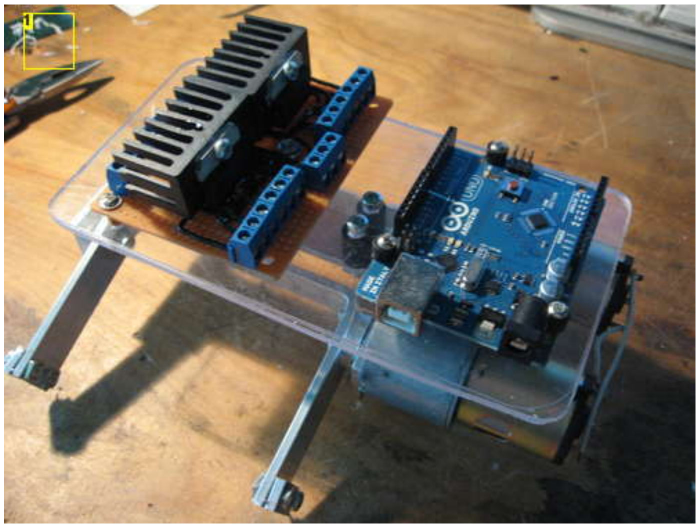


Image Notes
1. Starting to look like a real robot!

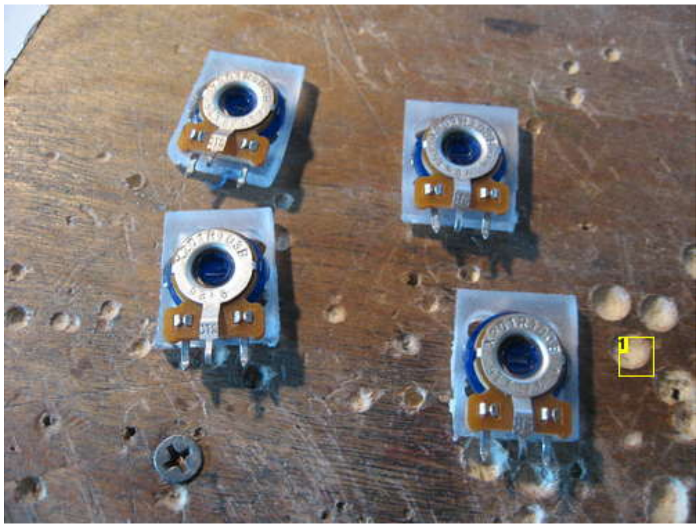
Step 12: Rotation Sensors

Rotation sensors are key to the operation of this robot. It has one rotation sensor per motor, so the robot knows the exact position of each leg at all times, allowing for precise control of the legs. For my rotation sensors, I used four very thin trimpots I had lying around. Pots are extremely easy to interface with the microcontroller, and are plenty precise for my purposes. They were not, however, very easy to interface with the hardware of my robot.

While designing and building the leg assemblies, I neglected to build in an easy way to connect the potentiometers to the legs. In the solution I came up with, one side of the pot is fixed to the inside of the leg by the protruding screw heads. The other side of the pot is fixed to the locknut on the end of the bolt that holds the leg in place. When the leg turns, the side of the pot fixed to the leg turns, while the side fixed to the locknut is held in place.

To interface the pots and the legs, I first sanded the plastic side of the pots flat. I took four squares of acrylic, approximately 3/4" on each side, and drilled four holes in each, corresponding to the four screw heads in each leg. Then I glued a potentiometer to the center of each acrylic square.

To fix the opposite side of the pots to the locknut I had to get even more creative. First, I glued metal standoffs scavenged from a PowerMac G5 case to the metal side of the pots. Then I glued the plastic shaft from a Bic pen to the metal side of the pots. The other ends of each pen were cut to fit within the metal legs. Then the pen shaft was forced over the square locknut and epoxied to it.



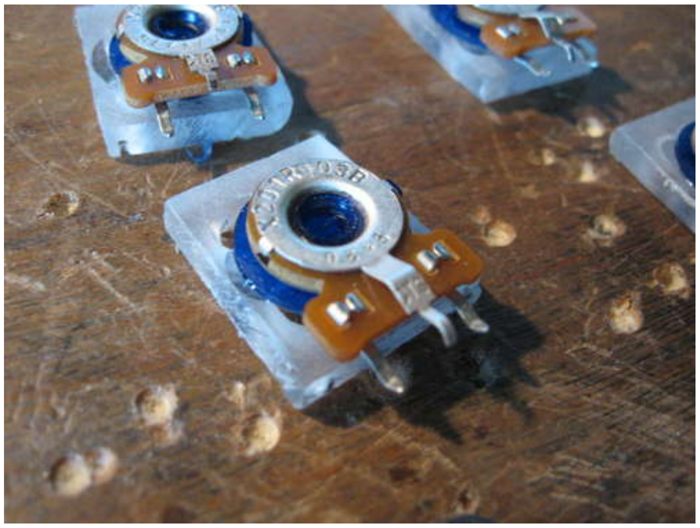
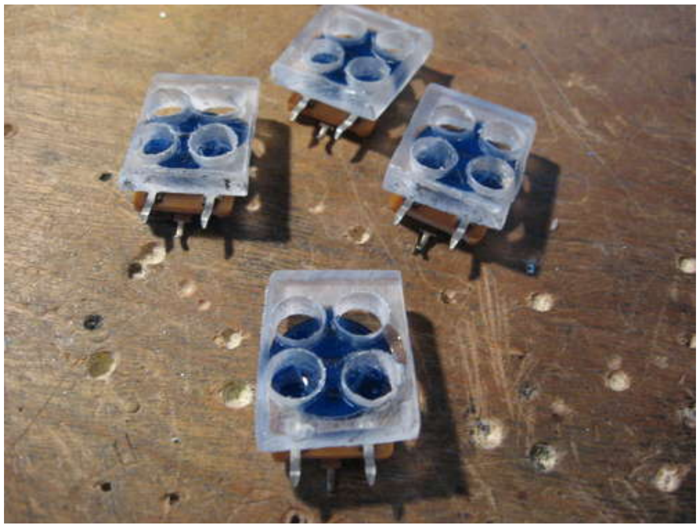


Image Notes
1. My poor desk!



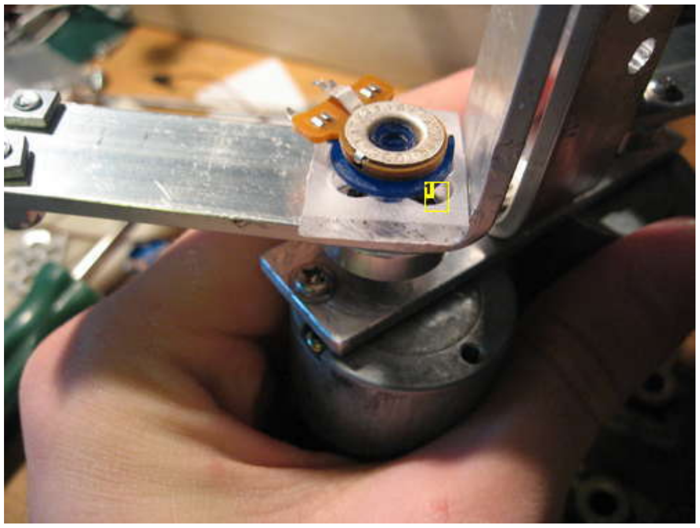


Image Notes
1. Screw heads are under there

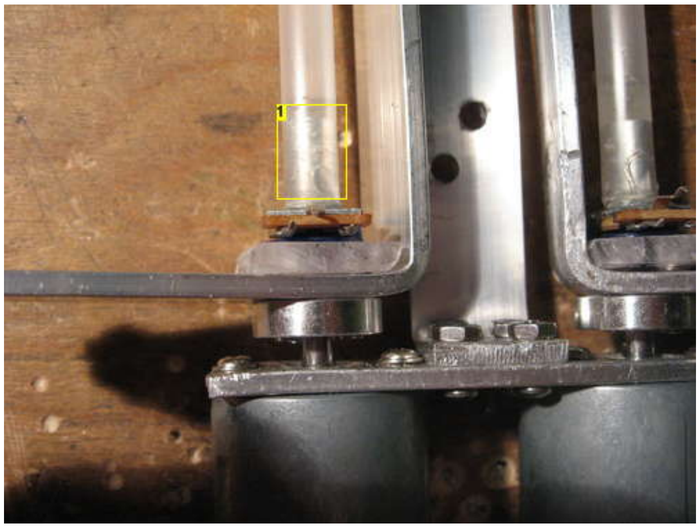
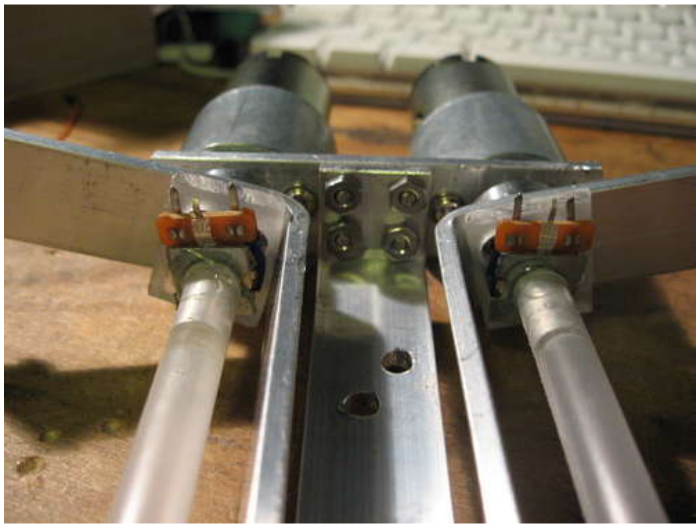
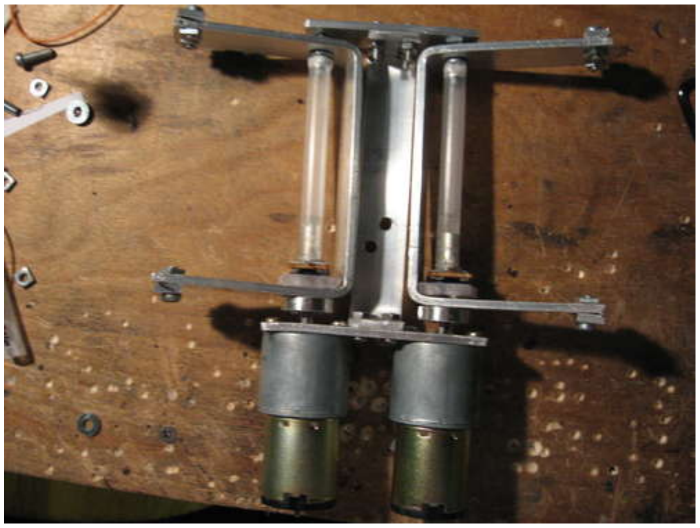


Image Notes
1. The standoff is in there

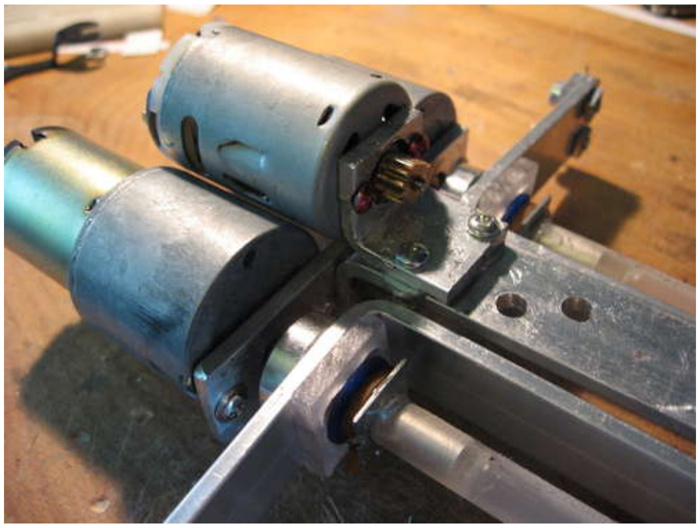


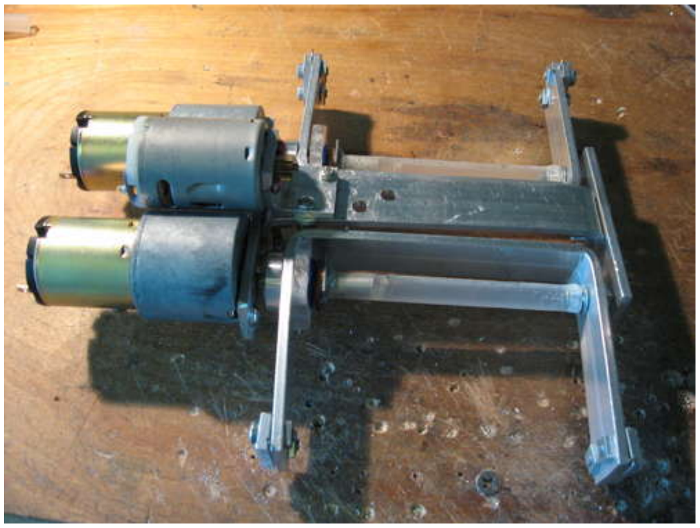


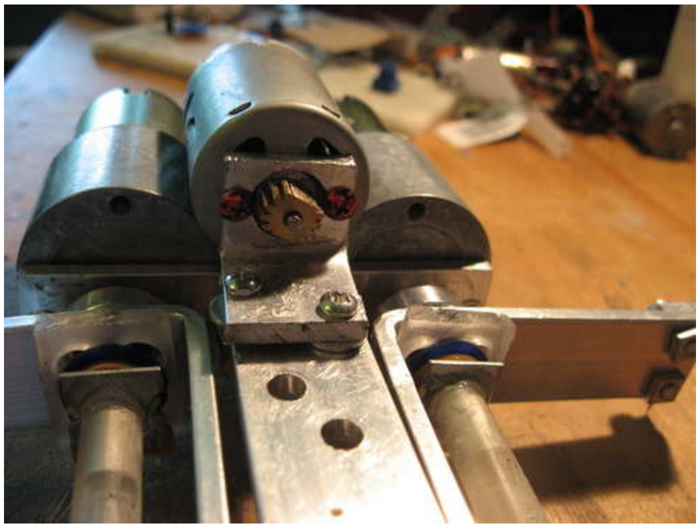
Step 13: Backbone Motor

To move up and down a tree, the robot extends and contracts by spinning a threaded rod that is fixed to the top segment. When the rod is spun clockwise, the two segments are pulled together, and the are pushed apart when it spins counter clockwise. To spin the rod, I needed a relatively high-torque low-speed motor that would run at 12V, and I happened to find just such a motor in my box of parts. This particular motor came fitted with a brass gear. To assist with coupling the motor to the threaded shaft, I filed two sides of the gear flat.

To mount the motor to the robot, I bent a short length of aluminum to an "L" shape. I drilled a large hole out of the center of one of the faces (for the motor shaft and gear) and two small holes in both faces for bolting the motor to the metal and bolting the metal to the robot. I drilled corresponding holes into the back of one end of the segment of the robot without the electronics, so that the motor was positioned between the two legs.



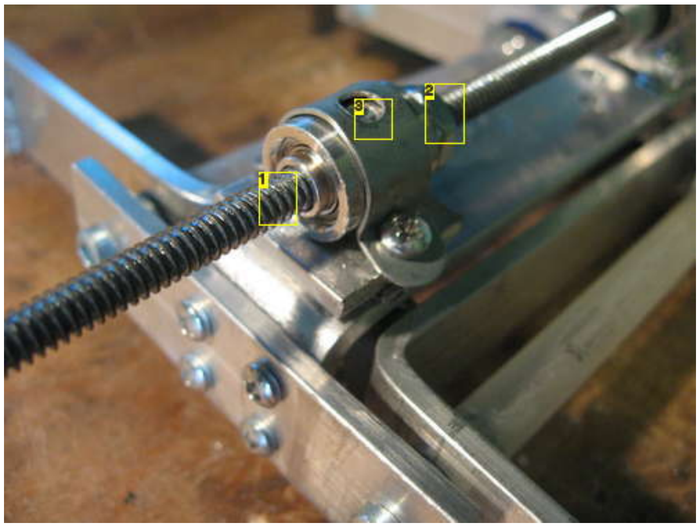




Step 14: Mounting the Spine

There are a couple problems with getting a threaded spine to spin smoothly. First, it must be coupled to the motor well, and second it must have some sort of bearing fixed to it on which it can spin. One of the first things I found while trying to couple the motor shaft to the threaded rod was that the connection should be flexible for smooth operation. I made my coupling out of two segments of clear nylon tubing. A wider diameter segment fits tightly over the gear attached to the motor shaft, while a thinner segment fits into the larger tubing and tightly over the outside of the threaded shaft. The coupling is secured with a zip tie.

With only the coupling, the threaded shaft still can not bear any load, because it would just pull off the motor. To support load, I made a bearing for the shaft out of an old hard drive read/write head bearing. I drilled out the center so that the threaded rod could pass through it. I then fed the rod through it, and fastened a nut on each side of the bearing, to hold the threaded rod in place. I then bolted the bearing down to the back of the robot's frame.



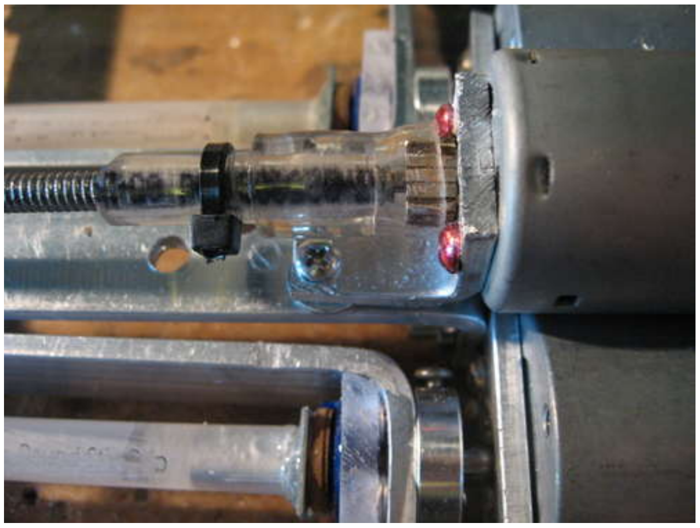


Image Notes

1. A nut will go here
2. Nut
3. This is a bit of scrap Meccano metal

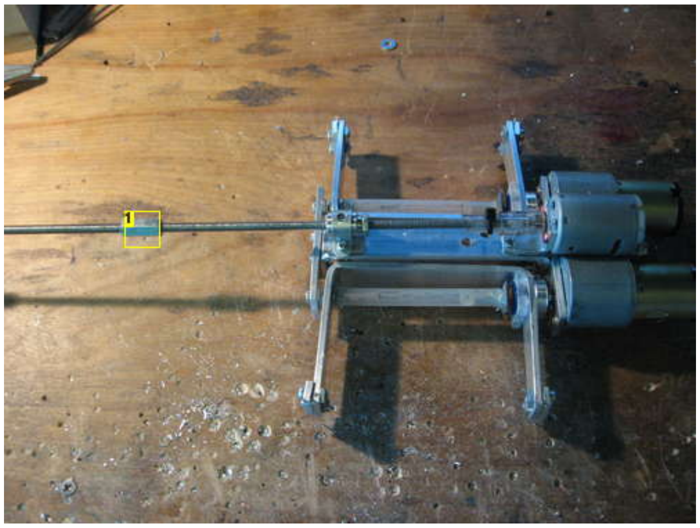


Image Notes
1. The other segment will be fixed to this nut

Step 15: Mounting the Spine, cont.

For the spine to work, it must pass through a nut that is fixed to the other segment of the robot. For ease of mounting, I used a large 1/2" long nut. To fix it to the segment of the robot, I cut two ~4" lengths of aluminum, drilled them to match the bolts that hold the acrylic piece, and mounted them through the screw holes, across the frame. These will later become supports for linear slides. I then drilled four more holes around the center of the back of the robot, with enough space between them for the nut to fit. I cut a piece of aluminum to run along the back of the robot, and drilled it to match the holes. I then placed the nut on the back of the robot, placed the bar of aluminum on top of it, and bolted through the bar, to sandwiched the nut between the two pieces of aluminum. Finally, I threaded the rod through the nut.

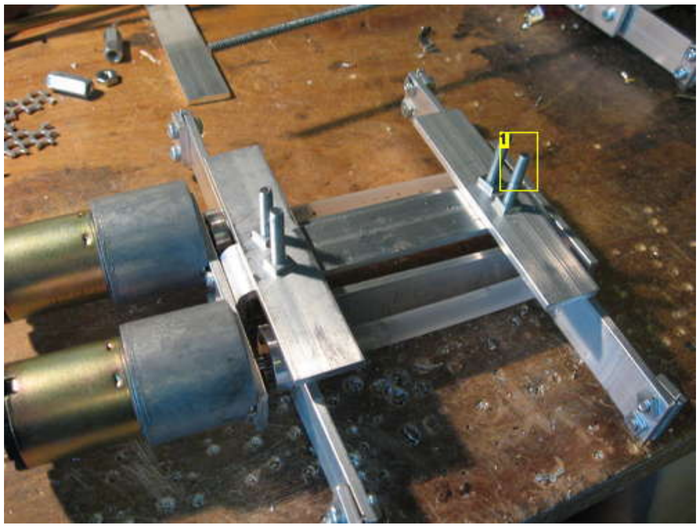


Image Notes
1. These will go through the holes in the acrylic

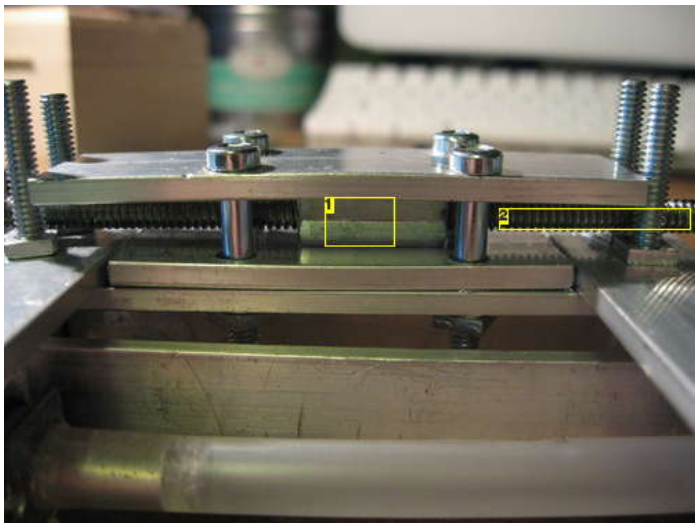
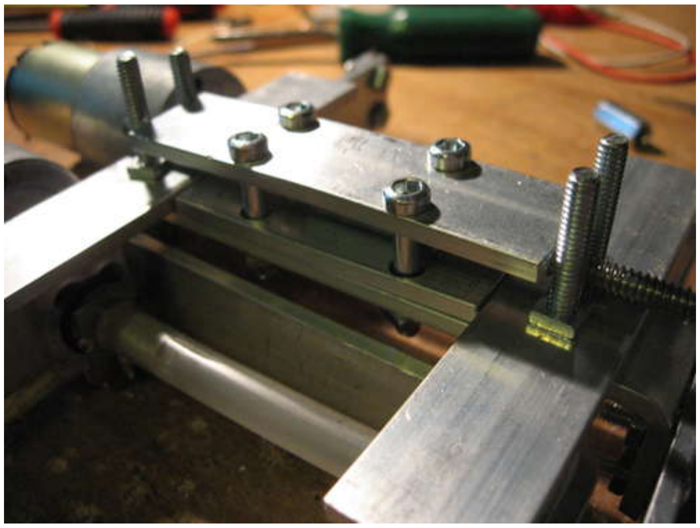


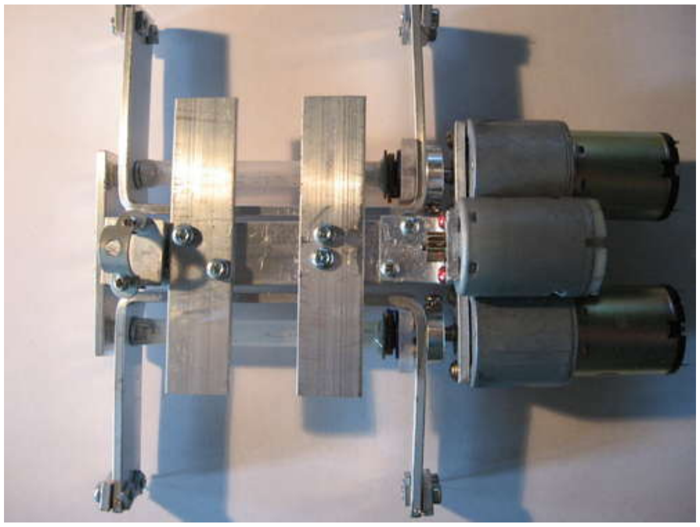
Image Notes
1. Nut
2. Threaded rod



Step 16: Linear Slides

Without something holding the two segments of the robot in the same plane, the top segment would turn when the threaded rod turned, instead of moving up or down. To keep the two halves of the robot in the same plane, I built linear slides out of two steel rods and brass tubing.

First, I added a pair of aluminum bars to the segment of the robot without the electronics, to match the pair on the other segment. To mount the steel rods and the brass tubing to these, I made a clamp system similar to the clamps holding the feet in place. To do this for the large diameter rods, I first clamped two 3/4" squares of aluminum together. I then drilled a 1/8" hole down the intersection of the squares, and then took them apart. I drilled two holes in each square, and corresponding holes in each supporting arm. Then I repeated the process four times. To get the slides perfectly parallel to the threaded rod, I had to bend up the supporting arms on the non-electronics segment of the robot.



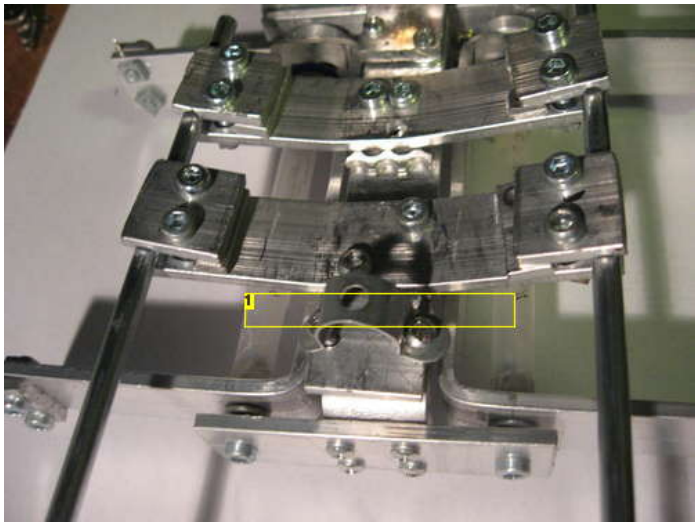
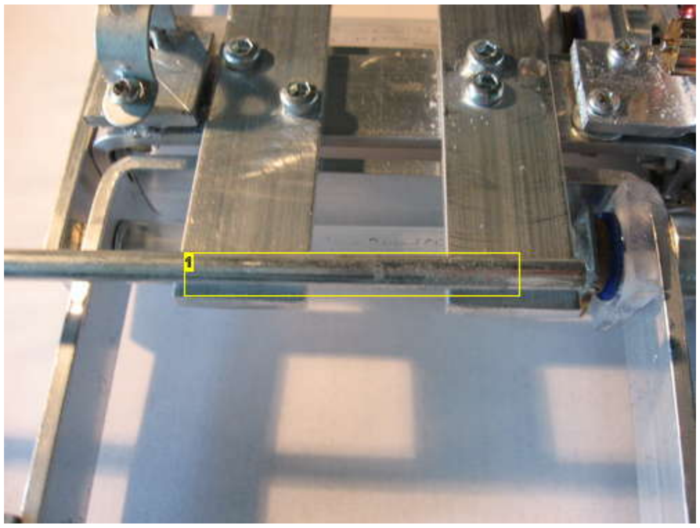


Image Notes
1. You can see the bend in the arms



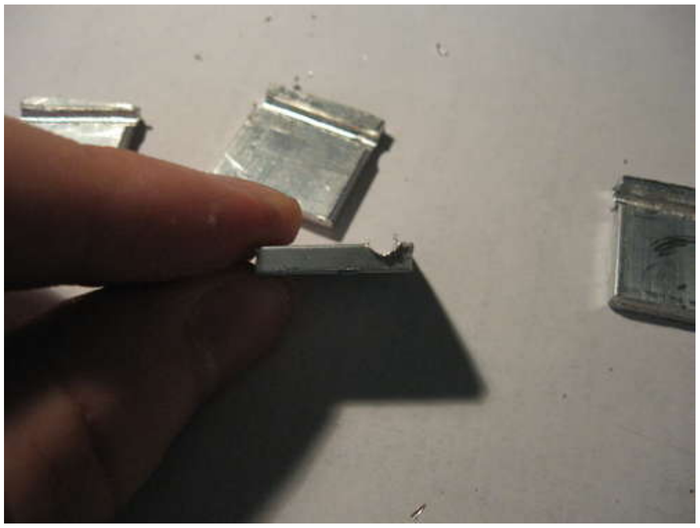
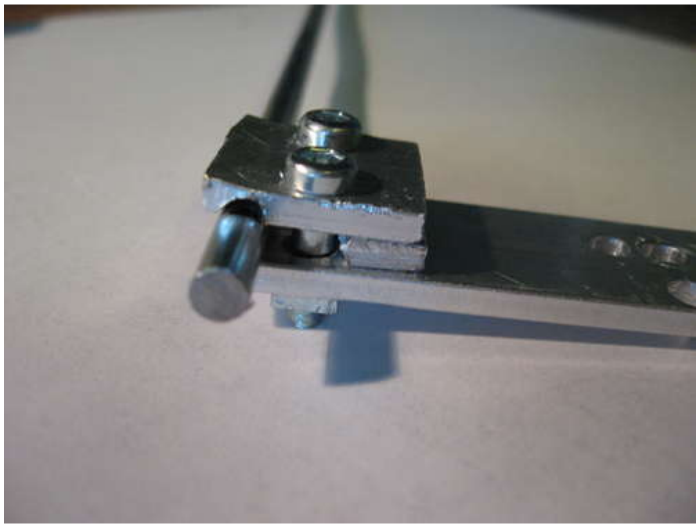
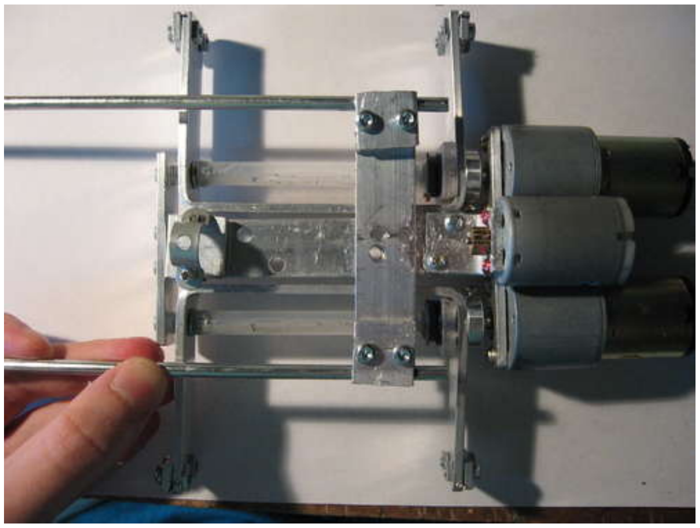
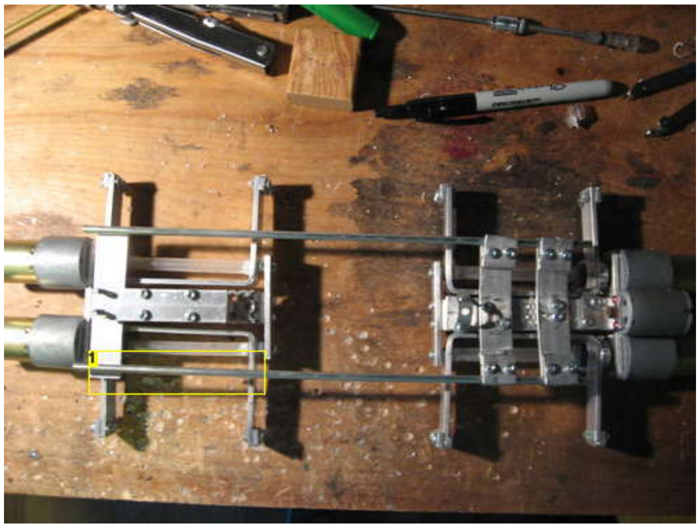


Image Notes
1. This needs to be fixed in this position







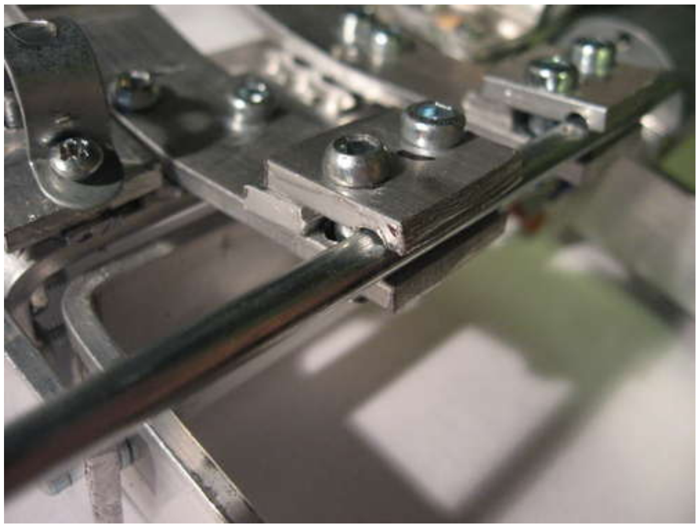


Image Notes
1. Brass tubing will go right here

Step 17: Wiring the Robot

The next step is to wire all the electrical components of the robot together. I started out by soldering long wires to the contacts on the motors. I twisted the wires together by chucking one end in an electric drill and holding the other end with pliers (a trick I learned from the Ben Heck show). Next, I wired together the pots on the legs. I did this using segments of ribbon cable from an old IDE cable. I wired the pots so that they all had a common ground and input voltage. The input voltage was connected to the +5V pin on the Arduino, and the four signal wires were soldered to headers and then connected to analog inputs A0 - A3 on the Arduino.

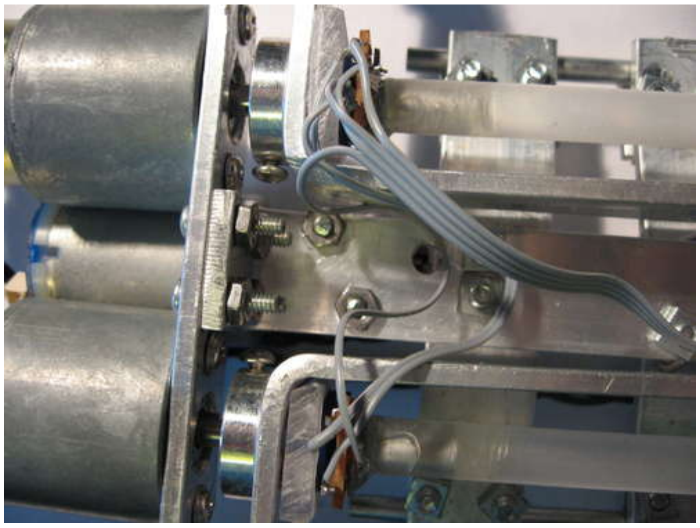
Because this robot is autonomous, I needed a method for controlling the robot's actions so that I could get it to release from the tree. For this, I just used a simple slide switch connected to a digital input on the Arduino

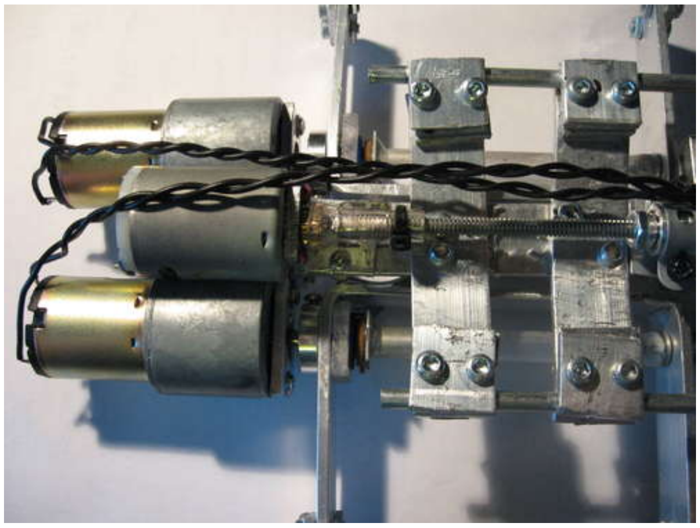
Next, I wired the digital output pins on the Arduino to the inputs on the motor controller. First, I connected all the motor enabling pins on the motor controller to eachother. The rest of the wiring went as follows:

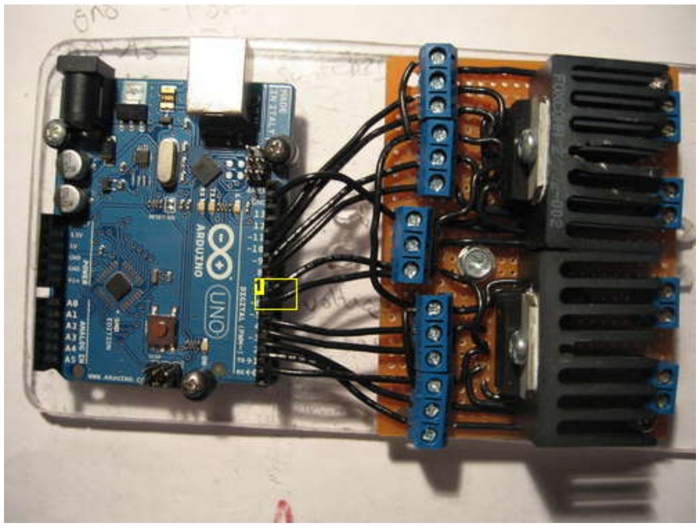
1. Enable Motors
2. Motor 4 Input 2
3. Motor 4 Input 1
4. Motor 3 Input 2
5. Motor 3 Input 1
6. Control Switch
7. empty
8. Motor 2 Input 2
9. Motor 2 Input 1
10. Motor 1 Input 2
11. Motor 1 Input 1
12. Motor 5 Input 2
13. Motor 5 Input 1

I then connected the motor's leads to the terminal strips on the motor controller, and connected the motor voltage terminal to the 12V battery pack, via a toggle switch. I connected the 5V regulator to the logic voltage terminal, via the same toggle switch.

I collected the umbilical cord of wires running between the two segments of the robot into a bundle, and fastened them together with zip ties and electrical tape, to keep them organized.







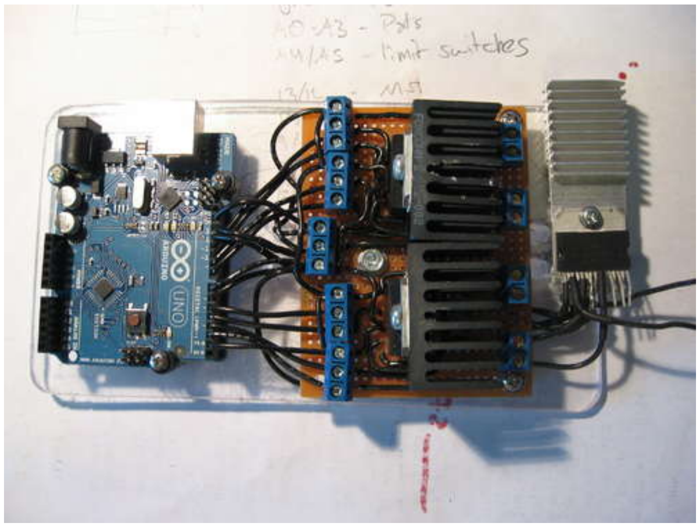
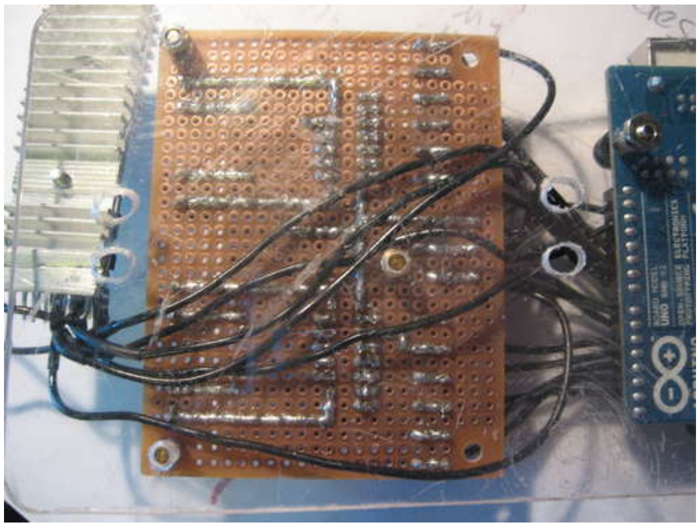
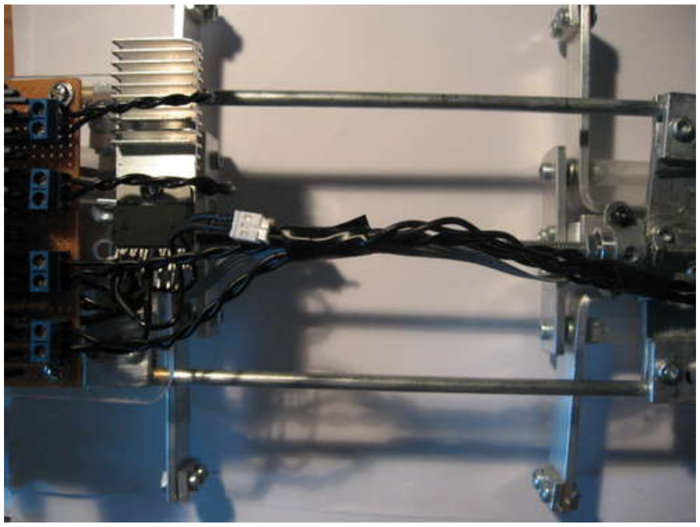


Image Notes
1. Ignore these. I removed them when I though to power the logic voltage off the

voltage regulator





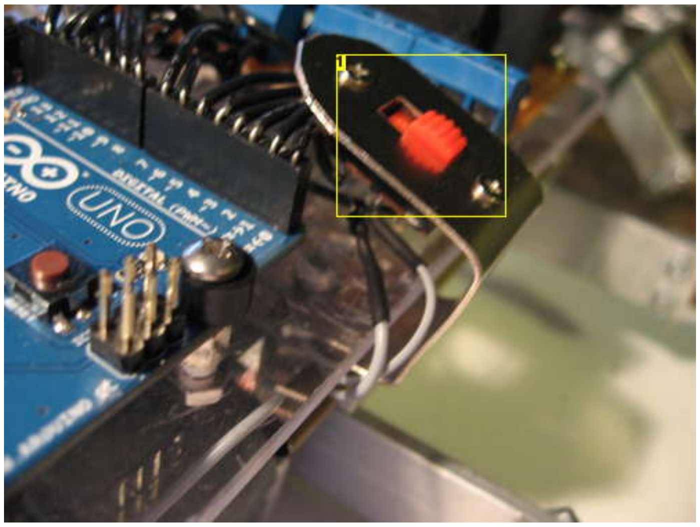


Image Notes
1. Control switch

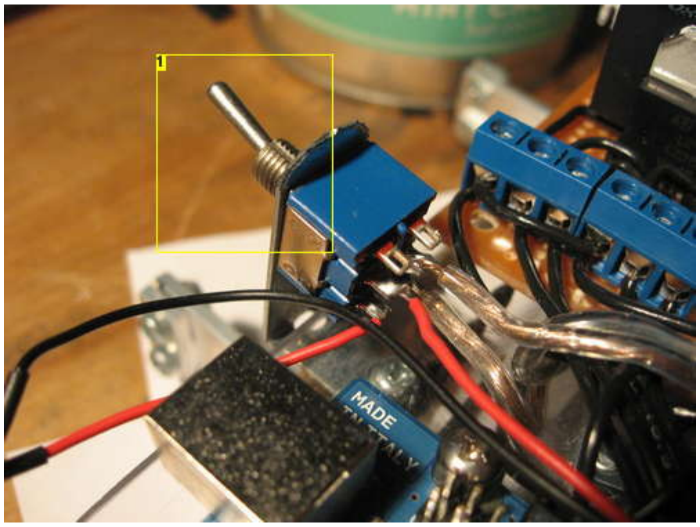


Image Notes
1. On/Off Switch

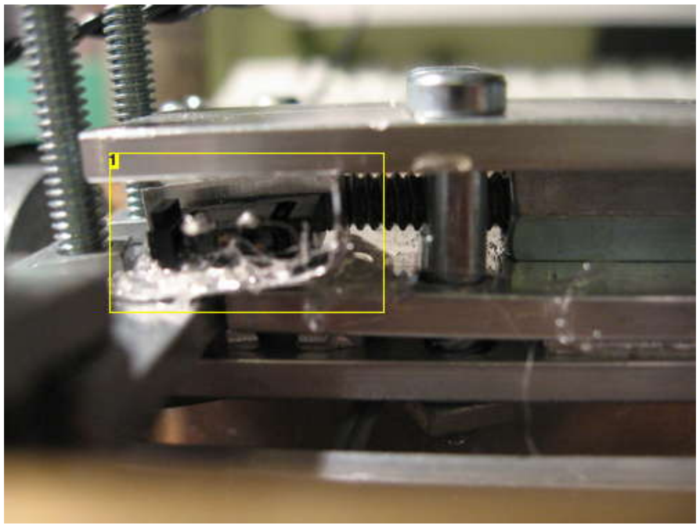
Step 18: Limit Switches

Because I used a regular DC motor instead of a servo or a stepper to spin the threaded rod that is the spine, the robot can not know the degree of extension of the spine at all times. Therefore, limit switches must be used to prevent it from extending or contracting too much.

The spine has two limit switches. One is pressed in when the two segments of the robot are pulled close together, and the other becomes un-pressed when the threaded rod retracts past it. The latter is a switch like this glued parrallel to the threaded rod, on the segment of the robot with the electronics. When the spine retracts, it pushes down the lever of the switch, and when it retracts, the switch opens.

The second limit switch is a push button switch that requires very little force to actuate. I mounted it on a strip of aluminum from the front of the electronics segment.

Both the switches are connected to the same 5V and ground lines as the potentiometers on the legs, and their signals go to inputs A4 and A5, which the Arduino is set to read as digital inputs rather than analog.



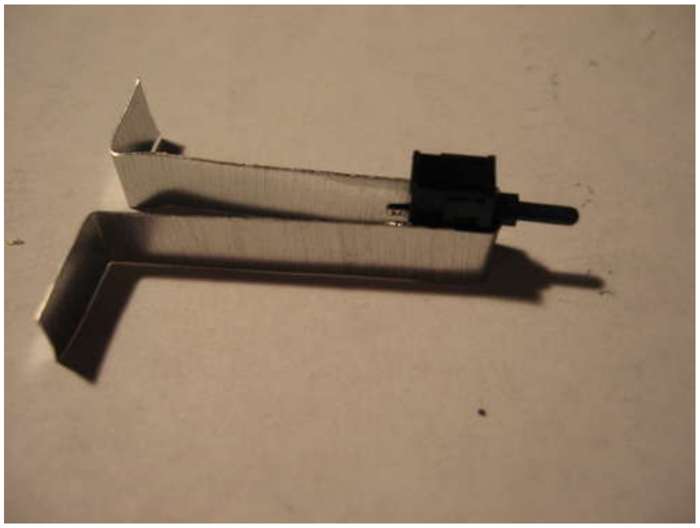
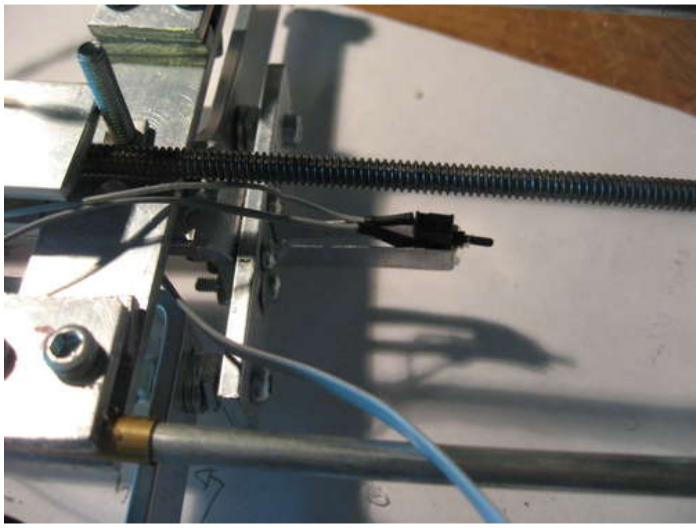
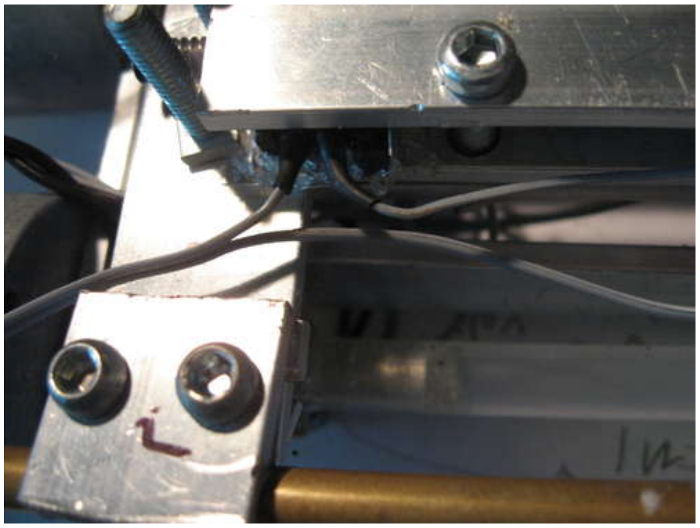


Image Notes
1. Switch

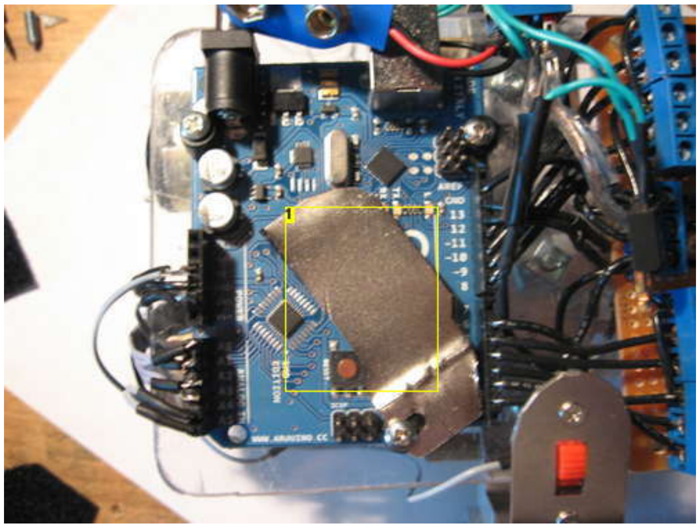



Step 19: Battery Holders

The last mechanical part of this project was to create a way to hold the batteries, while making sure that they are easy to remove for replacement or charging.

The perfect place for mounting the 9V battery was right above the Arduino, so I created a mounting system for it out of some scrap metal. A piece of metal (with an electrical tape insulated bottom) screws on above the Arduino through one of the standoffs. On top of the metal is a bit of stick-on velcro. A piece of metal bent into a "U" shape clips onto the 9V battery, and then sticks to the velcro above the Arduino board, holding the battery in place.

To hold the larger battery pack, I cut two brackets out of some soft plastic angle bar I had lying around. These brackets screw into the arms that hold the linear slides. The battery stays in mostly by friction, but a bit of velcro on one side helps to stop it from slipping out.



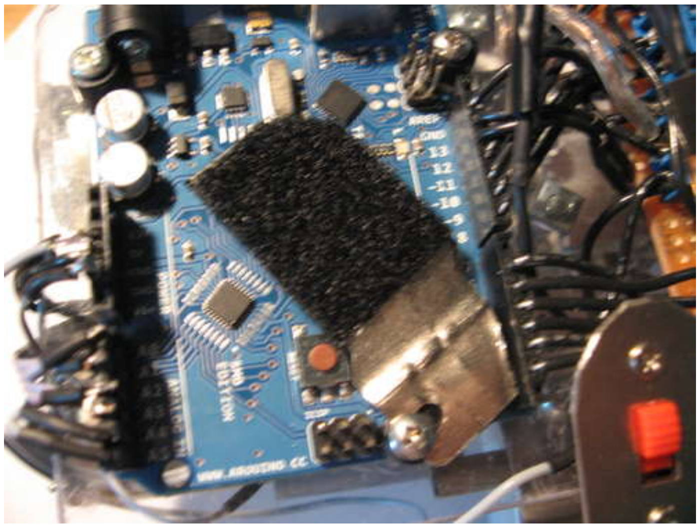
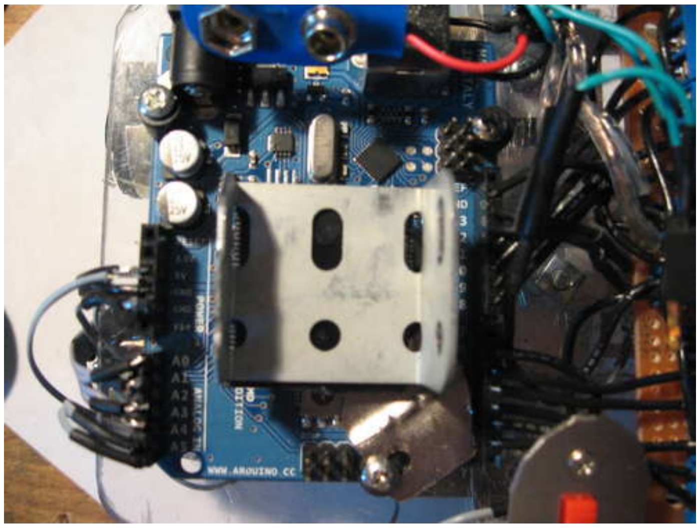
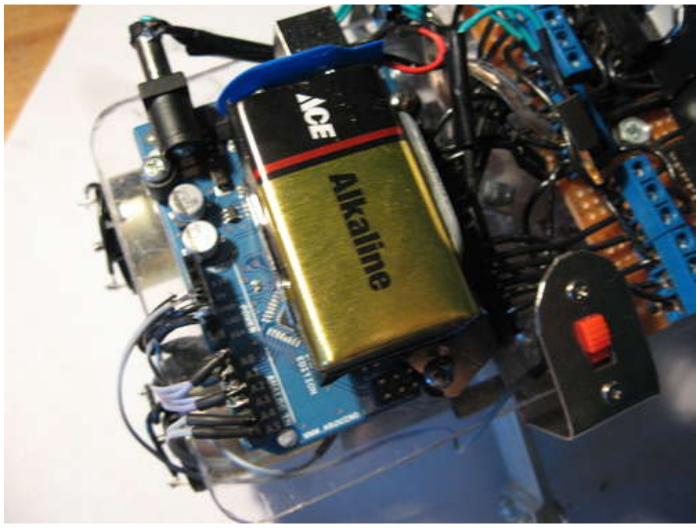
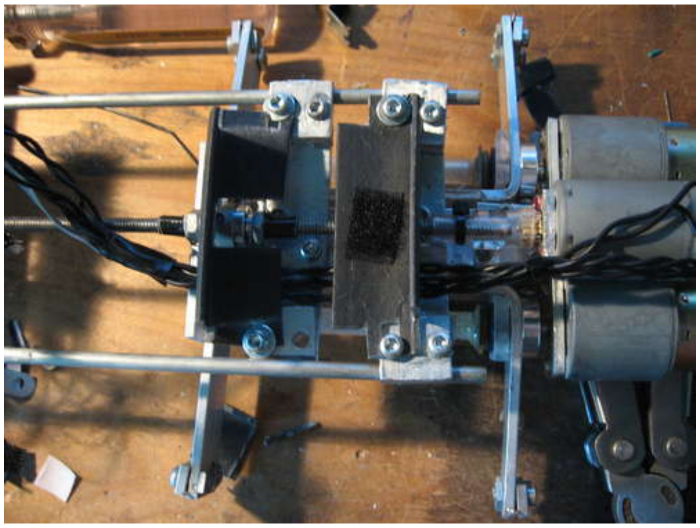
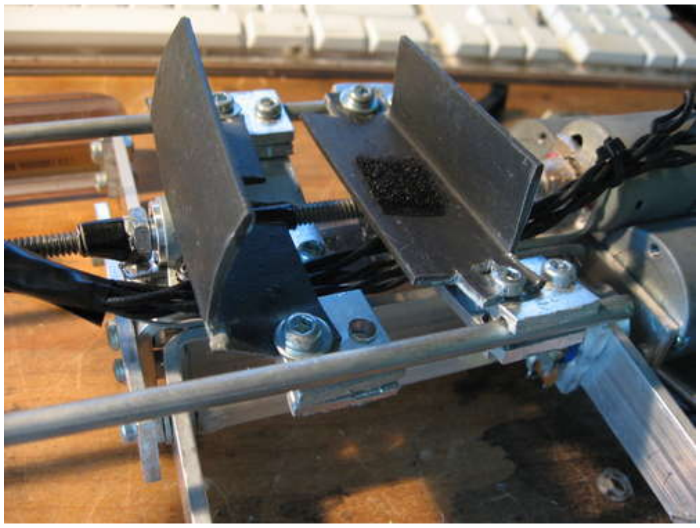


Image Notes
1. The bottom is covered with electrical tape to prevent shorts









Step 20: Programming

To climb up a tree, the robot goes through a simple series of motions. First, the top segment grips the tree and the bottom segment releases form the tree (if necessary). Then the spine contracts, pulling the bottom segment up towards the top segment. Next the bottom segment grips the tree, and afterwards the top segment releases from the tree. Finally, the spine extends, pushing the top segment upwards, and the cycle can start over again. For ease of programming, I wrote a function corresponding to each basic motion. These are as follows:

- closeTop
- closeBottom
- openTop
- openBottom
- Lift
- Push

By combining these functions in the proper order, the robot can be made to ascend or descend trees.

Opening the legs is very simple. The legs turn outwards from the tree until their rotation sensors reach a point set in the program. Then power is cut off to the motors. Closing the legs on the tree, however, is a little bit more complex. Since trees vary in diameter, the legs need to be able to grip a wide variety of diameters without reprogramming the robot for each size. To figure out when to cut off power to the motors, the controller first calculates the speed at which the legs are moving towards the tree. It does this by sampling the position of the legs' potentiometers every .05 seconds. It subtracts the previous value of the potentiometer from the current value to find the distance traveled by the legs over the time period. When the distance travels becomes close to zero (I used 1 in my program), it means that the legs have gripped into the tree and are beginning to slow down. Then the controller cuts of power to the motors, to prevent them from stalling out, or damaging themselves, the motor controller or the gearboxes.

The last piece to the programming puzzle is the method of controlling the robot's actions. If you look at the above movement cycle, you will notice that the robot is gripping the tree at all times. This makes it difficult to remove the robot, so I programmed the control switch to manually control the behavior of the robot. While the switch is off (circuit open), the robot keeps its legs open. Once the switch is turned on, the robot begins its climbing cycle. To remove the robot from the tree, the switch is turned back to the off position, and both sets of legs release.

If you liked this project, please vote for me in the Epilog contest! What would I do with a laser cutter? Well, I could use it to make parts for even more robots and machines (after I finished etching every electronic device I own, of course). Not having to manually cut, file, bend, and grind every component of my robots would let me significantly increase the complexity and variety of what I can build, and would also significantly cut down on construction time, so that I would be able to build even more interesting things.

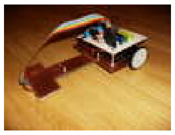
Update
This project was featured on Hack A Day! Thanks for the great article.

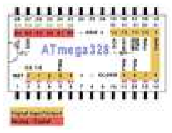
File Downloads

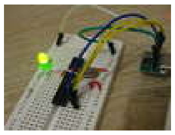
tree_robot.pde (5 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'tree_robot.pde']

Related Instructables

Computer Controlled Musical Christmas Lights by dnicky2288


Line Follower Robot by nbibest


Ubuntu and the arduino. by Computothought


Arduino on a Breadboard (Photos) by ThatCaliforniaGuy

robot with microcontroller (Photos) by www.robotscience.

Arduino Controlled Line Following Robot (video) by earthshine

Belvedere - A Butler Robot by wolffan

simpleWalker: 4-legged 2-servo walking robot by edwindertien

Rave Rover - Mobile Dance Stage

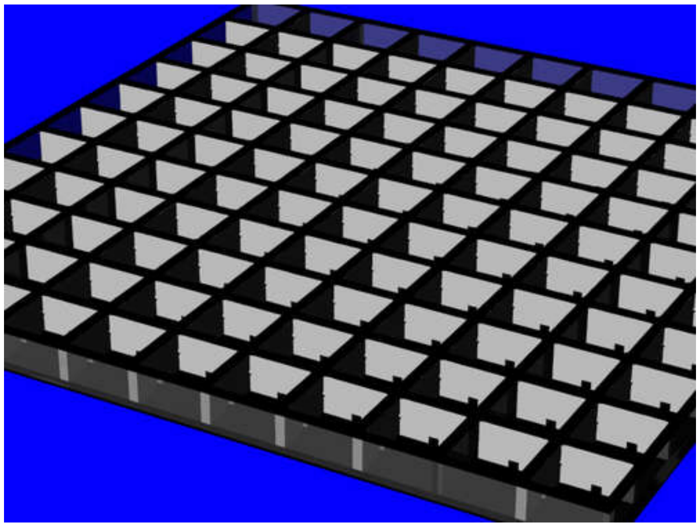
by **cwilliamson8** on September 10, 2011

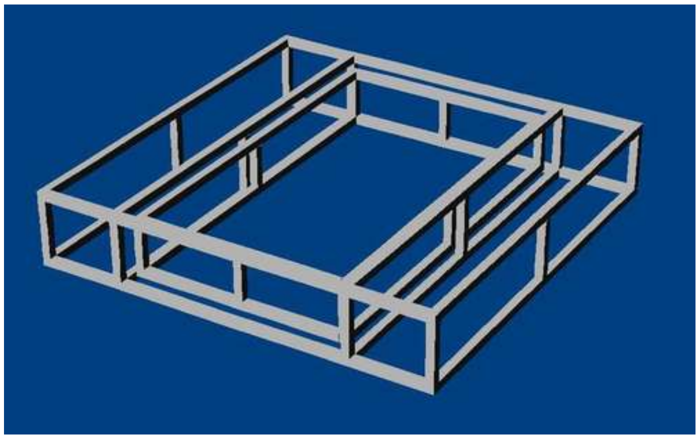
Intro: Rave Rover - Mobile Dance Stage

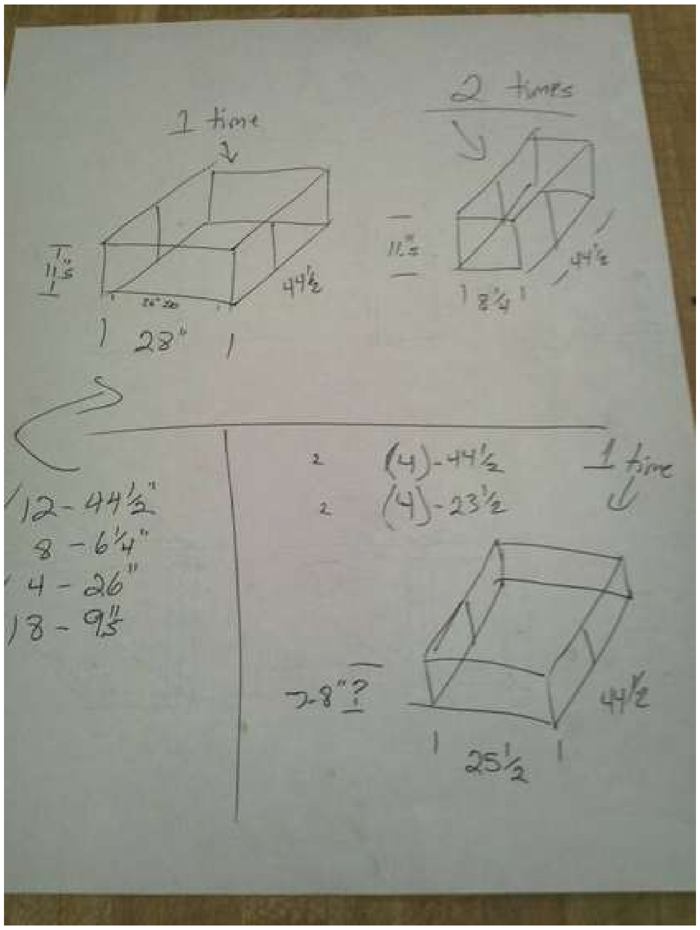
Rave Rover was designed and built to be a portable dance platform for parties, raves, and any other trouble we can get into! I will go into as much detail as I can explaining the entire build process, and where to find parts and other accessories. Be sure to check out more information, including party galleries on our website at www.raverover.com

Step 1: Starting the Build

Before doing any work on putting something together, I always like to sit down and think about the development and how something should go together. CAD is a great resource for this, so I designed most of the layout before spending any money.





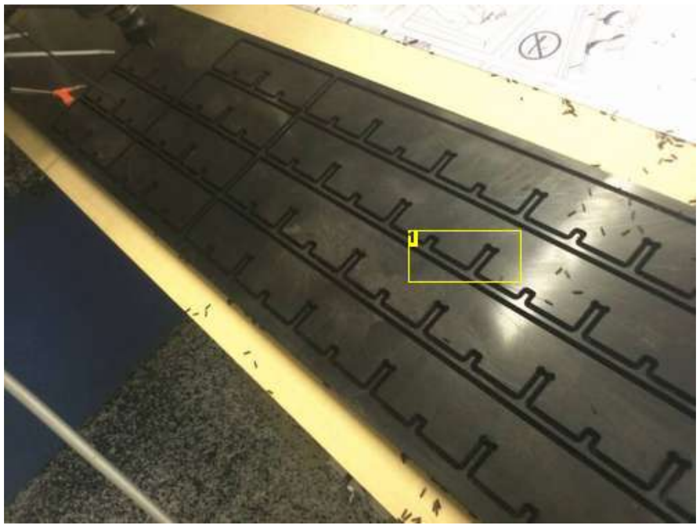


Step 2: Cutting Parts

After designing and seeing how things were going to start to fit together, I decided it'd be a good idea to start cutting parts. Luckily at work, I have access to a 5 foot by 10 foot CNC router, where I'm able to cut any types of plastics up to 2" thick.

From the CAD models, I was able to cut out the exact frame so that everything slides together and locks. I was also able to cut the top out of very thin ABS plastic sheet, which will give the 'rounded square' look once the LEDs are installed and lit up.

The reason for using black plastic is to try to keep this project as light as possible, while at the same time not allowing any light to go between boxes.



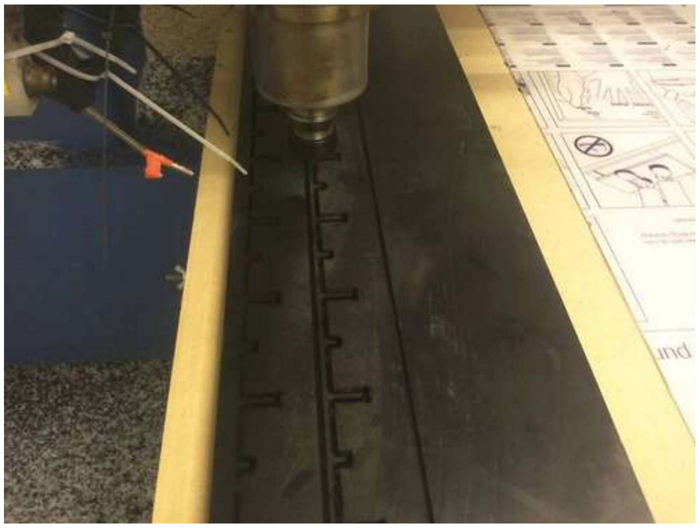


Image Notes
1. Interlock section and a groove for wires

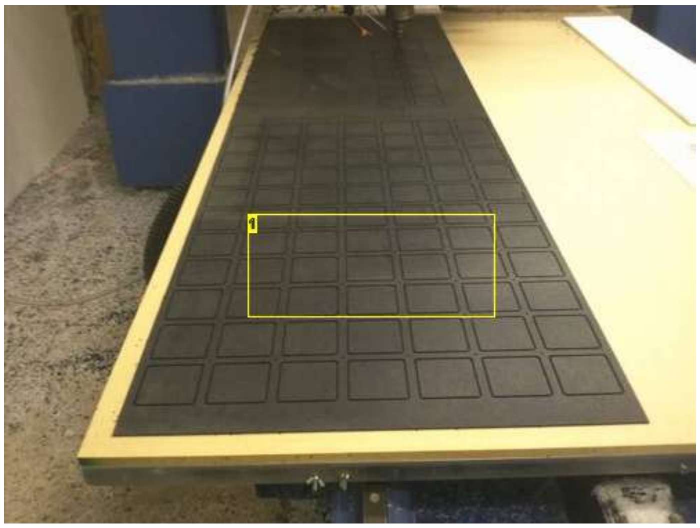


Image Notes
1. Top panel grid for the main portion of the stage.

Step 3: Fitting the floor

Once all of the floor pieces were cut out, assembly began of all the rails to check fitment and make sure enough pieces were cut. There are three main sections, the center of the stage, plus the two pieces that fold up.

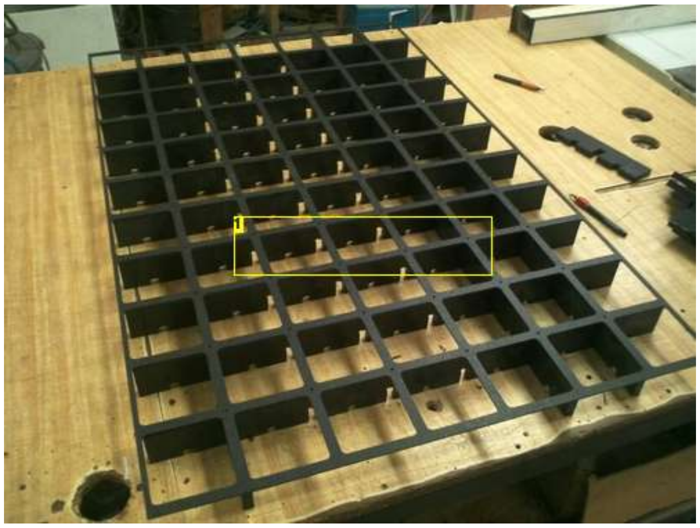


Image Notes
1. Making sure everything lines up correctly

Step 4: Getting LEDs ready

After cutting all of the parts, it's time to assemble LED modules for the floor. These specific LED modules have three SMD5050 RGB leds per module, and they are able to be controlled over an SPI interface. This makes for being able to change any module to any color at any time, and allows the most control for some really cool displays!

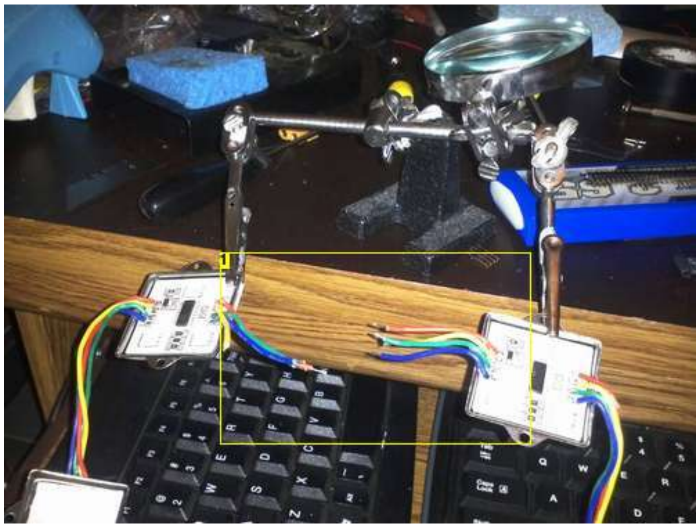


Image Notes
1. Four Wire interface for connecting, Power, Ground, Clock, Data

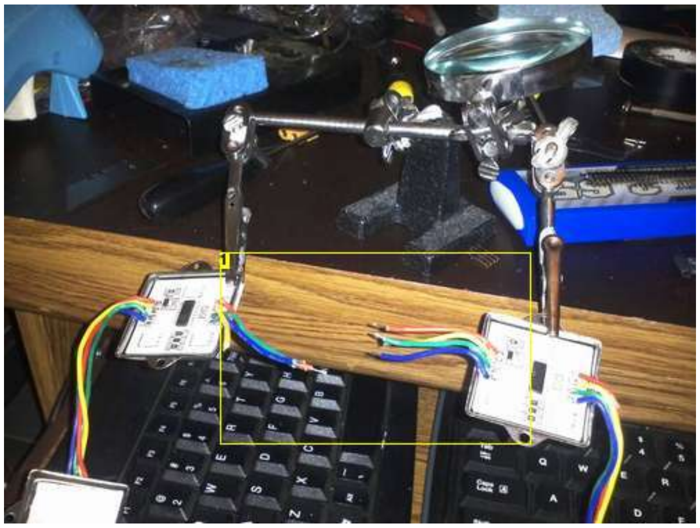


Image Notes
1. Light up test to make sure each strand works before installing!

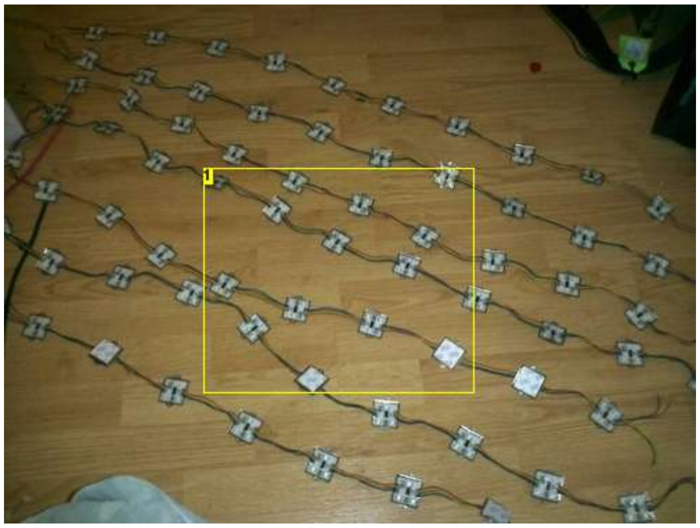


Image Notes
1. Almost all the strands complete

Step 5: Installing the LEDs

Once all of the LED strips were set up (the matrix was 11 x 11, so we built 11 strands of 11 LED modules), it was ready to start installing the modules. Luckily, these modules have 3M double sided tape on the bottom, so positioning them were very easy, but we did come back with some hot glue to make sure they stuck to the bottom panel.

The have to be wired all in series, so each row has to follow the row before it, and the 'flow' has to be correct, else you'll not be able to light up some of the LEDs. The way these LEDs work is by sending them a serial string of data, the first LED takes it's data off the top, and then sends the rest of the packet down the line, basically bit shifting the data stream. You can't individually address the LEDs, but knowing where they are in the data stream, you can change their data in the stream itself.

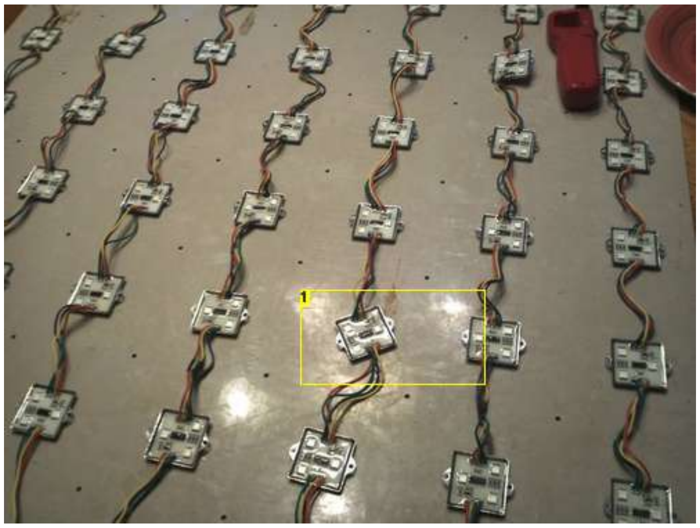


Image Notes
1. Sometimes 3M tape doesnt always stick, always best to go back and make sure they stick with Hot Glue or another adhesive!

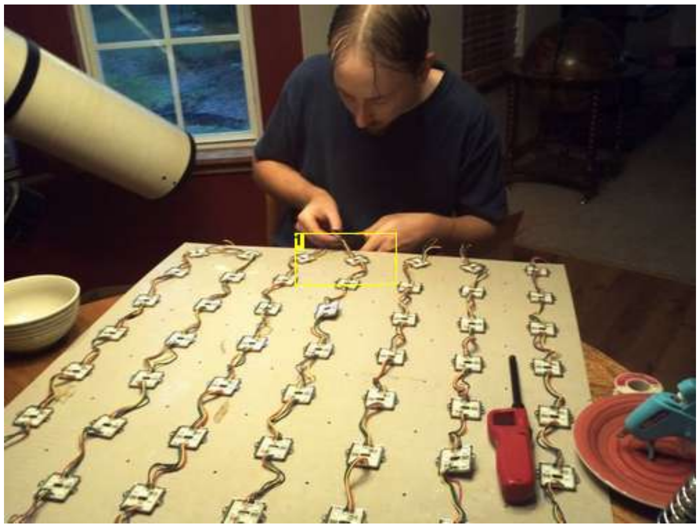


Image Notes
1. All of the LED ends have to be wired together, so once the strands are installed, they still have to be coupled together.

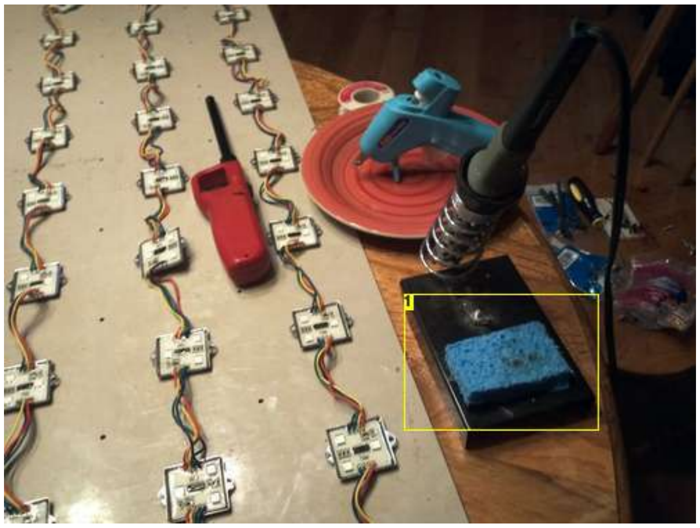
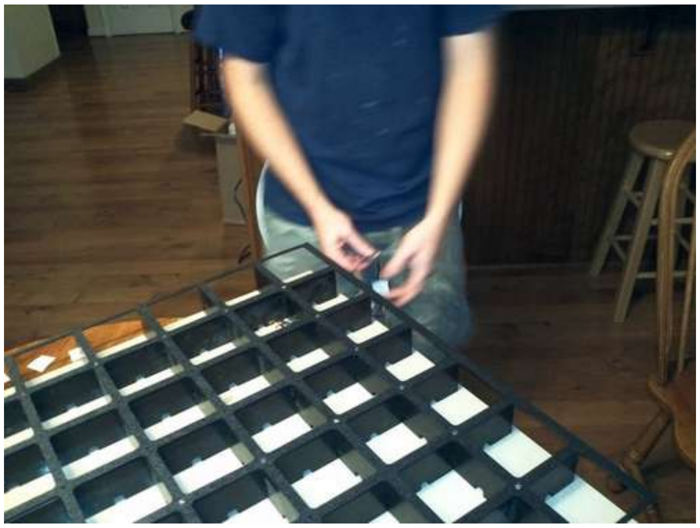
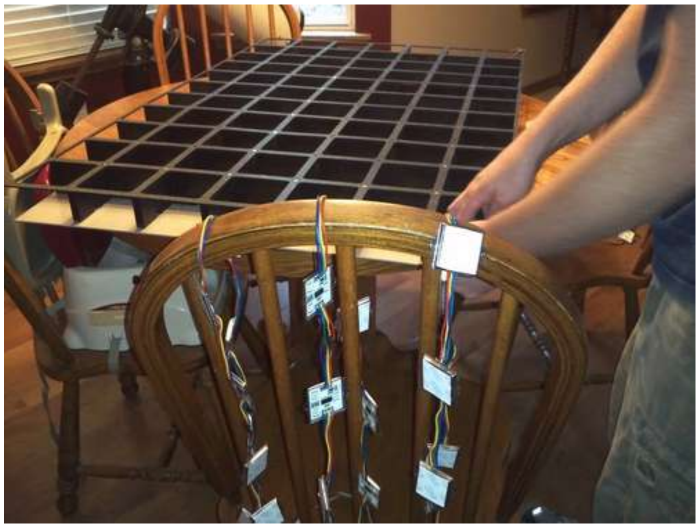


Image Notes
1. The tools of the trade, Soldering Iron, HeatShrink, a lighter for the heatshrink, and a glue gun!

Step 6: Adding the Frame

Once the LED modules are all planted on the bottom panel, it's time to over lay the cut frames to set up the light 'boxes' or 'pixels'. The holes in the bottom panel (and top panel) allowed us to be able to screw the top and bottom overlays to the rails themselves, and make the entire structure much more solid and to keep it from sliding apart.

You can see how the slots in the rails we cut out are perfect for running the wires between the boxes.

Step 7: LED Color Check and Testing

Once everything was wired and completely set up, it was time to power it up and check for colors and tracking.

For driving the LEDs, we're using a simple Arduino by outputting data out of the SPI channel. Most of what you see is just random algorithms to make sure the colors are in working order and all of the pixels are working.

The top piece is white translucent plastic, works as a great diffuse panel as one is needed!

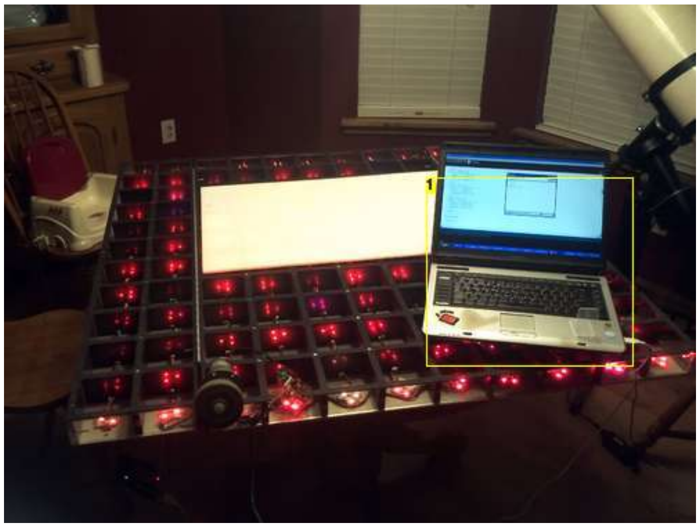
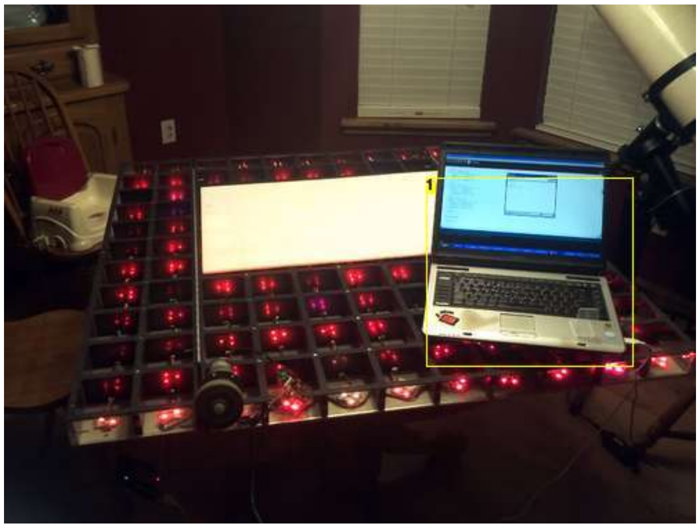
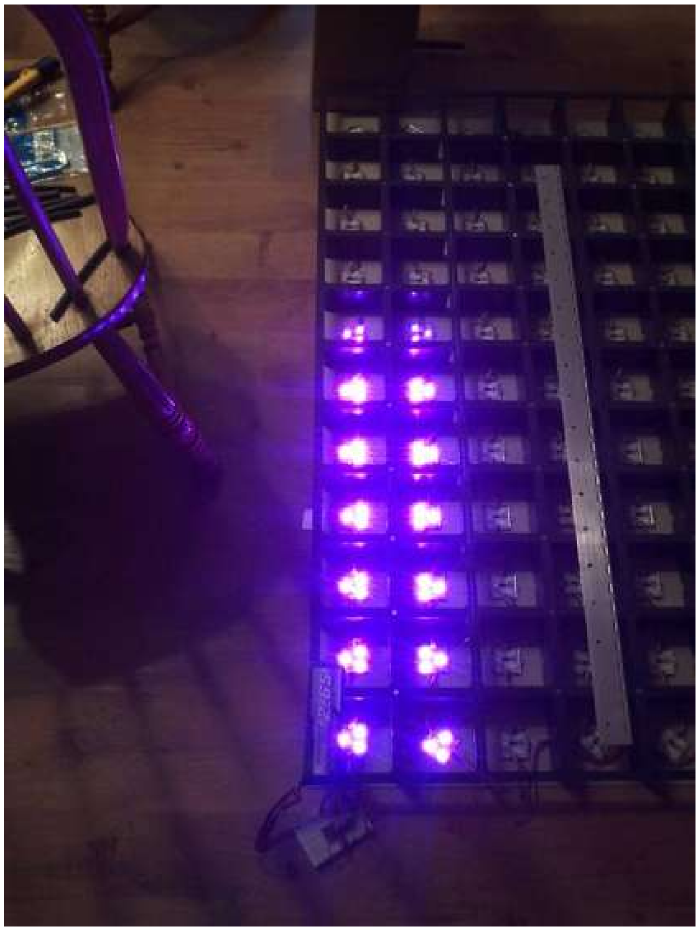


Image Notes



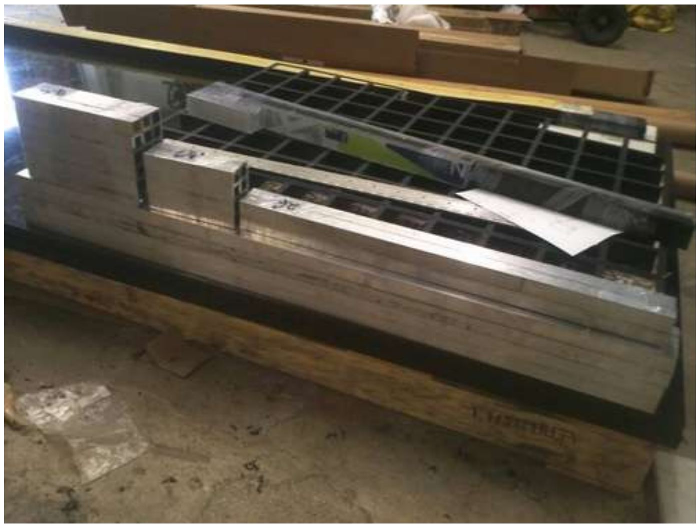
1. Arduino Programming



Step 8: Gathering More Materials

Once happy with the LED floor itself, it was time to start gathering materials to build the drive train and frames for mounting all of the rest of the electronics.

We picked up all of the aluminum (1x1x1/8" wall) tubing, and cut it up to the sizes that we needed for the frames. While doing this we also picked up all of the pneumatic components which I'll get into later in the build and explain WHY we need air cylinders on this project :)



Step 9: Frame Building

The goal behind the project this year was to be able to make the stage sit completely on the ground when 'in use'. The method to do that was to build essentially four frames. The main stage area frame was 28 x 44.5 inches, with a foldable wing on either side that were each 8.25" x 44.5". This allowed us to be able to drive through a standard 30" door opening once folded up. The drive system frame was built to fit inside the main stage frame, and by using drawer slides as linear rails, the entire stage could be lifted up or down and be allowed to be raised to drive around, and then lowered for stability for the dancers.

The next few steps will show the construction of these aluminum frames.

The original idea was to have all of the aluminum to be welded, but running out of time it was decided to use L brackets to bolt everything together. This seemed to be extremely strong and held together very well!

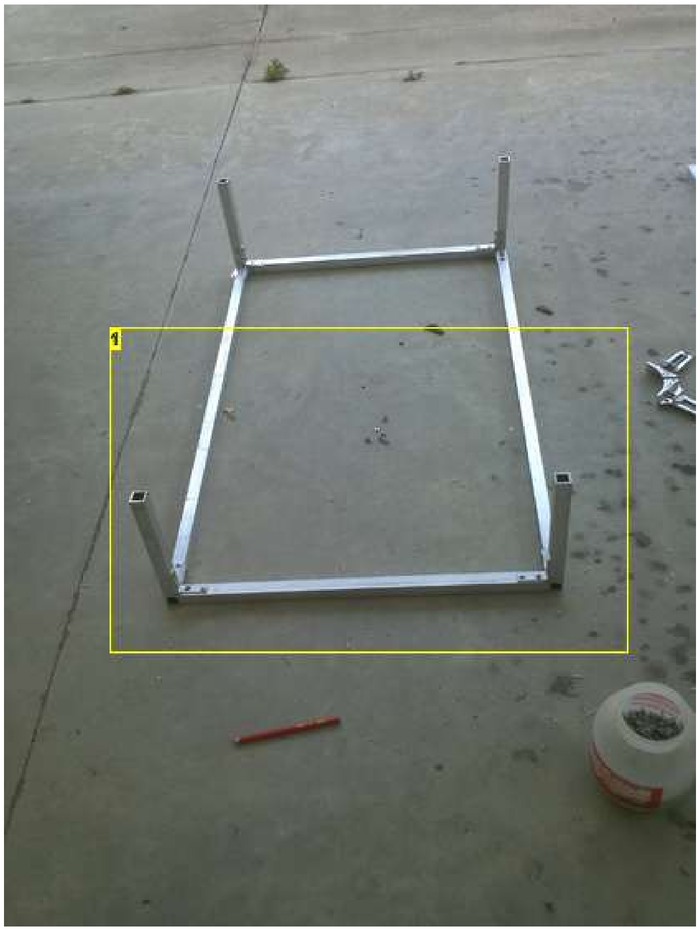


Image Notes
1. Main stage frame going together

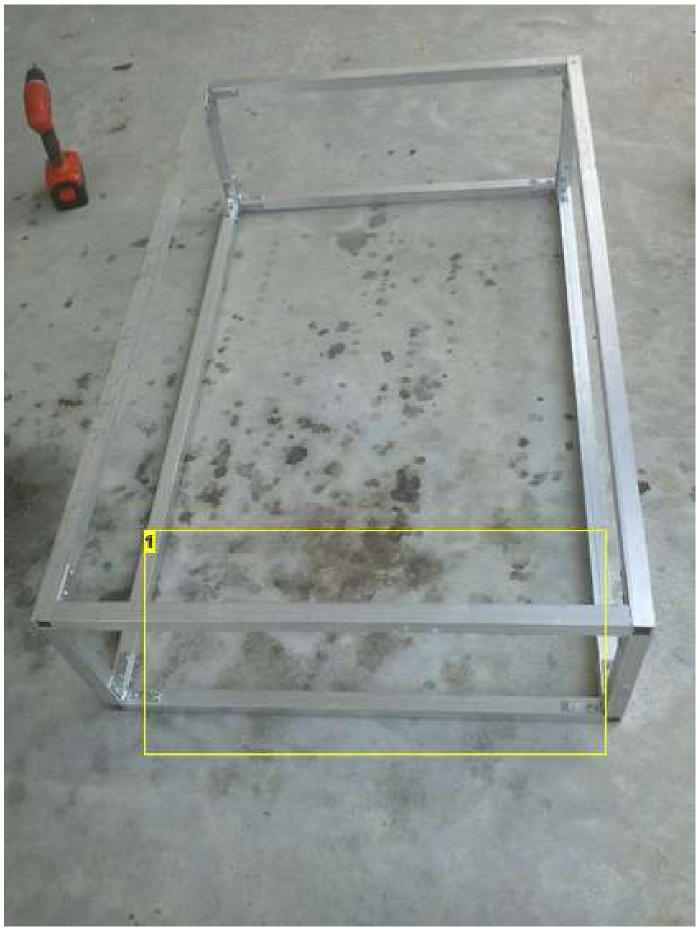


Image Notes
1. Main Stage Frame Complete

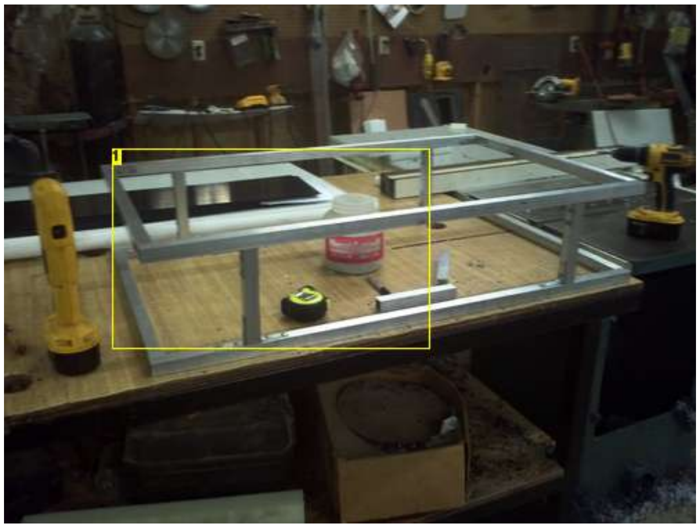


Image Notes
1. Drive train frame complete

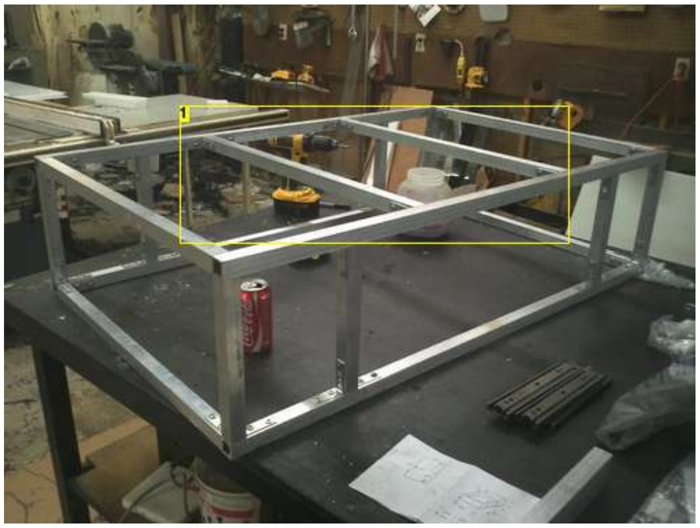


Image Notes
1. Extra rails added on top of the Stage frame to support the LED frame and dancers.

Step 10: Getting frames to fit...

Once all of the frames were built, it was time to get the two main frames sliding together, so the linear rails (drawer slides) and the pneumatics started going together.

In the video you can see how hard it is to control the air, I am using a standard blow nozzle and just shooting air into the input to make sure the frames will move and not be locked together. In the final version, I fixed the flow by adding in a flow restrictor on the solenoid input, make going up and down very smooth.

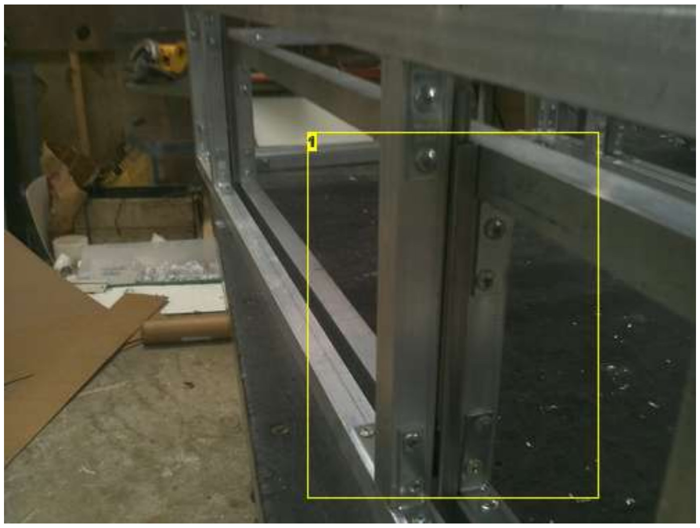


Image Notes
1. Linear Drawer Rails/Slides that were 8 inches long

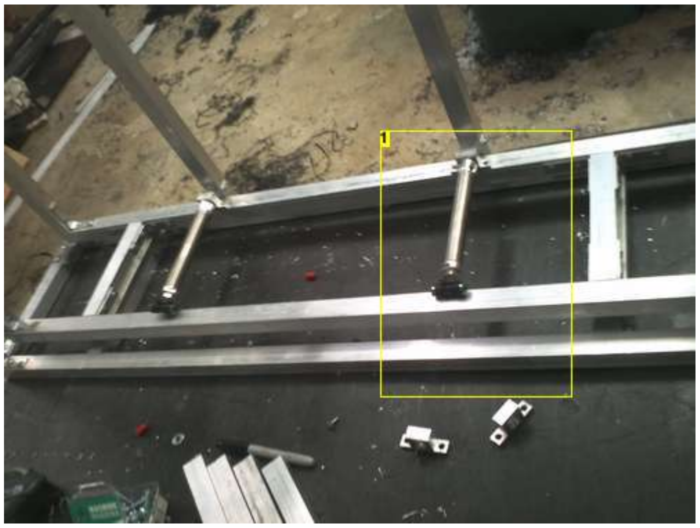


Image Notes
1. Air cylinders mounted and ready to be aired up

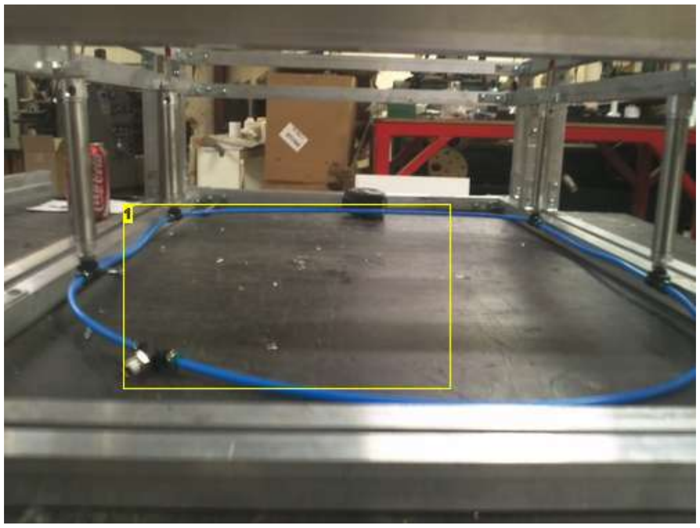


Image Notes
1. Air line installed for testing in the video.

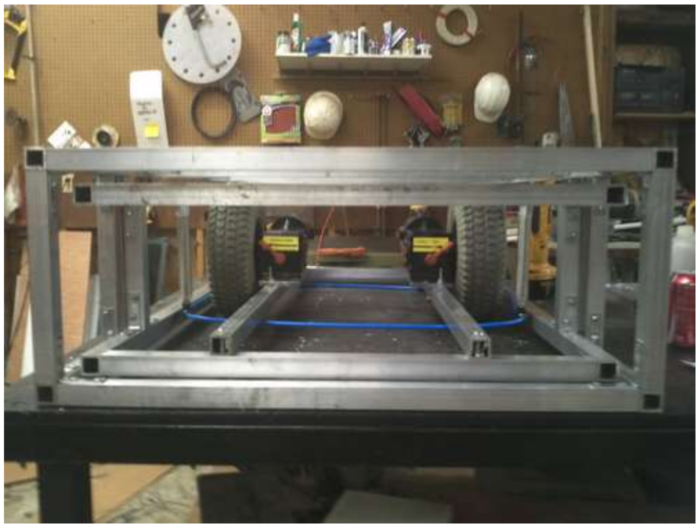
Step 11: Mounting Components

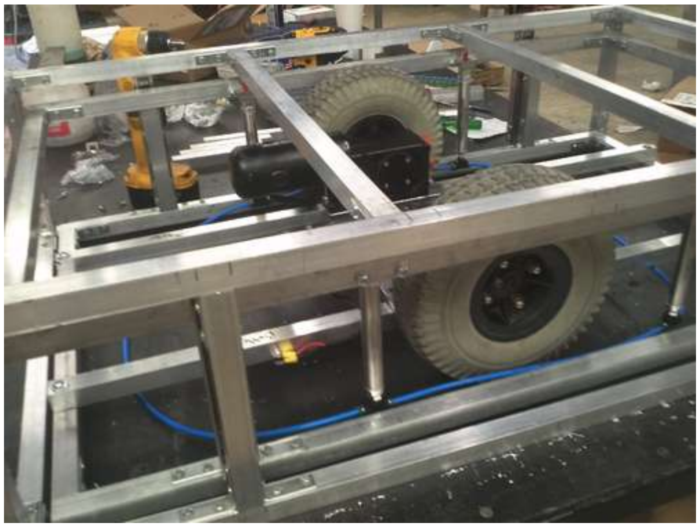
Now that we have the frames built and the air cylinders working together, it's time to really get down to business and start trying to figure out how to shove:

(2) Drive motors with 10" Wheels (From Electric Wheel chair)
Custom 10" Subwoofer Box
Amplifier for Subwoofer
Car Radio for powering mids/highs and taking computer input
Onboard PC
(2) 12v 35Ah SLA Batteries
Compressor
Air buffer tank
Electronics (Solenoid, drive speed controllers, Arduino, power switch, etc)

Now if you remember, the Main Stage frame was 28 x 44.5 inches, this means that the drive train frame was smaller, around 25x42" where all of this stuff has to fit. What are we waiting for?! Lets get to it!

We start installing by necessity. Obviously we need to drive around, so the motors and wheels get mounted first! Next is the batteries (can't forget those)..and then the next biggest item which was the subwoofer.





Step 12: More Mounting...

Once we had the big parts out of the way, it was time to start finding room for all of the smaller items. Also we can't mount anything in the center, because we have to leave room to be able to mount the pole!

With the onboard computer, we're running Windows XP and custom software along side of RoboRealm, which is a high customizable robotic software. I came across a 10" vga LCD and decided it wouldn't be a bad idea to slap it onboard too just incase debugging was needed in the field.

The computer itself is a Zotac Mini ITX in a custom case with 2Gb Ram, and 32Gb Solid State hard drive.

Compressor was picked up from Harbor Freight, just one of the small simple compressors that you keep in your car incase of a flat, plugs into the lighter.

We decided to use Victor 884 motor speed controls, these deliver plenty of amperage for the wheel chair motors at 12v.

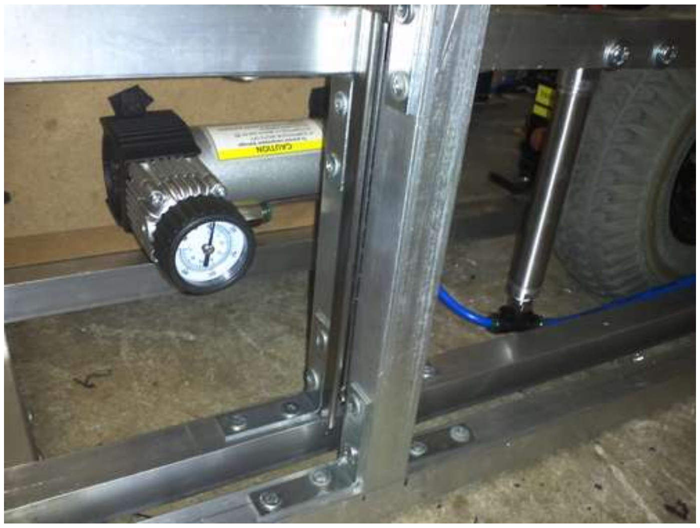
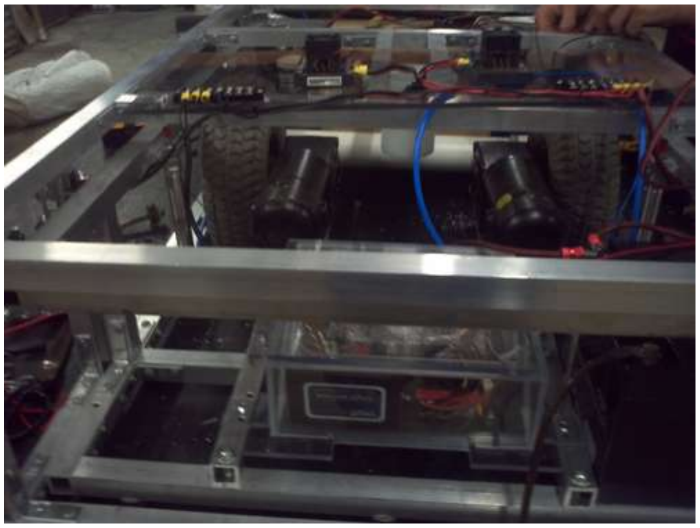
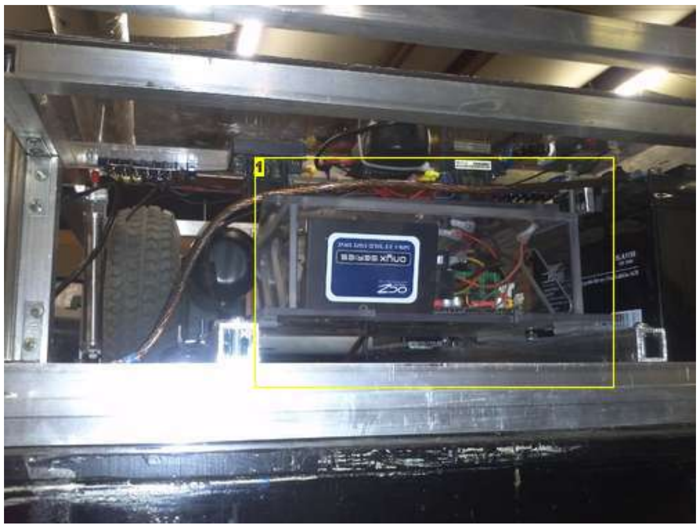
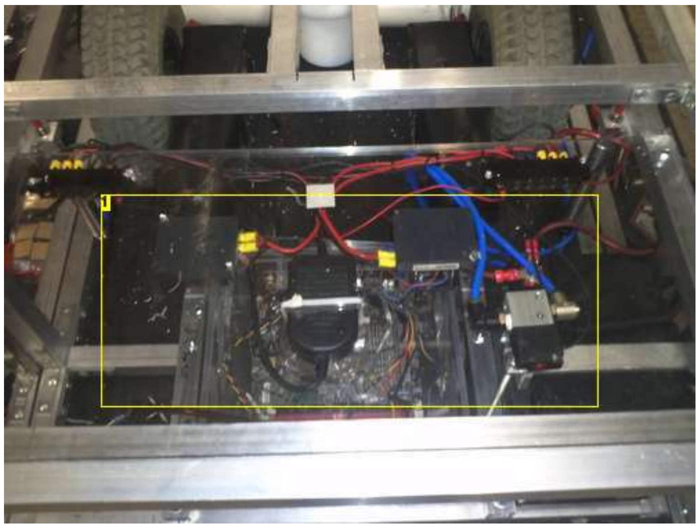





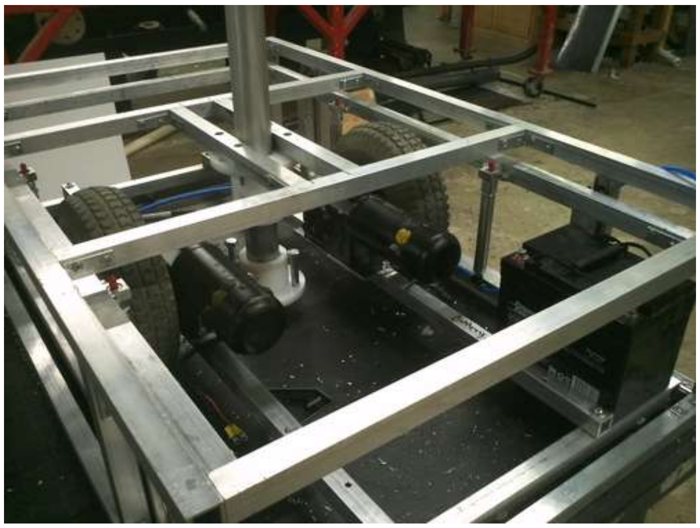
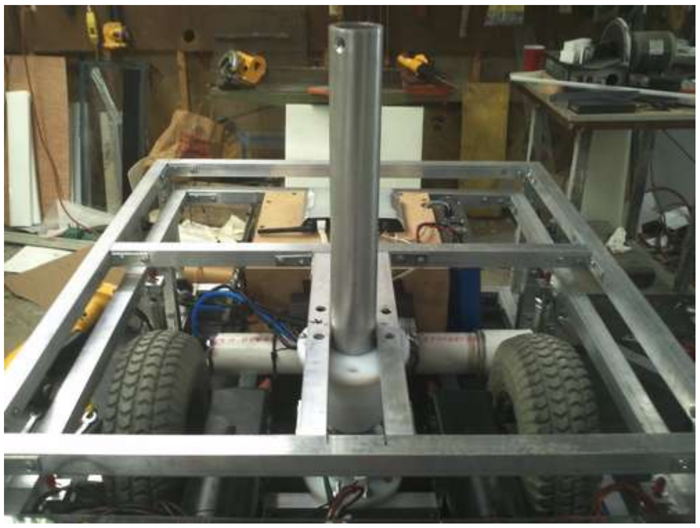
Image Notes
1. Zotac Mini ITX with 2Gb Ram and 32Gb Solid State Harddrive

Image Notes
1. Victor 884 Speed Controls 12v to 19v Converter for PC Standard 3 Position Air Solenoid Ground and Power Strips

Step 13: Pole Mounting

We can't forget about the pole! The pole is a two piece design, where around 20" is always mounted inside the Rover, and the other 70+ inches slides into a custom fabricated mount system.

The pole is mounted to the Stage Frame at the top and bottom for added stability.

Step 14: Finishing the Electronics...

After installing all of the drive parts, it was time to install the arduino, start securing some wiring, and run air lines to get ready for a drive test!

We also mounted the WiFi router which allows us to access the onboard PC by just a simple HTTP server. This allows any cell phone with a browser to go to the Rave Rover webpage being served, and change music, the LED modes, and in the future more stuff!

For Audio control, we feed the Audio line out into the AUX line in on the Car Radio. Using the Car Radio saved a lot of time as it has a built in 4 channel amplifier which was perfect for mounting the four external speakers.

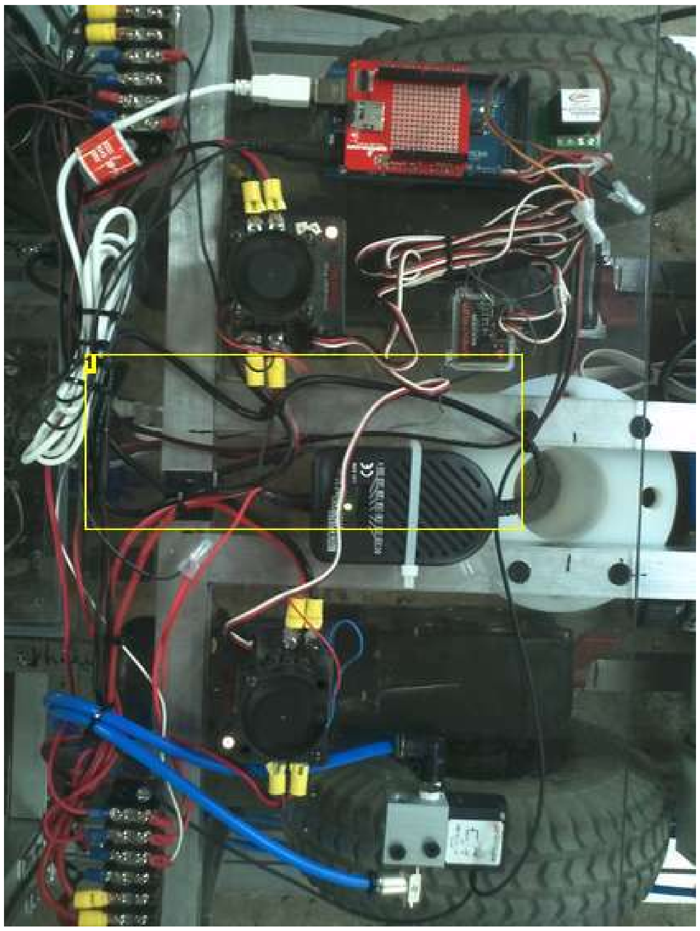


Image Notes
1. Electronics mounting complete!

Image Notes
1. Zoom Player is great for its TCP interface to be able to control it over a network.

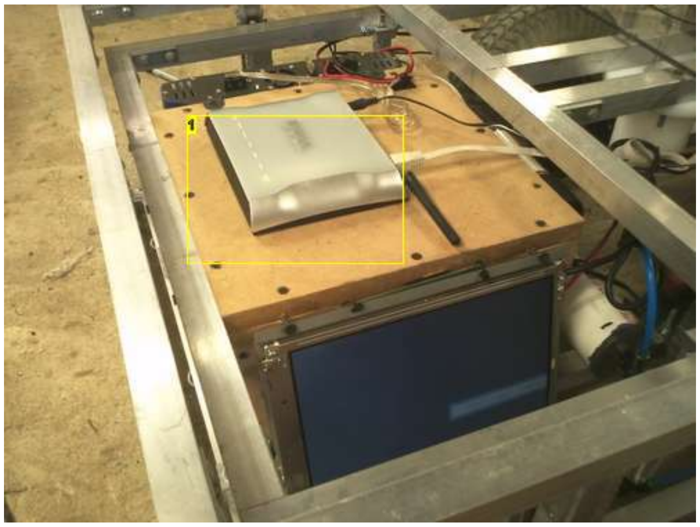


Image Notes
1. Wireless Router Mounted

Step 15: Drive Test!

Once things were secured, a power check was in order, and then a drive test. The control system is a standard Spektrum DX6 Radio Transmitter and Receiver, you can find the newer models from places like http://www.robotmarketplace.com or http://www.towerhobbies.com

Step 16: Installing Floor

Now that the frames were built, everything mounted, and we were able to drive, it was time to mate the pieces together that we built earlier. Time to install the LED floor and get some music going! I'm getting anxious to party!

The floor fit perfect, and just using some simple self tapping screws, it was a super fast job of attaching it to the aluminum frames.

Hinges were mounted on either wing to allow that section to fold up to retain the size of being able to go through a standard 30" door.

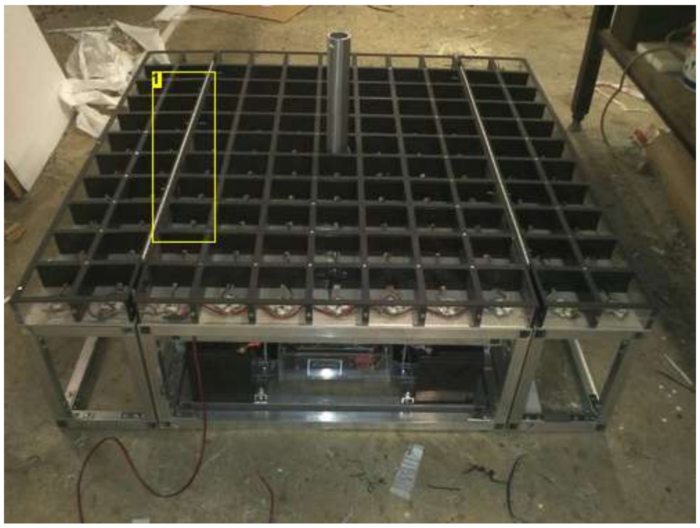


Image Notes
1. Aluminum Piano Hinges for Folding

Image Notes
1. Power Test, it works!

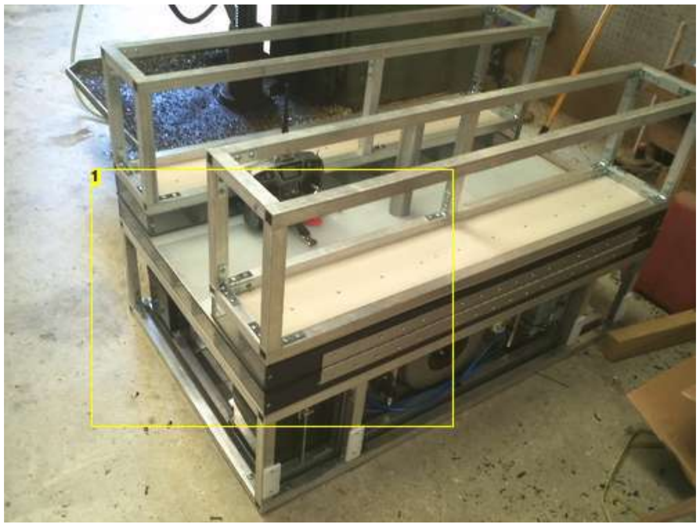


Image Notes
1. Once folded, it can easily be tucked away in a corner.

Step 17: Final touches

You guys ready to party? I know I am, but first....

Gotta make it pretty!

It was time to cut some side panels, and install the speakers. For the front and rear panels, we wanted to make sure everyone knew 'Rave Rover' when they saw it.

We took some acrylic mirror as a backer, installed some stand off rails, and a front clear piece of acrylic with a vinyl decal with the Rave Rover logo cut out. Trying to find a diffuser element was tough, but tissue paper worked perfectly for us. The inside of this box was lined with green LED strips so that when power was on, the logos lit up bright!

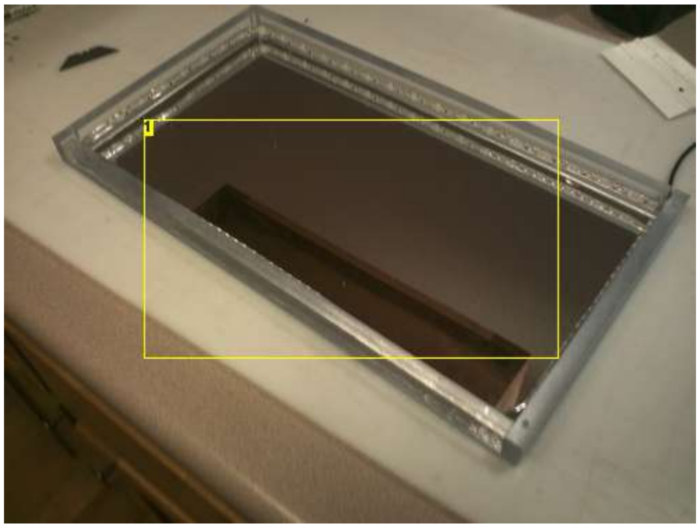


Image Notes
1. Mirror background, LED strips around the edges.

Image Notes
1. Vinyl decal with letters removed.

Image Notes
1. All layered together

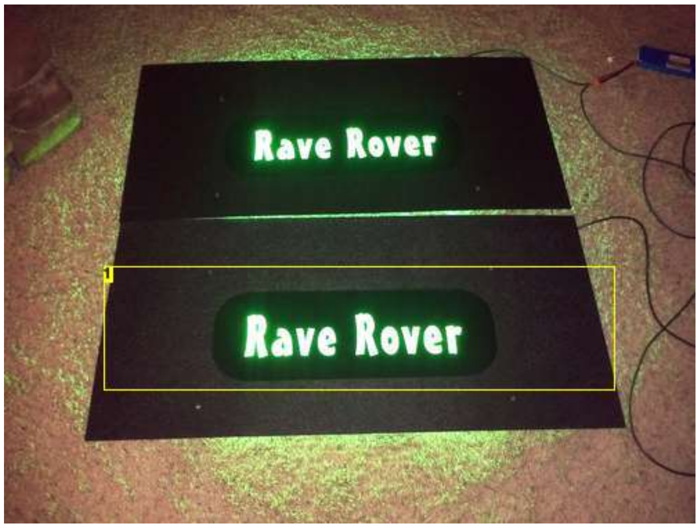


Image Notes
1. Lit up with front panels overlaid.

Step 18: Speaker Install

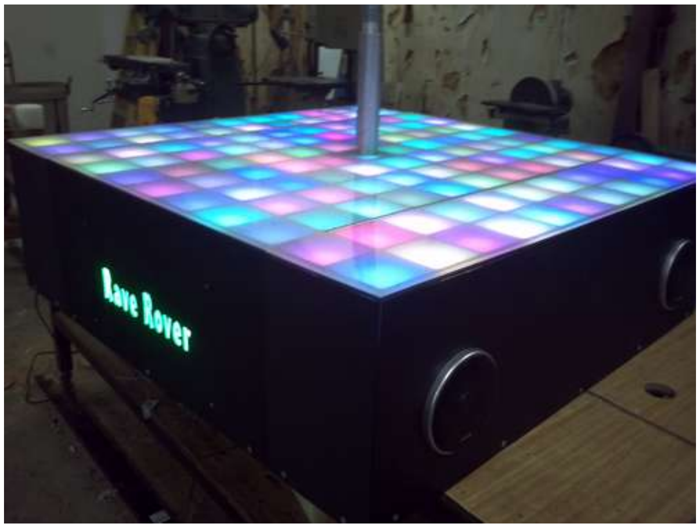
Time for speakers! Simple install of just screwing them in, and wiring them to the car radio. For this build we used Fusion 5.25" 2 - Way Car door speakers, cheap enough not to be worried about being kicked.

Step 19: Finally Done!

Everything is finished, and we're ready to party....

First party is Dragon*Con 2011, the next step will show you some pictures that we got from the event! For now, check out the modes of Rave Rover below.

Yes, the audio IS coming from the rover itself....



Step 20: Where to find parts...

Just wanted to give you guys a source list for some of the parts that I managed to pick up:

www.ebay.com -> Sourced Wheel Chair Motors
www.surpluscenter.com -> Sourced Air Cylinders, Pressure Swtich
www.arduino.cc -> Best source for Arduino References
www.robotmarketplace.com -> Anything Robotic!
www.hobbyking.com -> Sensors, Radio Control Gear and More
www.towerhobbies.com -> Radio Control Gear
www.pololu.com -> Sensors
www.sparkfun.com -> Arduino!
www.tigerdirect.com -> Sourced Onboard Zotac Mini ITX Motherboard and Harddrive
www.interstatebatteries.com -> Sourced Two 12v 35Ah SLA Batteries
www.mscdirect.com -> Sourced Air line, bolts, nuts, and other small hardware
www.harborfreight.com -> Sourced 12v Air Compressor!

I'll add more as I remember them!

Step 21: Party Time!

Here's just a few images from Dragon*Con 2011, for more, check out our website at http://www.raverover.com

You may notice some of our friends in the background...Yep, that's Bar2D2 and Marc DeVidts, be sure to check out their webpages too!

http://www.uiproductions.com

http://www.jamiepricecreative.com/bar2d2.html

We hope you enjoyed our instructable! This project was a lot of fun to do, and we can't wait for the next event.

Related Instructables

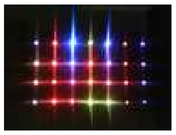
Easy Autochanging RGB Dome Lamp by powerman666

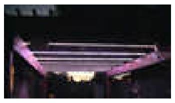
Cyber Mask Mod by neologik

Installing 12V Color Changing RGB Lighting by Oznium

Carlitos' Projects: Wireless Speech-Controlled Arduino Robot by RobotShop

How to Make a Super-Bad Rave Mask (LED) by gardnsound

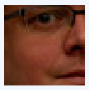

Sound Activated 4 X 7 RGB LED Matrix by jwflammer


The Lightning Simulator/Breatha Equalizer - Arduino Powered by alinke

Raving out your Computer by itstemo1

Type Case, the making of a low-resolution display

by Martin Bircher on July 16, 2011



Author:Martin Bircher   author's website
I am an artist, son, electrician, student, lecturer and uncle.

Intro: Type Case, the making of a low-resolution display

What this text is all about
What I describe here is how I got from an idea to a presentable art piece and I don’t want to write up a construction manual: How to build your own “Type Case” installation.

Abstract
The installation "Type Case" is a low-resolution display with 125 rectangular pixels of different sizes. These are formed from the reflecting light of digitally controlled LEDs, embedded in each section of a European printers' type case. Due to the standardized fragmentation of its compartments, the density of visual information is decreased towards the objects' centre. Viewed close by, it is nearly impossible to recognize more than a flicker – however after moving some distance away, it becomes distinguishable, that the lights and shadows give a representation of the latest headlines.

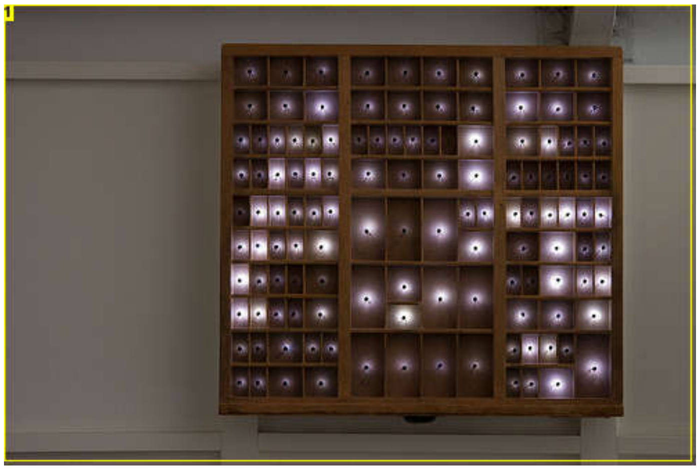
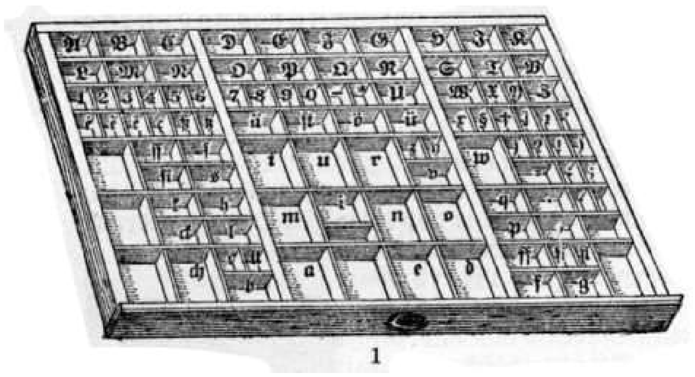


Image Notes
1. The ready "Type Case" installation, displaying recent newspaper headlines.

Step 1: The idea

On a late winter day last year, I looked at a picture of a European type case. I was fascinated of its odd segmentation and got interested whether it was possible to use the case as a display with different sized pixels.
Most of the ideas I get, I forget as fast as they came, but I immediately had a good feeling about this one.



Step 2: Simulations

I've learned it the hard way a few years back: It is better to make some mockups and visual tests before jumping off. Simulations have many benefits. First of all they can be changed more easily then the real deal and therefor are better suitable for experimentation. It can be seen faster whether something might work or not and after everything is right, a simulation is a proof of concept and can help to get funding for a project.

I did some compositions in Photoshop to see what the 125 pixels are be capable of. It became clear that it was not possible to recognize images, with so few pixels – text on the other hand worked somehow. Since it is not so interesting to display static letters I needed to get going with an animation. I love to use Processing for all kinds of things and therefor it was the tool of choice.
My first test in Processing was a conversion of a movie to the 125 pixels. Here is a flick which shows a Betty Boop cartoon, the Processing simulation and a later recording of the real thing.

(The original Betty Boop cartoon from 1933 is in public domain and not copyright protected anymore. Source: archive.org)
I think that this shows quite well the limitations due to the very few pixels.

After that I moved on to animate text. The simulation fetched resent headlines from an online newspaper and displayed them as scrolling text. The bigger part of the Processing code was then later used for the real object, which after this tests, I was confident to build.

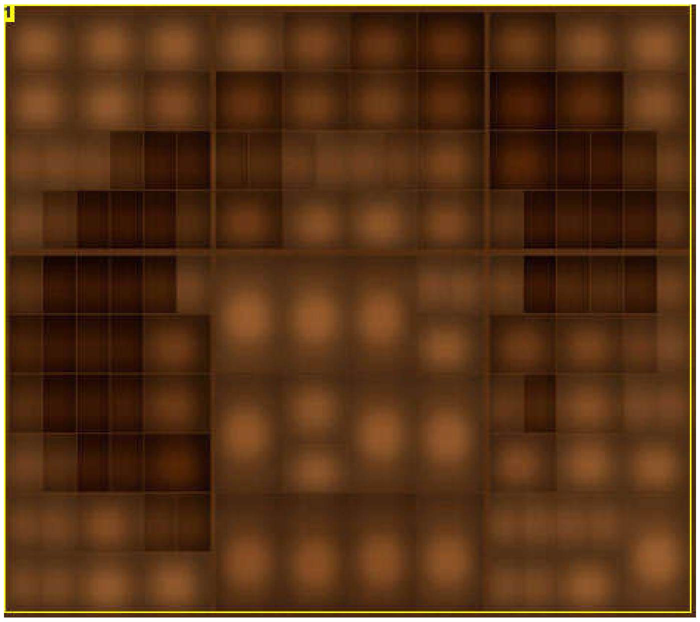


Image Notes
1. Photoshop simulation: Movie star with hat.

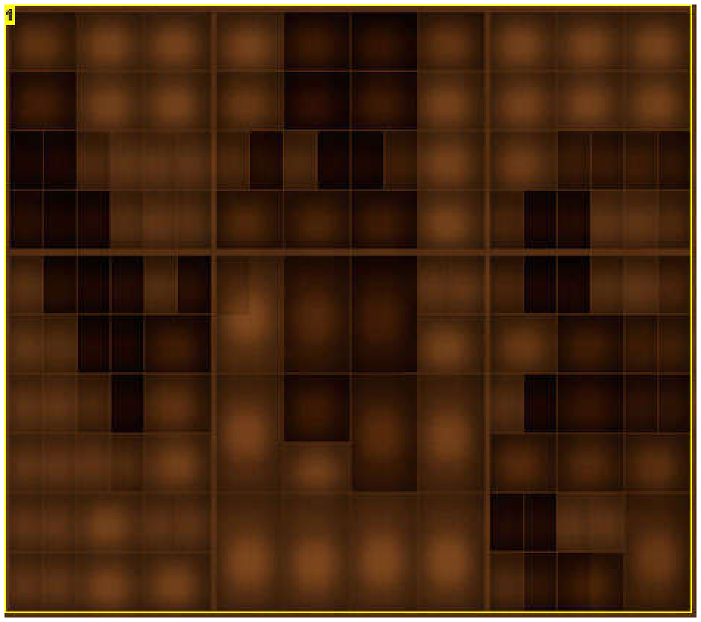


Image Notes
1. Photoshop simulation: Some simple text.

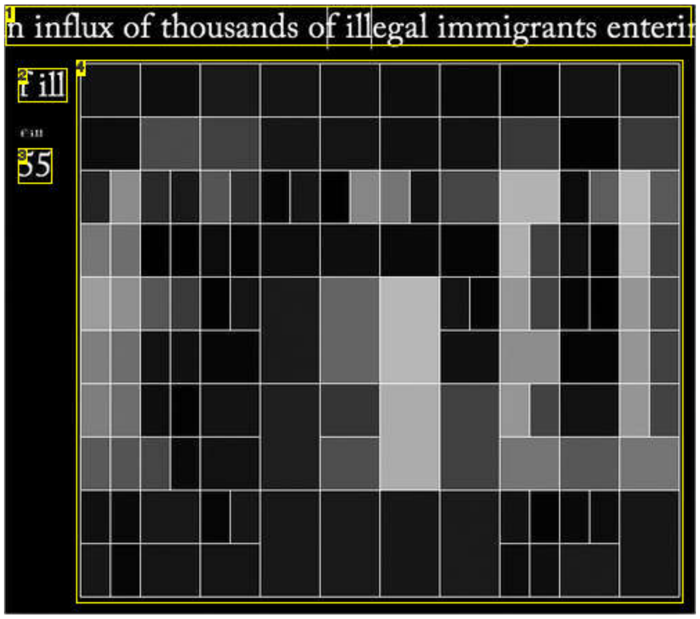


Image Notes
1. The headline.
2. The actual displayed part of the headline.
3. The resent frame rate.
4. The brightness values of the according compartments.

Step 3: Development = solving problems

After having the visual proof of concept, I had to solve a huge amount of problems. To be still standing at the beginning and being confronted with so many questions can be discouraging and stressful. I usually try to break down the big problem into smaller ones, which aren't so complex anymore. After this process I ended up with the following partial questions:
How will I be able to use the compartments of the case as a pixel?
What light sources will I use?
How can I control this light sources and their brightness?
How will I distribute the task between computer and microcontroller?
How will I mount the hardware on the case?
How will I mount the case?

Fortunately I did not have to find answers to everything yet and started with the electronic: I decided to use Arduino and TLC5941 LED drivers. After some tinkering and reading thru many different forums I had a working solution.

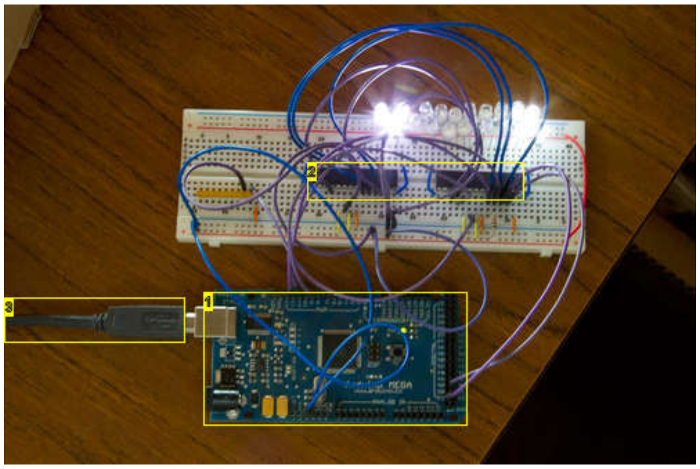


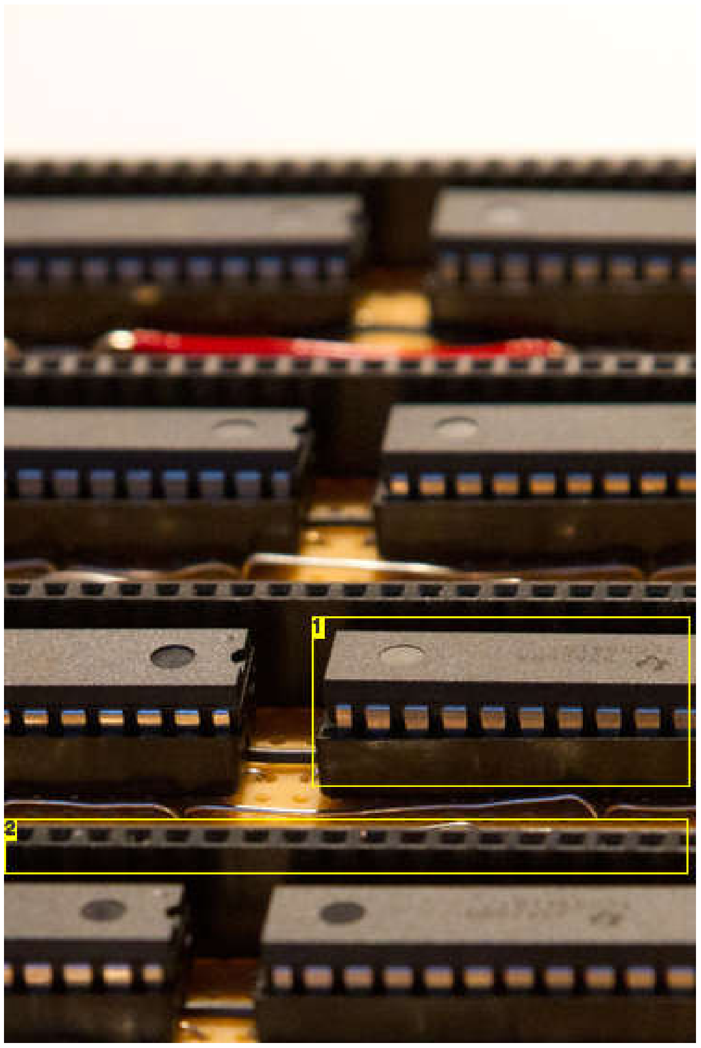
Image Notes
1. Arduino MEGA
2. 2 TLC5941
3. Serial data from Processing

Step 4: The build

Building the object was then just a question of time and endurance. Looking back I calculated, preparing, mounting and soldering one LED took about 20 minutes. Adding this up for 125 pixels we are talking a Swiss working week of 42 hours.

Here a small movie about the assembly process:

Now it was finished!



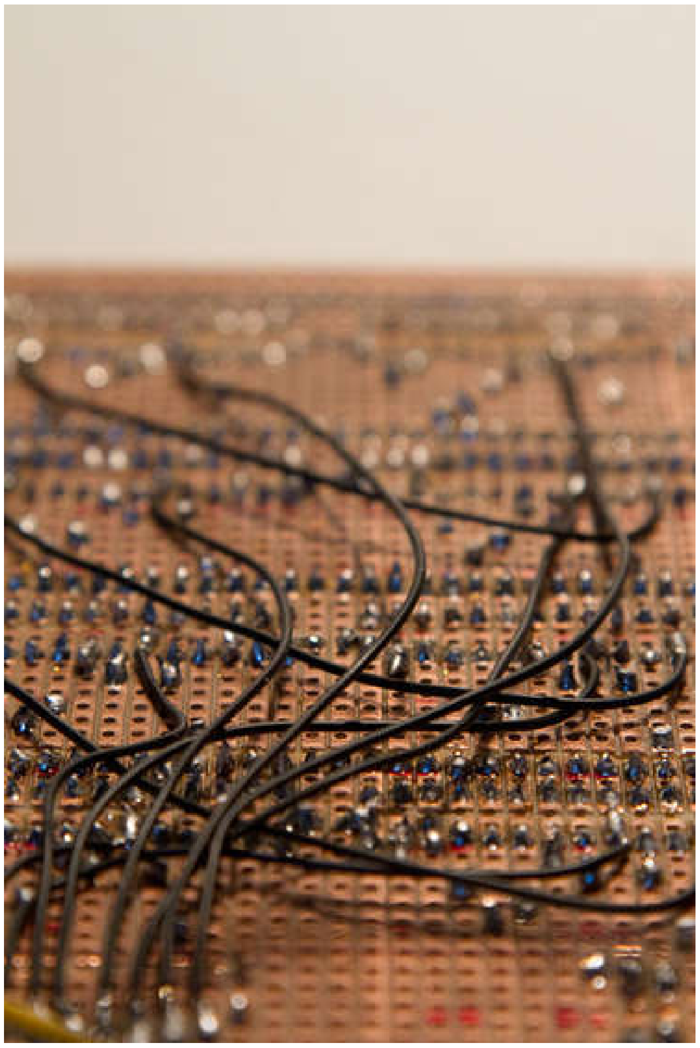
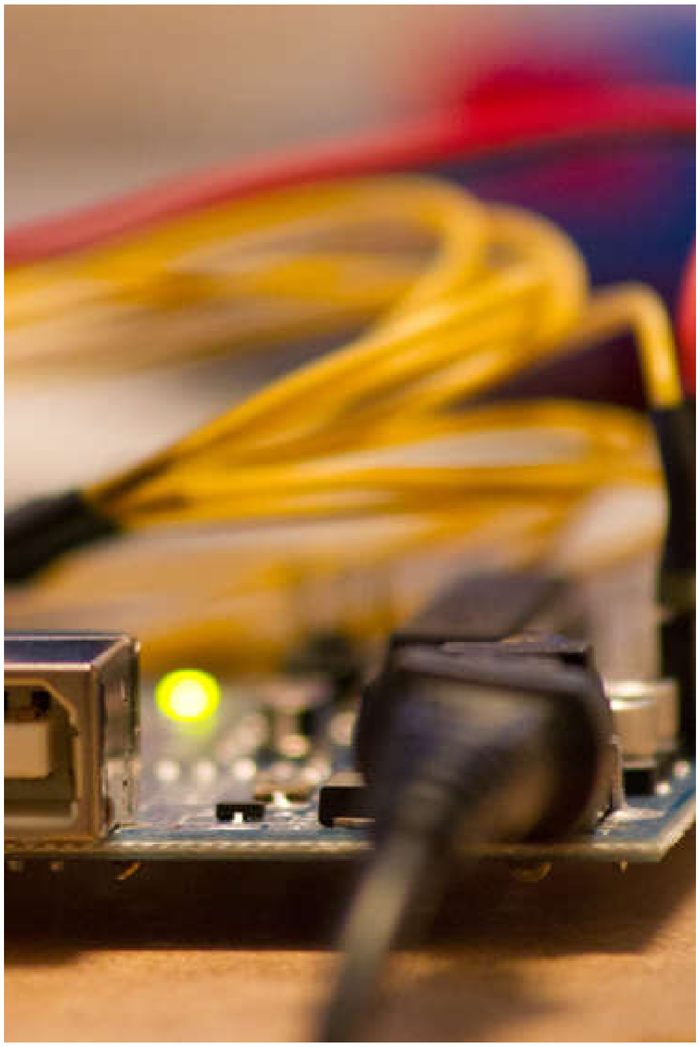


Image Notes
1. TLC5941 LED driver.
2. Outputs to LEDs.



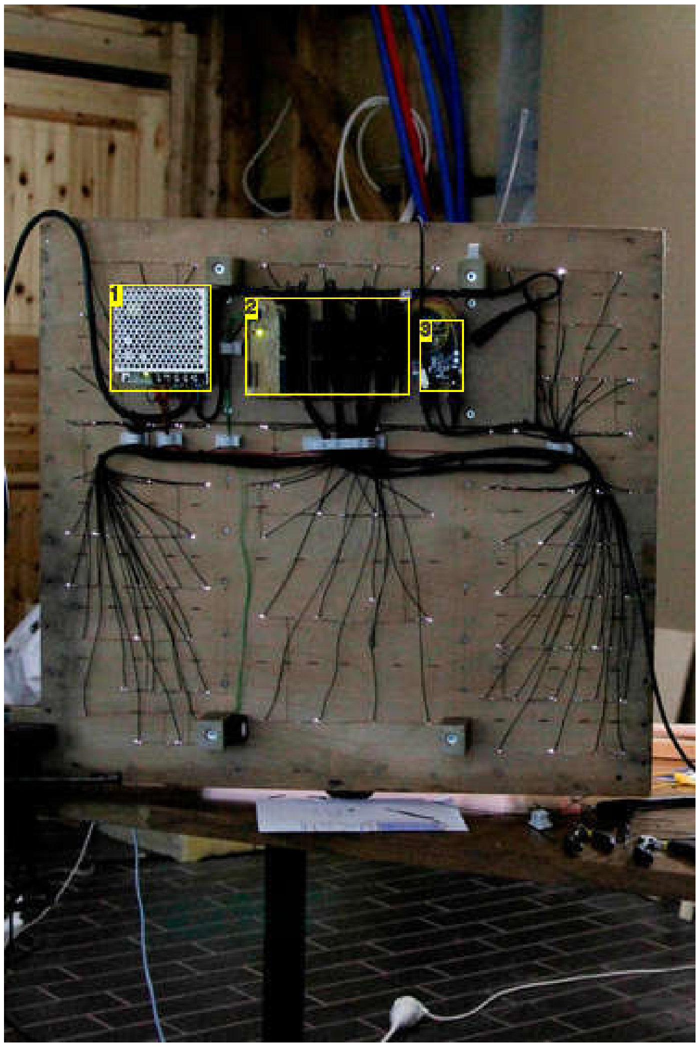


Image Notes
1. 5V power supply.
2. Driver electronic.
3. Arduino.

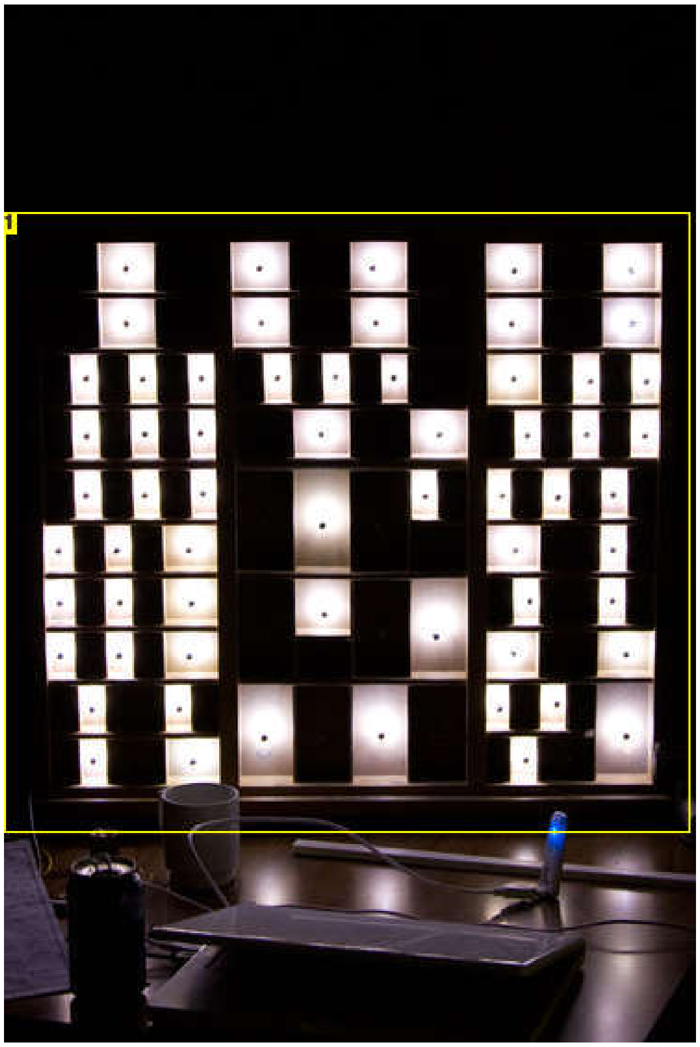
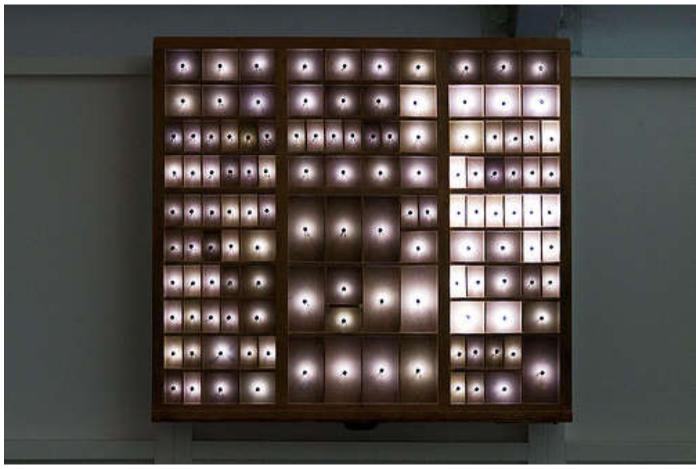


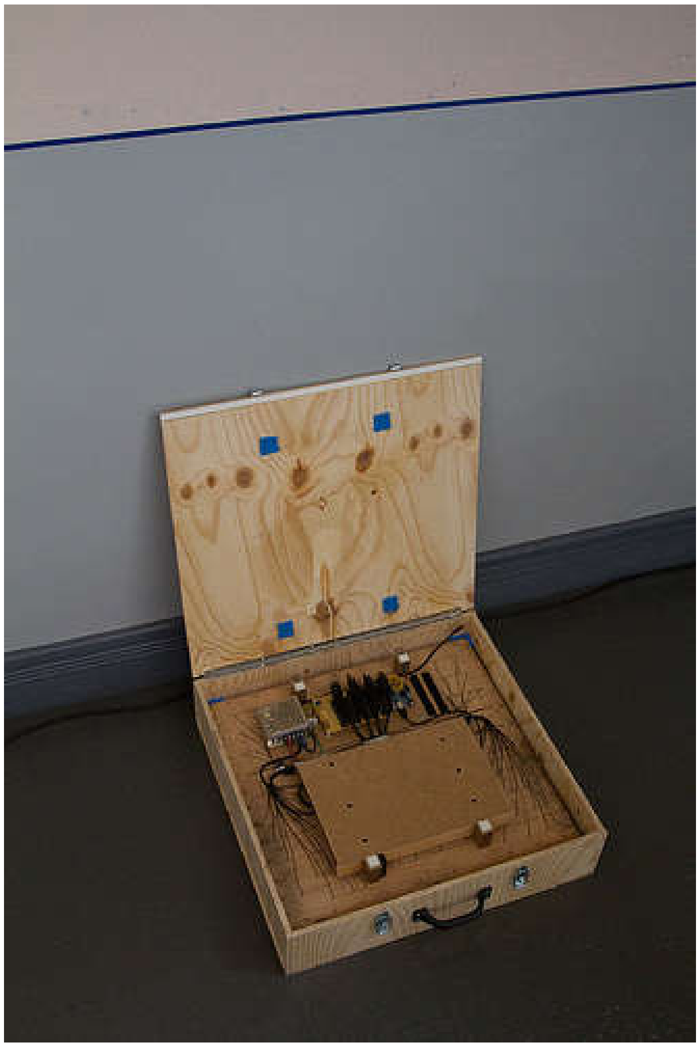
Image Notes
1. Running a test pattern.

Step 5: The documentation process

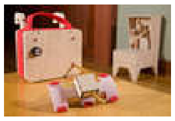
As important as not just talking – but doing something, it is to be talking about what you do.
I took loads of pictures during the process and after it was done, I took movie clips and photographs for almost a day. I wrote texts and edited the small film which can be seen in the beginning of this instructable.
Last thing to do was to update my own website and upload files to Vimeo , YouTube and flickr .



13
175
8
Post-Versal



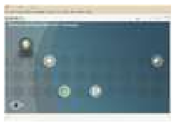
Related Instructables


Foot Tap Amplifier by mkontopo

TurtleArt Turtle by IT_Daniher

Smallsword Choreography Shirt by grossmr1

How to have fun with Arduino (and become a Geek in the process) by john otto


How to get started with Eclipse and AVR by andy

Public Window with HTML5 by PublicWindowHSL

Office Phone by bdddbbd.b

Sigh Collector by mkontopo

Sigh Collector

by **mkontopo** on March 2, 2009

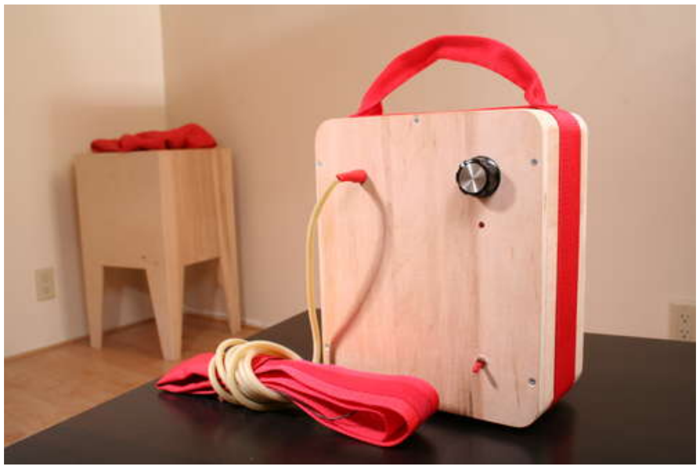
Intro: Sigh Collector

Sigh v. i. [imp. &p. p. {Sighed}; p. pr. & vb. n. {Sighing}.]
1. To inhale a larger quantity of air than usual,
and immediately expel it; to make a deep single
audible respiration, especially as the result or
involuntary expression of fatigue, exhaustion,
grief, sorrow, or the like.
[1913 Webster]

Description:
These are instructions for building a home monitoring system that measures and 'collects' sighs. The result is a physical visualization of the amount of sighing, for personal use in a domestic environment.

The project is in two parts. The first part is a stationary unit, which inflates a large red air bladder upon receiving the appropriate signal. The second part is a mobile unit, worn by the user, which monitors breathing (via a chest strap) and communicates a signal to the stationary unit wirelessly when a sigh is detected.

Assumptions:
1. You have a basic understanding of construction and fabrication techniques,
as well as access to the appropriate tools and facilities.
2. You have a working knowledge of physical computing (reading circuit diagrams)
3. You are overwhelmed with the anxiety of living in a failing state, and frustrated
that most of your household objects address only physical rather than emotional health.



Step 1: Material Needed

Here is an overview of the materials that will be needed.
Each individual page has more details and links on where you can purchase some of these materials.

Physical Materials:
> 1, 4x8 Sheet of Plywood. I used a piece of shop-grade maple ply.
> 2, 2x2 for the structural frame
> ~2 yards of red nylon strap fabric
> Some loose red fabric from a fabric store
> Latex tubing (Inner Diameter: 1/8", Outer: 1/4")
> Wood Screws (5/16, 3", 4")
> 1 Rechargeable battery powered air pump (Coleman Rechargeable Quick-Pump)
> 1 unidirectional "Check Valve"
> A piece of a garden hose
> Liquid Latex & Red Pigment, or a large red balloon of some kind.

Electronics, Misc:
> 1, 20cm Stretch Sensor
> 1 red RCA cable, Male and female headers
> 1 10K Potentiometer with large sized knob
> 1 3-way toggle switch
> 2 Arduino Microcontrollers (Diecimille or newer)
> 2 9V battery clips with 5mm (center positive) male jacks.
> 2 xBee wireless modules
> 2 xBee shiels from LadyAda
> 1 FTDI cable for programming the xBees
> 1 LMC662, "rail-to-rail" OpAmp chip
> Misc Electronics components (see circuit diagrams for details).

Step 2: Build and Program Circuit. Hack into Air Pump

I like to start by getting the electronics working first, usually with a prototype of what I want to build (made from cheap exterior plywood, or even cardboard and hot-glue).

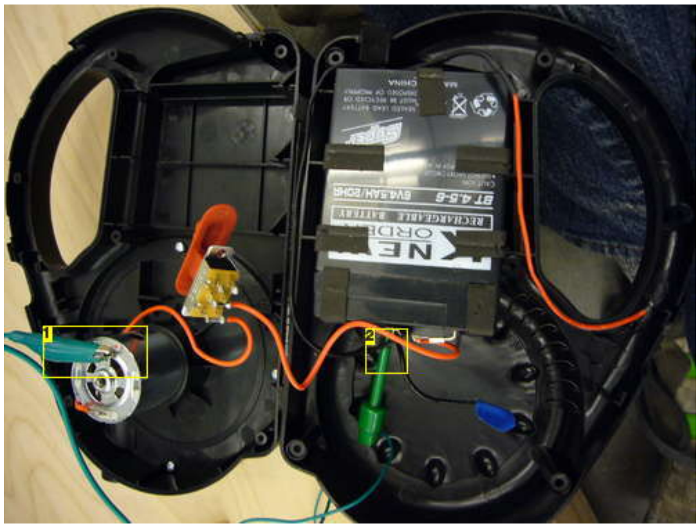
The electronics are divided into two parts. This part is the receiving end. It will receive a wireless signal from the wearable unit and use that signal to turn an air pump on for ~2 seconds and then turn it off.
Between the pump and balloon, is what's called a check valve , which lets air pass one direction but not the other.

The air pump is a Coleman Rechargeable "Quickpump" . I like it because of the rechargeable battery, and the different sized nose attachments.

Open up the pump and rework the toggle switch, so that it's bridging between the battery and one terminal of the motor. The other terminal of the motor will run to the collector of the TIP120 transistor. To do this, you'll have to de-solder the black wire from the second motor terminal, and also de-solder the lead coming from the battery charger and going to the other end of the toggle switch. Be sure to common ground the motor's battery with the arduino's power supply.

Build the circuit in the diagram below. There is also a PDF attached for higher resolution.
Program the arduino with the code supplied in the text file. You'll need to install this library .

If you don't know how to work with Arduino, here are some references so you can learn:
> Main Arduino Website
> Freeduino -- Repository of Arduino knowledge and links
> NYU, ITP's in-house physical computing site with tutorials and references.



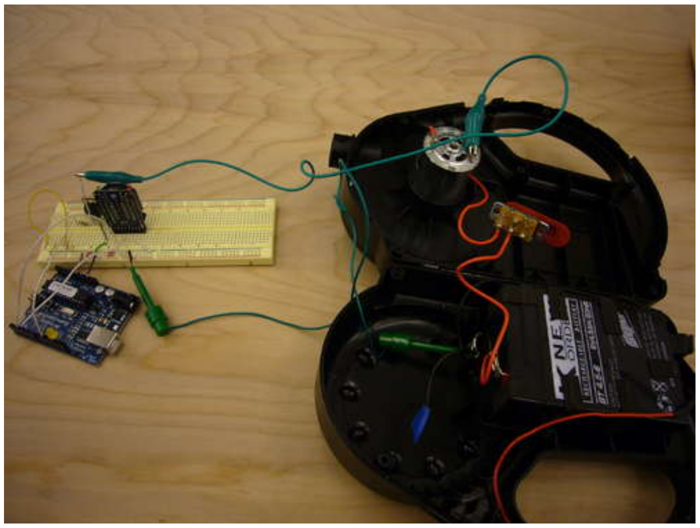


Image Notes
1. One end goes to the motor, from the TIP120's collector pin.
2. common ground with the arduino's power supply!

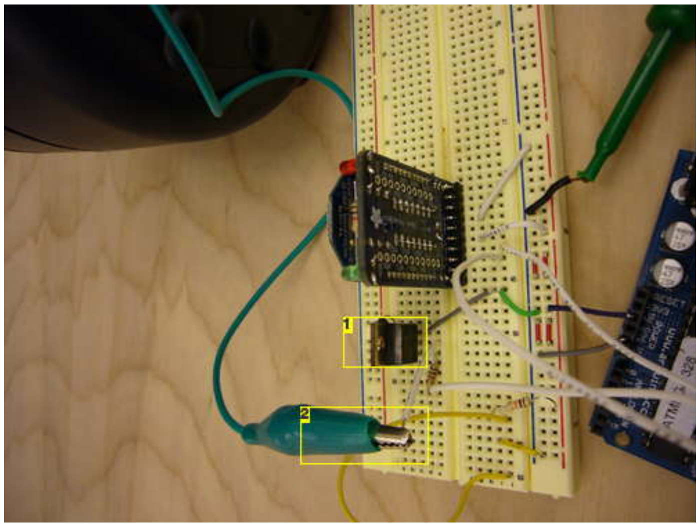


Image Notes
1. TIP120
2. To motor.

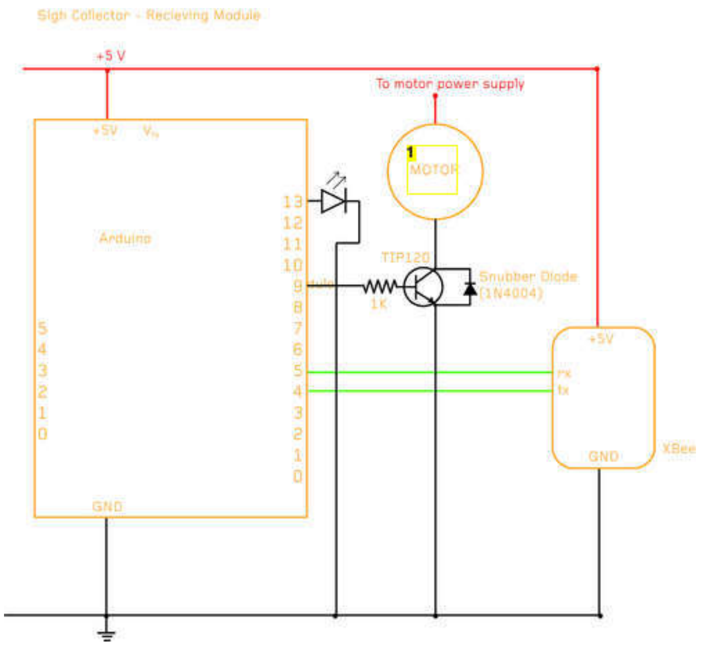


Image Notes
1. Pump.

File Downloads

circuit-sigh-recieve.pdf ((425x425) 241 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'circuit-sigh-recieve.pdf']

sc-reciever.txt (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'sc-reciever.txt']

Step 3: Build the Sigh Collector main unit

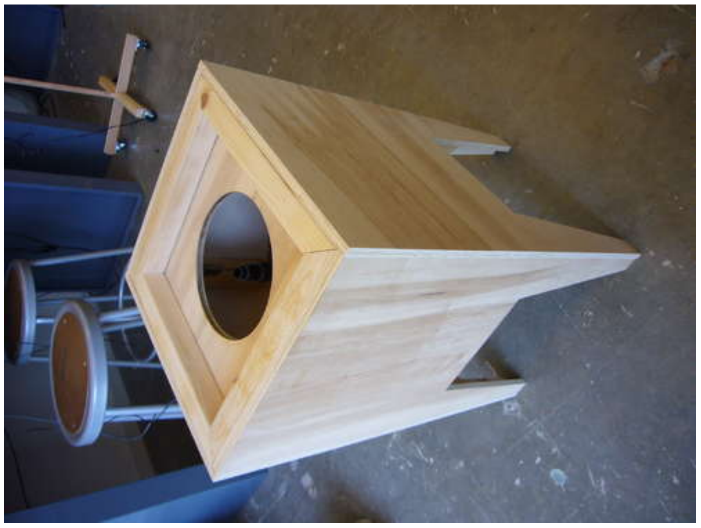
For the sake of brevity, I will not detail every step in the process of building the main unit. Suffice it to say that it can be as simple or complex as you wish; anything from cardboard and hot glue to custom fabricated or more advanced materials.

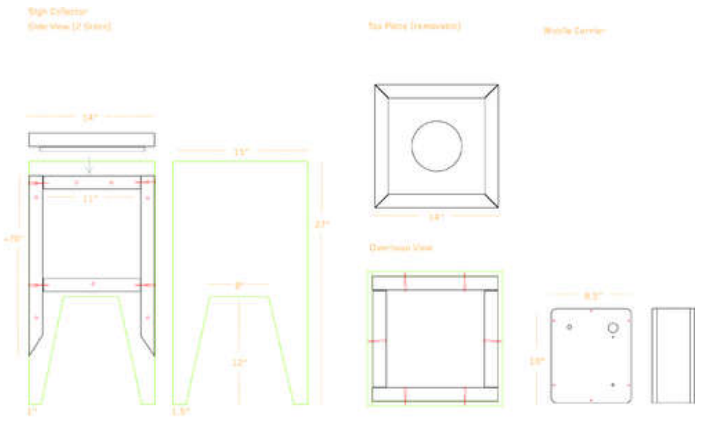
I have designed mine this way, which isn't to say it's the only way it could be done. If you care to follow or elaborate on my instructions, see the diagram below. Again, a higher resolution PDF is attached. On the diagram, you will find exact measurements and specifications on how to build the unit pictured below.

As stated in Step 2, I built mine out of shop-grade Maple plywood. It has a nice grain and cuts well. I left the surface raw.

A couple design notes:
I decided to drive all the screws in from the inside so that you wouldn't see them from the exterior. It can be tricky to sneak a drill inside of the unit, so I recommend building it in sections. I angled the bottom edges of the 2x2 frame, so that they would look a little sleeker when visible.
The top piece with the mitered corners and circular opening is removable, for easy repair of inside parts. The pump and electronics will sit inside the box, on a shelf that is held up by two of the 2x2's on the inner frame (see diagram).

The reason I built it on a frame is so that the corners would stay square. Otherwise, plywood can tend to warp. This way, also, everything can be held together by screws and therefore broken down easily into pieces.





File Downloads

sight-collector-diagram.pdf ((684x432) 231 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'sight-collector-diagram.pdf']

Step 4: Make the air bladder

I wanted a more organic, fleshy texture of my air bladder, so I cast it out of liquid latex. Liquid latex of many different sorts can be bought in a craft store, prop shop or easily on the internet. I mixed the latex with red pigment to color it, and painted it, in layers, onto the outside of a large balloon. The many layers built up to form a big, floppy fleshy balloon, with the texture I created with the brush.

A simple balloon, beach ball or even a garbage bag could replace. Check out this website for different types of large-sized balloons.

Step 5: Combine electronics with main unit. Install Check Valve and Pump

Place the air pump and circuit inside of the main unit, on the lower shelf. Now it's time to make a connection between the air pump, and the air bladder/balloon, which will sit on the surface.
We only want air to go one way, and not come out the other direction, so we use something called a "check valve" . The basic principle is that a hinged door, rubber diaphragm or ball is displace by air going one way, but then prevents the air from going back.

I bought my check valve on McMaster Carr's website; More specifically it's called a PVC Swing-check valve. I'm using the 1" diameter one. This one was attractive to me because of it's extremely low "cracking pressure", or the pressure needed to displace the barrier. < 0.1 psi !!

I used a simple garden hose to run from the pump, to the check valve, then from the other side of the valve into the balloon. The fittings are coupled and sized properly, and I used some glue to further secure them, and prevent any air leaks...

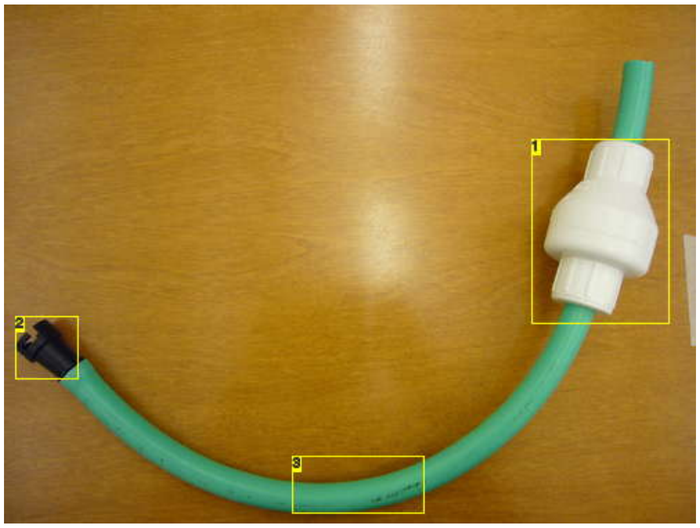


Image Notes
1. Low psi Check Valve, purchased from McMaster Carr.
2. Pump Attachment.
3. Typical garden hose from a hardware store.

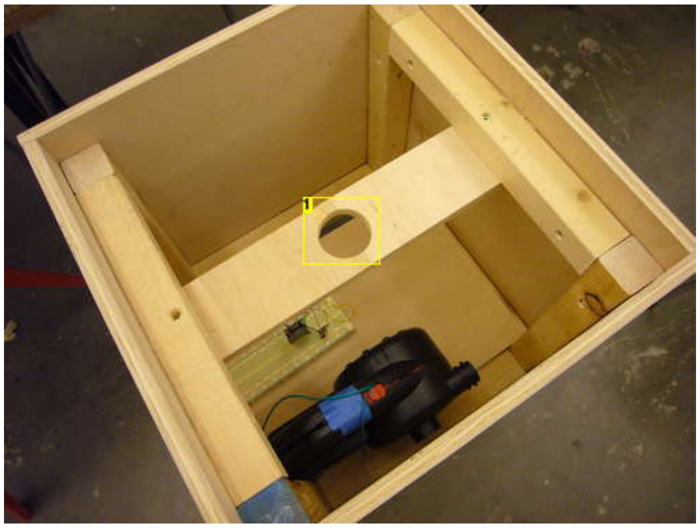


Image Notes
1. Added this piece at the end, to secure the check valve in place.

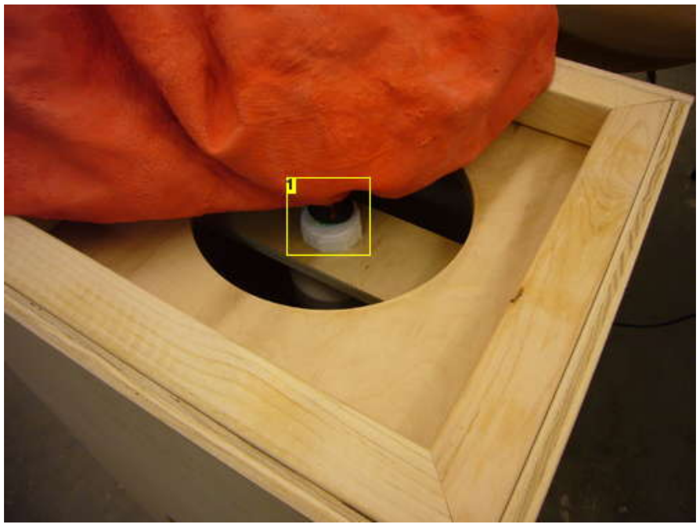


Image Notes
1. Secured the air bladder to the hose with a couple of zip ties..

Step 6: Build carrying case, Sew handle.

Sighing is monitored by a chest strap that you will wear. To hold the electronics and power supply, you must build a "carrying case". This will be mobile and will attach to the chest strap. You will carry this around with you while you perform your daily tasks and it will monitor your sighing activity. When a sigh is detected, the mobile unit will send a wireless signal to the main unit.

Again, you may follow the diagram I've provided and find measurements on how to build the carrying box. Or you may choose to make your own, unique version, or improve upon my own. I modeled mine after various kinds of medical, patient monitoring devices .

Notes:
I spliced an RCA cable in between the circuit and the sensor/chest strap (Steps 7 & 8) so that it can easily plug in and out of the box. I chose RCA cable because it's a simple way to have two stranded wires, packaged nicely with an easy to plug/unplug header. I slipped the RCA cable into a length of latex tubing, for aesthetic reasons.

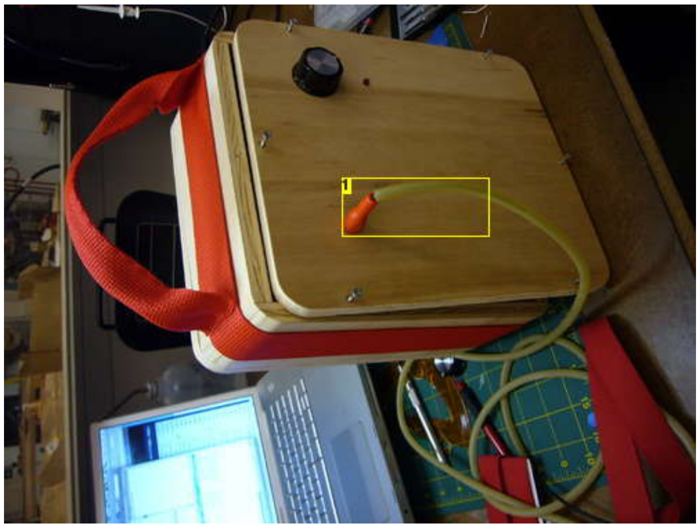


Image Notes
1. Sensor connects through an RCA cable. The cable is encased in latex tubing.

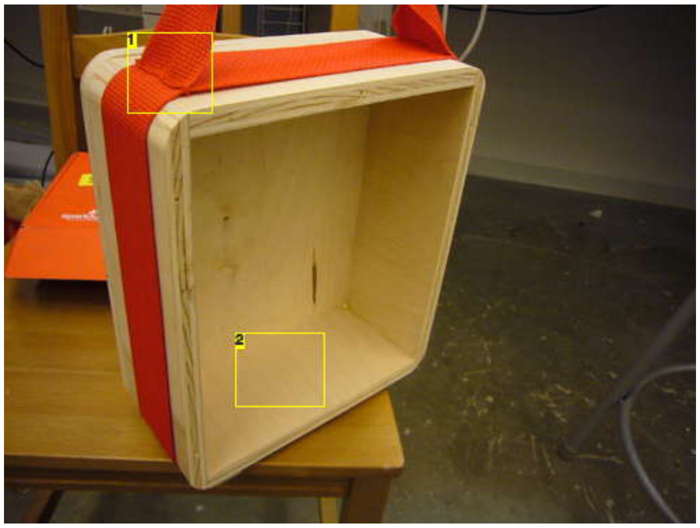


Image Notes
1. I sewed the nylon strap so it runs around the box and attaches to the handle.
2. The circuit will go in here.

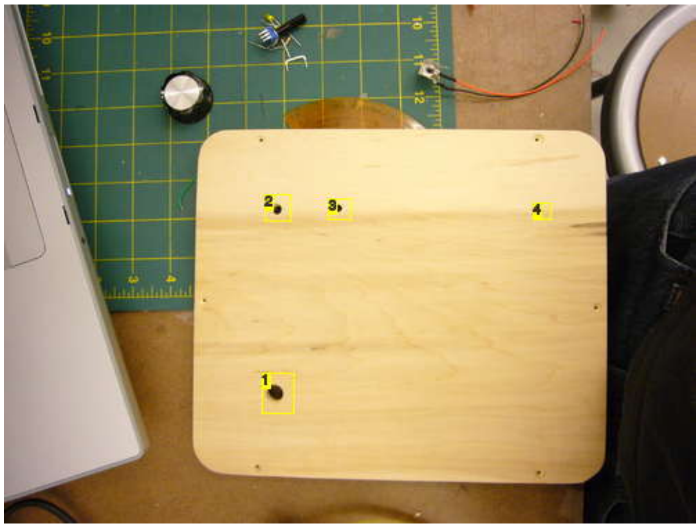
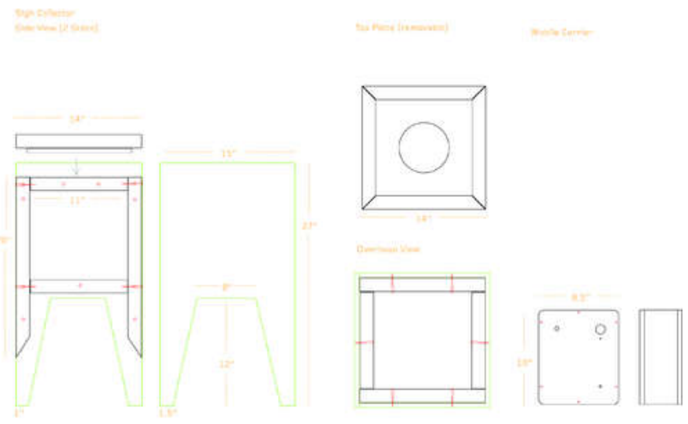


Image Notes
1. Sensor will attach here, via a hacked RCA cable.
2. A hole for the potentiometer to poke through. We'll attach the knob from the front.
3. Hole for the LED indicator light. This LED will illuminate when a sigh is has been detected.
4. A hole for the power switch will be here.



File Downloads

sight-collector-diagram.pdf ((684x432) 231 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'sight-collector-diagram.pdf']

Step 7: Build and Program circuit for sigh detection. Assemble electronics into carrying case.

Follow the circuit diagram below. A higher resolution PDF is also attached.
Program the Arduino with the provided code.

To monitor breathing, we will be making a chest strap that is outfitted with a stretch sensor. The expansion and contraction of the chest will provide us with data that we can use, in code, to extrapolate what normal breathing is, and therefore determine with a larger than usual inhalation (followed by large exhalation) is. A 10 or 20K potentiometer will be used to dial in a threshold value, which will represent how large of an inhalation is associated with a sigh.

I purchased my stretch sensor from Merlin Robotics , a company in the UK. They stock a variety of sizes. I'm using the 20cm sensor.
In my circuit, i'm amplifying the signal from the sensor with a resistor bridge and an OpAmp chip (see diagram). This is the method suggested by the manufacturer. You can find the datasheet on the internet. Note: I imagine a similar idea could be done with pressure sensor instead of a stretch sensor. You'd could attach the pressure point on the sensor to some kind of tubing and wrap that tubing around the chest.

Drill holes in the front face of the carrying case and attach the potentiometer, indicator LED, power switch and stretch sensor attachment (RCA, female) to it from the back before screwing the box back together.

I'm powering the Arduino with a 9V battery. I've got 2 of them wired in parallel so i'll get the same voltage, but double the amperage (it'll last longer).

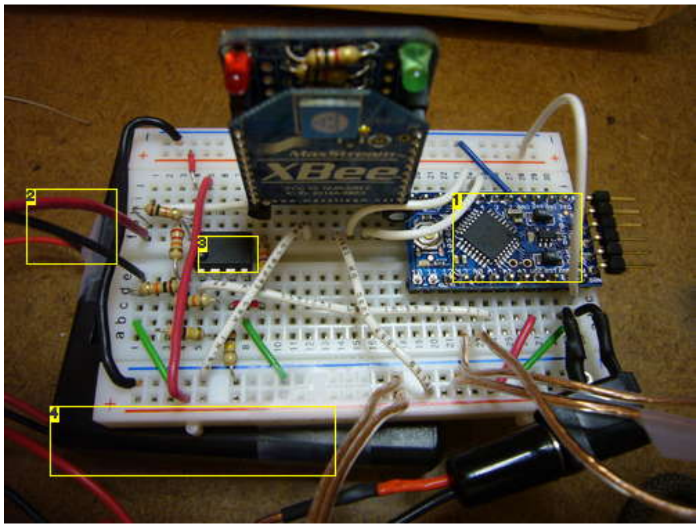
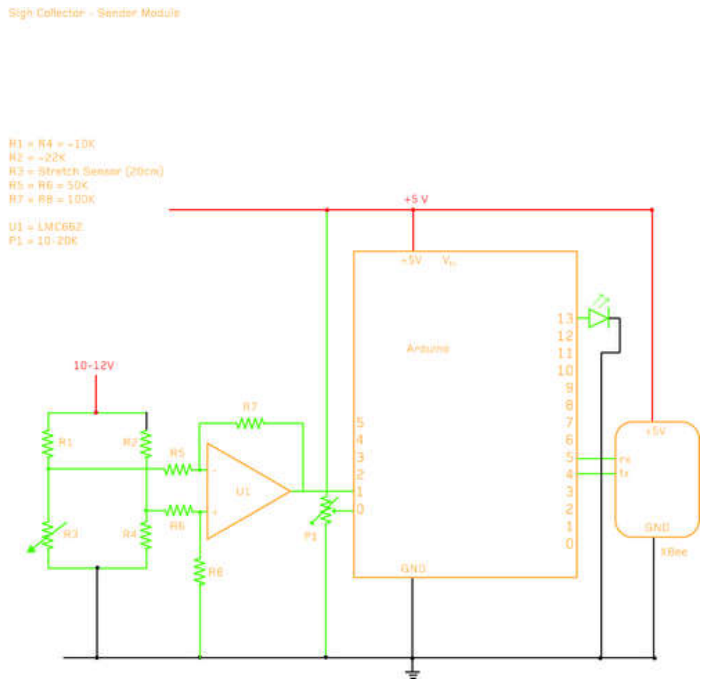


Image Notes
1. Arduino Pro Mini (just a smaller Diecimille)
2. Stretch sensor
3. Op-Amp
4. Powered by 2 9V batteries, wired in parallel for extra amperage (but same current).



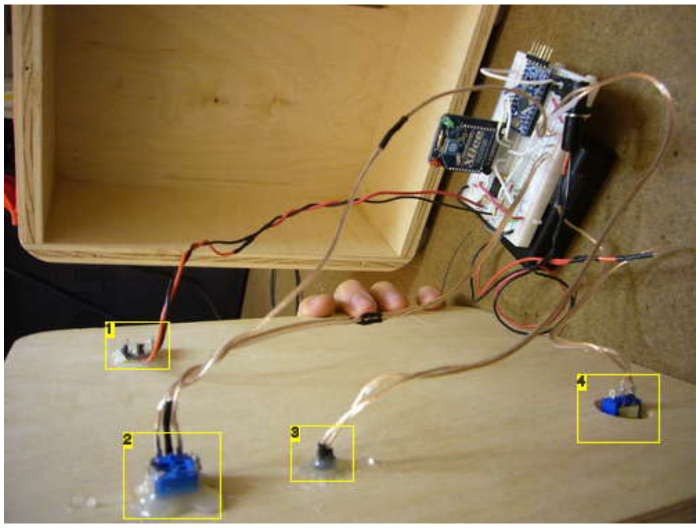


Image Notes
1. Stretch sensor connect
2. Potentiometer.
3. LED
4. Power switch.

File Downloads

circuit-sigh-send.pdf ((500x500) 247 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'circuit-sigh-send.pdf']

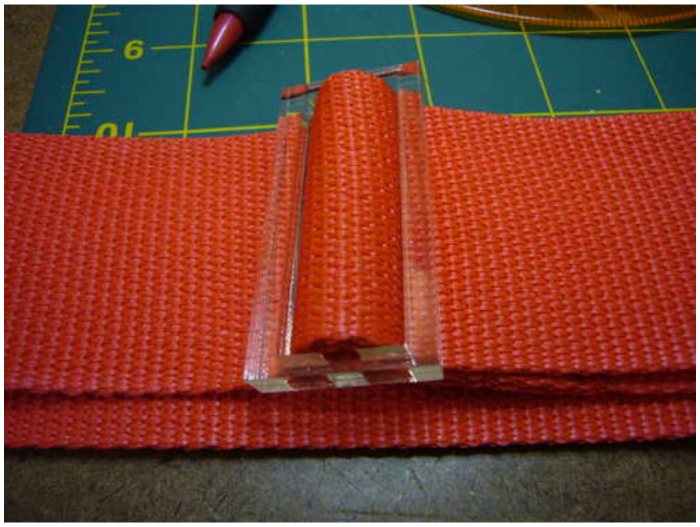
sc-sender.txt (2 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'sc-sender.txt']

Step 8: Cut and Sew chest strap and attach the stretch sensor.

The basic idea here, is that a fabric strap is wrapped around the chest by the lower ribs (where the most motion occurs). The stretch sensor bridges a small gap in the chest strap, the rest of which is not stretchy, so breathing, subsequently deforms the sensor as needed.

You'll have to measure the length of strap to your individual body type. I sewed an extra strip of fabric around the strap, so the wires can safely sit inside. In the front, where the stretch sensor connection is, I sewed a 'sleeve' of fabric that would loosely cover the sensor so it wouldn't get rubbed or damaged.

In the back of the chest strap, I made a simple shape (like on a backpack) for tightening and loosening the strap. I had the shape laser-cut out of clear acrylic (see image), but you can make it any way you are able to.



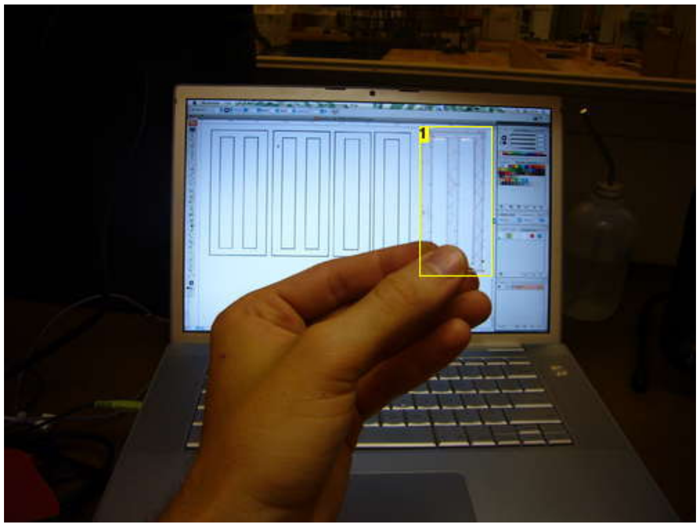
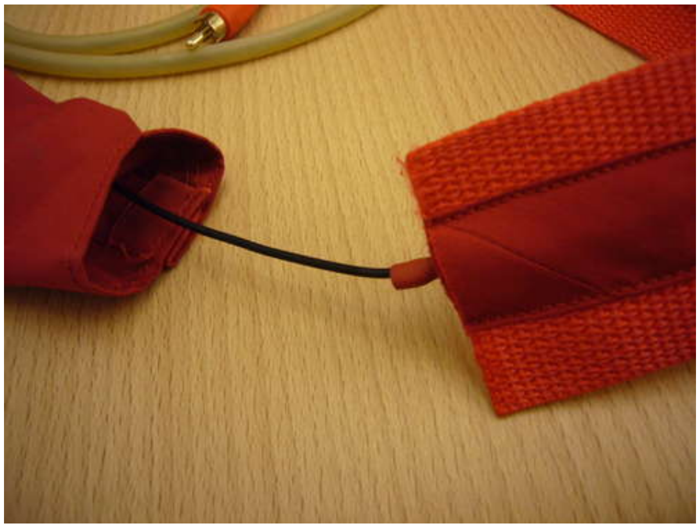


Image Notes
1. Final laser-cut piece, made from the background Illustrator diagram. This will be the strap tightener for the chest-strap.

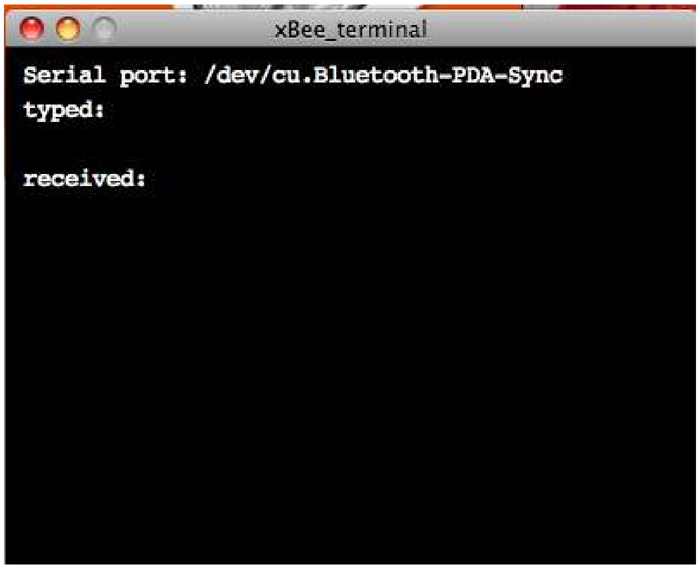


Step 9: A word on Wireless

One thing I haven't talked about yet, is how the wireless communication is being achieved. I am using xBee wireless modems. xBee's are an easy way to make a wireless point-to-point connection, or create a mesh network. To interface with my Arduino board, I used LadyAda's xBee adapter. It's inexpensive, easy to put together and there is a detailed instructional website explaining how to configure it.
Through a combination of this website, and a chapter on xBee radio's in the book "Making Things Talk " (Tom Igoe), I implemented, possibly what is the simplest use of these radios, which are actually quite powerful.

I got my adapters and xBees (+ the appropriate cable) from here .
Instructions on configuring the xBees are here .

The only thing i'm not going into is how to configure the xBees. I did it very easily (on a mac) by transcribing some code from Igoe's book that uses Processing to create a simple terminal for programming the xBee. That code is on page 198.

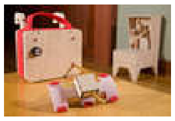


```
Serial port: /dev/cu.Bluetooth-PDA-Sync
typed:

received:
```

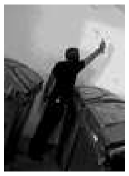
Step 10: Finished

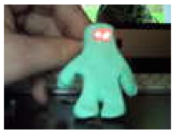
Congrats! You're finished. You can now use your Sigh Collector to monitor your emotional health.

Related Instructables

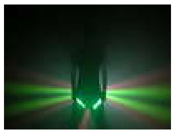

Foot Tap Amplifier by mkontopo

Pacing Track by mkontopo

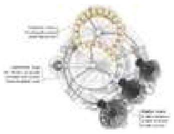

Smell Graffiti by numberandom


claymation improvement by shiboohi

Emotidora: Hats with Emotions by aiswaryak


Create A Laser Projector Show Without A Laser by Jixz

Cheapest storage box (Photos) by RebesaurioRex

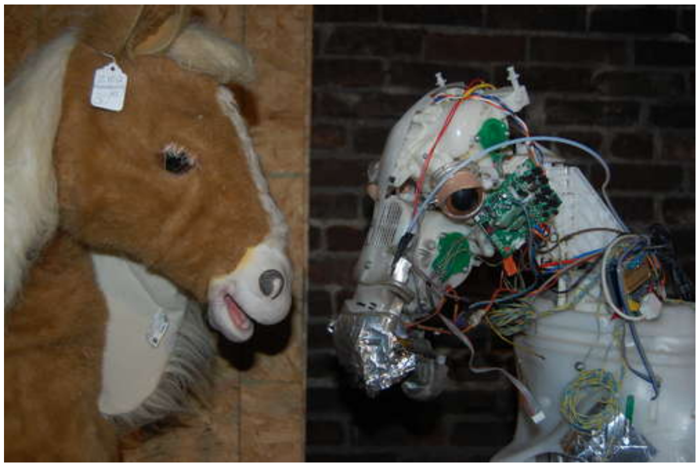

Custom Mechanical Biorhythm Computer, 3D printed by LabRat

Make a Fire Breathing Animetronic Pony from FurReal Butterscotch or S'Mores

by **lvl_joe** on November 12, 2011

Intro: Make a Fire Breathing Animetronic Pony from FurReal Butterscotch or S'Mores

For Maker Faire Detroit 2011, I displayed a hack I made to a FurReal Friends Butterscotch Pony. My fellow LVL1 Hackers and I had taken control of the motor control system of the toy and added a flame thrower to it. It seemed to go over really well with the crowd, so I am putting up the information for anyone to make there own. It was a blast to make and I hope everyone has as much fun remaking it. Just remember that this project uses Fire and should only be built and operated by no less then 2 adults with appropriate experience in fire safety and proper fire safety equipment on hand.



Step 1: Get it before you hack it

At one time, Butterscotch and S'more ponies both sold for around $300, but they seem to be discontinued. I would never suggest someone pay this much for something new, just to make it better. Thankfully, there is a fairly steady stream of them showing up on Craigslist and second hand stores.

I purchased my first Butterscotch off of Craigslist for $20. I have since picked up a second one for $25 from a peddlers mall. I commonly see them listed for ~$100, but with a little negotiation and/or patience you should be able to pick one up for dirt cheap.

raleigh craigslist > for sale / wanted > toys & games

Avoid scams and fraud by dealing locally! Beware any deal involving Western Union, Moneygram, wire transfer, cashier check, money order, shipping, escrow, or any promise of transaction protecti

Butterscotch Pony - $50 (Holly Springs)

Date: 2011-12-01, 6:45PM EST
Reply to: [illegible]

Furr Real Friends Butterscotch Pony in good shape.

Location: Holly Springs
it's NOT ok to contact this poster with services or other commercial interests

PostingID: 2731113285

Copyright © 2011 craigslist, inc. terms of use privacy policy feedback forum

Step 2: What you will need.

The tools you will need vary in this project. I will try to tier it based on what you want to accomplish with your FurReal pony.

Hardware you will need:

- FurReal Butterscotch or S'More Pony
- Arduino Mega
- Wire 18g
- Solder
- Electrical tape
- Wii nun-chuck
- Wii Nunchuck breakout adapter
- 0.1" 16-pin strip male header
- 1/8th OD ptfe tube (trade name Teflon)
- Bowden cable (brake cable for the back wheel of a bike)
- Scrap PVC tube around 3" at about 1' long
- Scrap plexi glass

Tools you need

- Wire Strippers
- Razor blade
- Phillips head screwdriver
- Flat head screwdriver
- Multimeter
- Soldering Iron
- Computer to program the Micro Processor (Any OS)

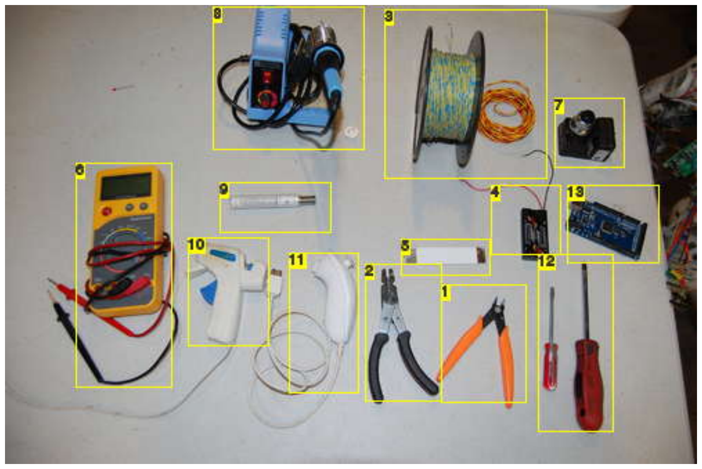
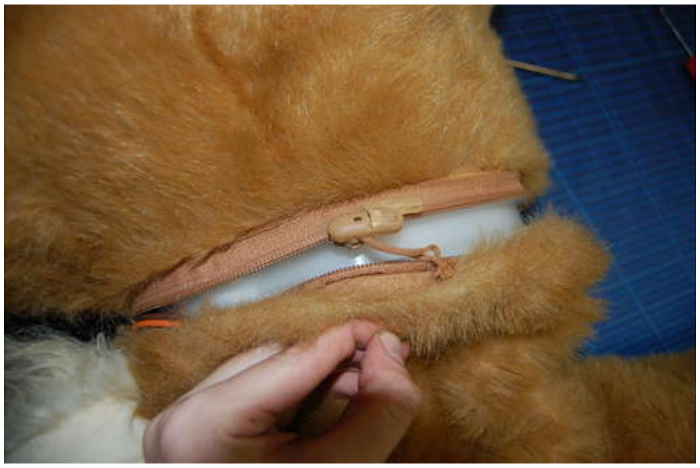


Image Notes
1. Wire cutters
2. Wire Stripers
3. Wire
4. 3v (2 AA battery pack)
5. razor
6. Multimeter
7. Grill igniter
8. Soldering Iron
9. Solder
10. Hotglue Gun
11. Wii Nunchuck
12. Screwdrivers

Step 3: Removing the skin: Head first

Before you get into the really fun parts, you will need to skin your pony. I started at the head as it already has a zipper. Move the mane out of the way, and locate the zipper at the base of the neck. You will find that the zipper pull has a cap over it to prevent it from unzipping. Simply break off this cap and unzip the skin from around the head.



Step 4: Removing Skin: ENT

Now we are going to remove the skin from around the face. The face is attached in 4 areas: the ears, eyes, nose, and mouth. Pull the skin up the back of the neck so you can get inside the back of the ponies head.

The ears break off easy (one was already missing when I got to this point). The other broke off when I was trying to get the skin off. This is not a problem as you can glue them back on easily. If you wish to keep the ears attached, you can cut the cloth around the ear holes with a razor. If you are OK with taking them off, a hard tug should pop them right off. Once the ears are removed, pull on the fabric where it tucks into the head. This will rip the seams, freeing it up.

Roll the skin further down, and you will get to the eyes. The fabric is sewn into the top and bottom of the eyes. Simply cut the stitches here to detach the fabric. Try to not look your pony in the eyes when you do this, as you may start feeling bad about what you are doing.

Moving down to the ponies nose, there are 2 pegs holding the rubbery snout on. One in each nostril. These slide out without much trouble, away from the body in a parallel fashion. The rubber on the snout is thin, so try to get a grip on the hard plastic with your tool as not to tear it.

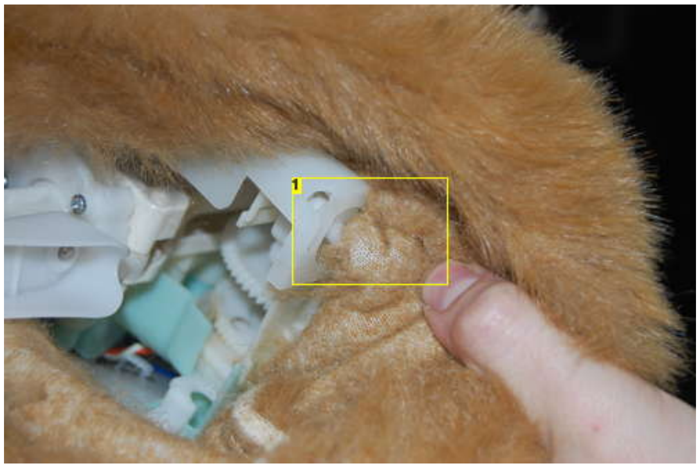


Image Notes
1. Fabric tucked in around the ears. I already removed the years at this point.

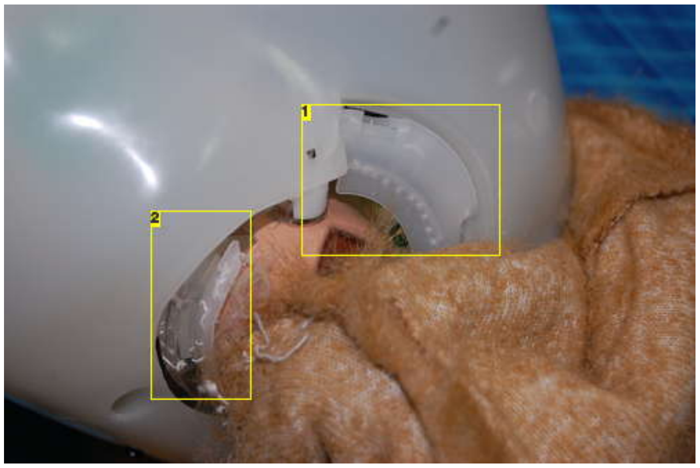


Image Notes
1. Fabric sewn into the plastick under the eye.
2. Fabric sewn into the top of the eye.

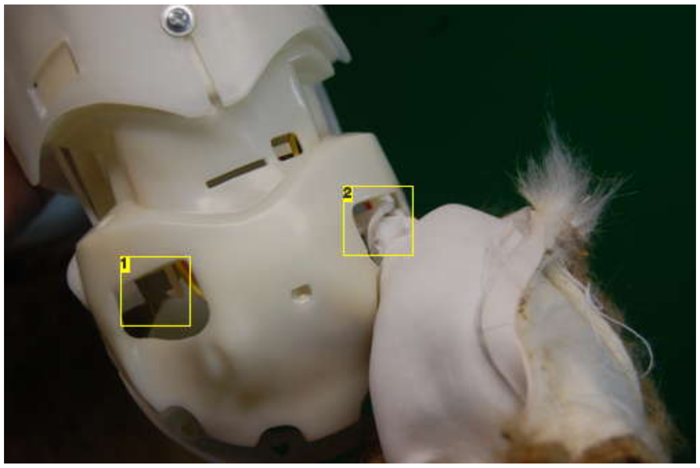


Image Notes
1. already removed
2. Slide out away from the body and it should pop right out. be careful not to beak the rubber on the skin.

Step 5: Remove Skin: Straight from the horses mouth

The last step of peeling your ponies face off is to removed the skin from the mouth. The the skin on the upper and lower jaw are both connected in their own way.

To removed skin from the upper jaw you just need to fold the face down until you see a horse shoe shaped piece of plastic around the mouth. It will have 4 pegs pushed up into holes with 3 legs that close around them. You will just need to bend 2 of the legs on each peg back and they will pop right out.

The bottom jaw will most likely have popped off by this point. We need to remove the jaw plastic from the rubber. It for the most part will pill off but it will take some time. I also had to cut some spots with a razor that where to fused.

After you get the lower jaw removed slide it back into the slot under the chin and glue it into place.

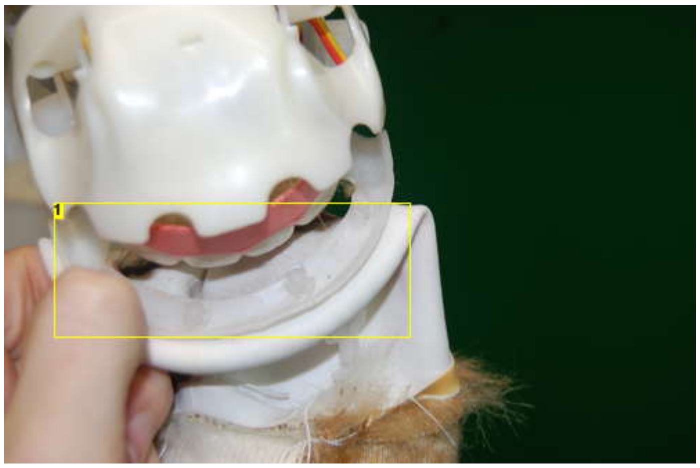


Image Notes
1. The 4 pegs holding the upper lip on.

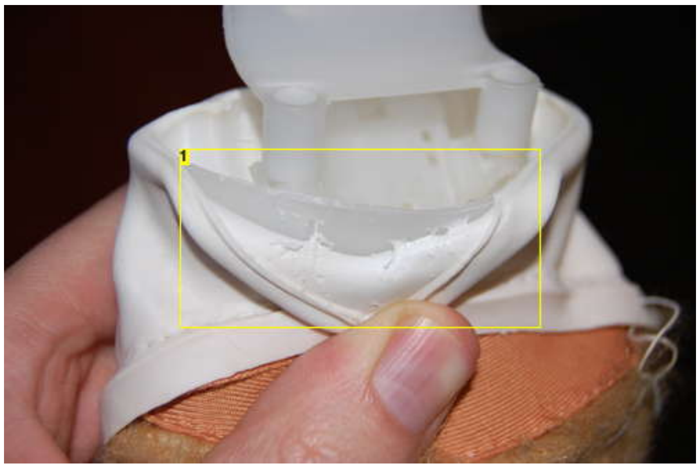
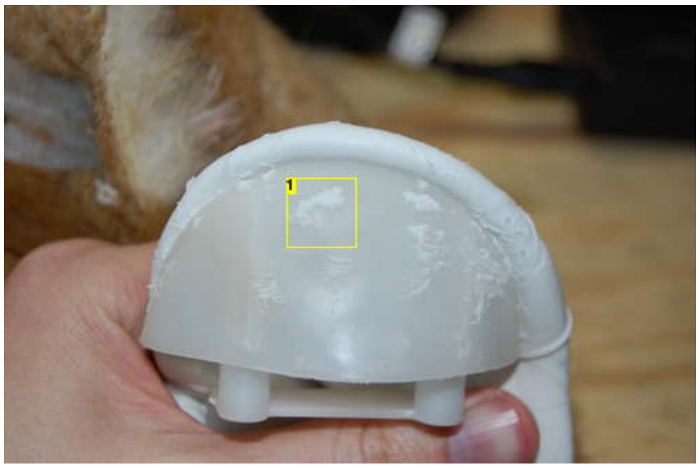


Image Notes
1. Removing the rubber from the plastic. pulling will do it for most parts but a few will have to be cut.



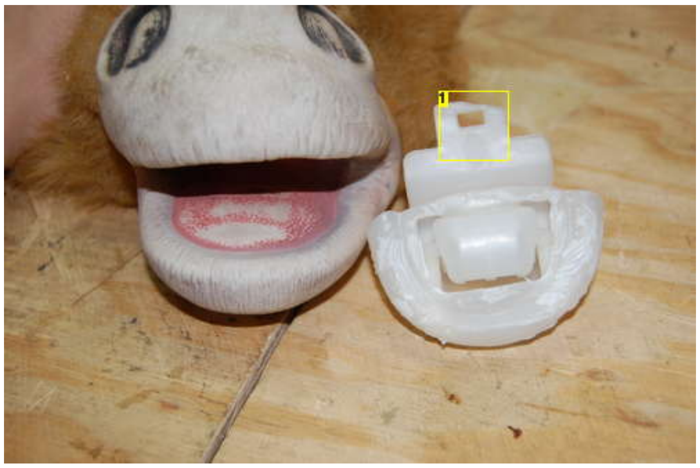


Image Notes
1. Part like have a little rubber on then is from where I had to cut it with a razor.

Image Notes
1. This part slip into a slot under the chin. It will not stay so you will have to hot glue it.

Step 6: Remove Skin: The body

Removing the skin from the body is a lot of the same. there is a few more places that they sew the fabric to the plastic. I followed the seem in most parts. just cutting a little bit of the thread then pulling it apart.

On the underside pull the velcro open. In the back side where the Velcro ends you will find a zipper leading up to the tail. There will be stitches at each end holding it together but no slider. cut the stitches and it will come unzipped. There will only be a small area holding the zipper area to the Velcro area. Cut this small bit of fabric and we will move to the front.

The frond end of the Velcro has a small stitch leading to a T intersection. Unstitch this area then unstitch along both sides of the T till you get to the legs.

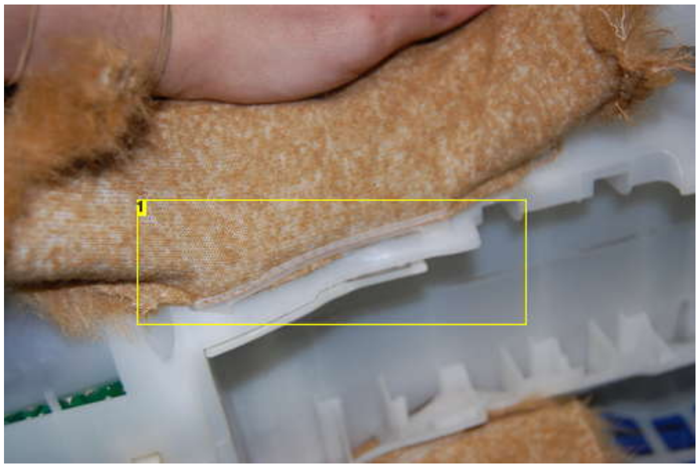


Image Notes
1. Fabric sewn to the underside of the pony.

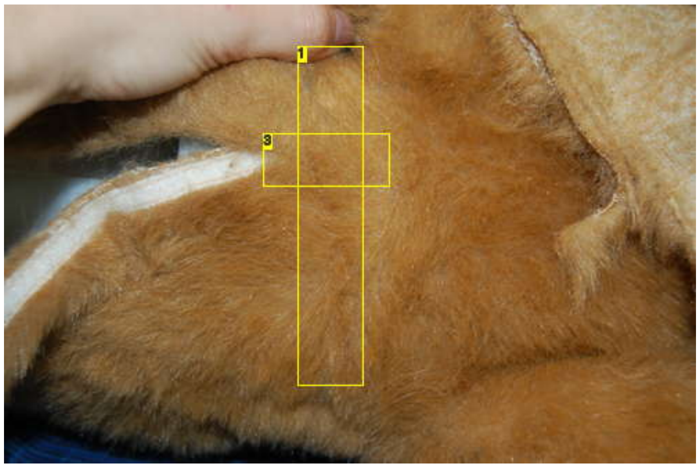


Image Notes
1. Seem that crosses from one leg to the other in the lower front of the pony.
2. Seem continues from the Velcro to cross seem.
3. Seem continues from the Velcro to cross seem.

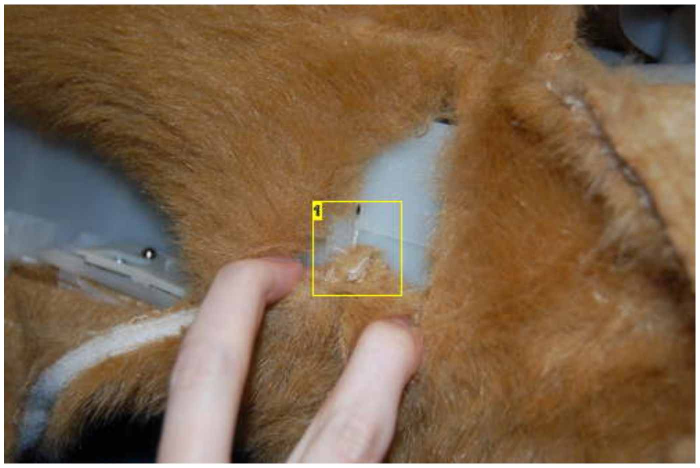


Image Notes
1. Another place it is sewn to the plastic.

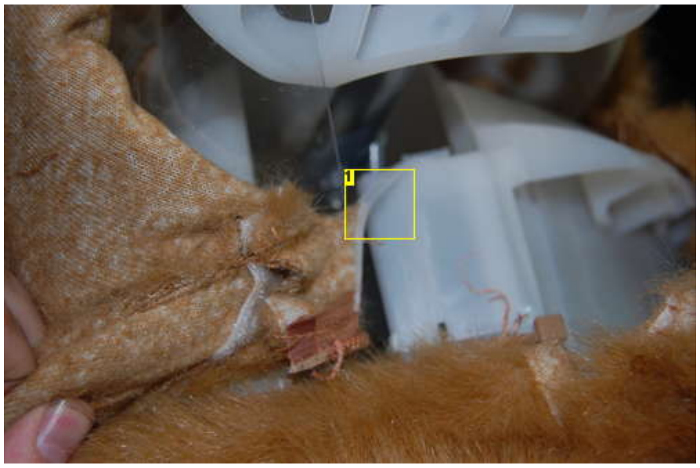


Image Notes
1. and yet again.

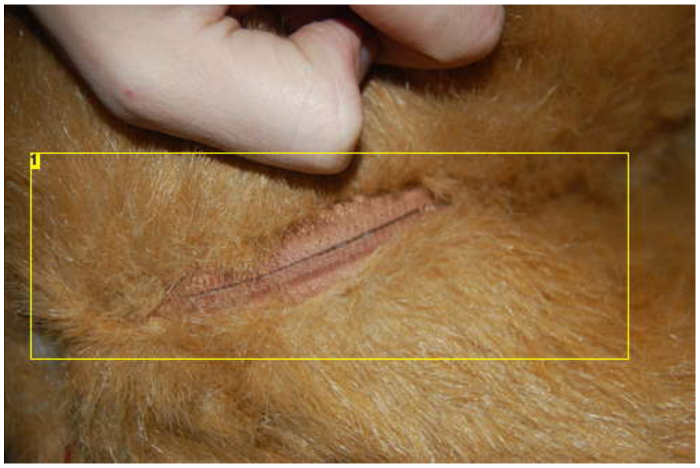


Image Notes
1. Zipper running from the tail to the Velcro under the pony. It is sewn over at each in but after you cut the threads it just unzip without a problem.

Step 7: Removing the skin: The legs

There is an inner seem along each leg. Snip the stitch and tear down each of them till you get to the feet. The feet are held on with a different fabric that is then looped around a rope inside of the hooves. Just cut around the bottom of the leg removing this other fabric and freeing the rest of the leg.

At the top of each leg you will need to remove some stitching leading around the intersection of the inner leg and the lower side of the body. When you get close to the stitches running along the bottom of the pony that you have already cut, snip away the fabric in between. You should not be able to lift the skin up all the way around the head.

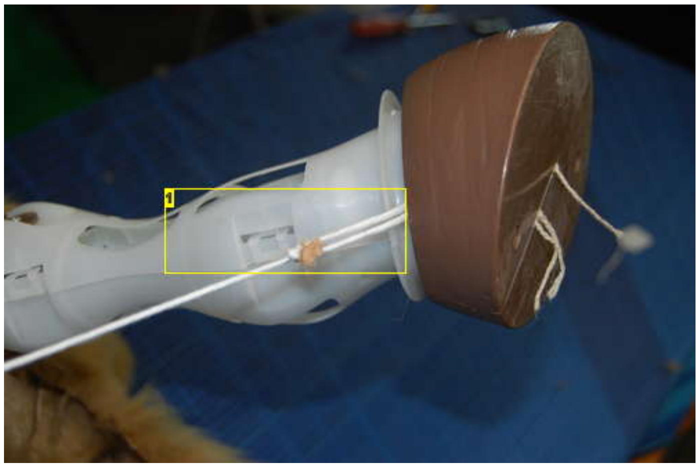


Image Notes
1. The string that was inside of the fabric loop

Image Notes
1. What you should have at the end of this step

Step 8: Removing the skin: the Neck

The very last step is to remove the part that is holding the fabric to the neck. I did this by cutting through the lighter, fur-less fabric at the neck. After I removed it, I found that it was held on by a zip tie inside of that loop of fabric. You can do it my way, or you can insert the wire cutters into the fabric at the nap of the neck and simply cut the large zip tie. This should allow you to remove the skin completely.

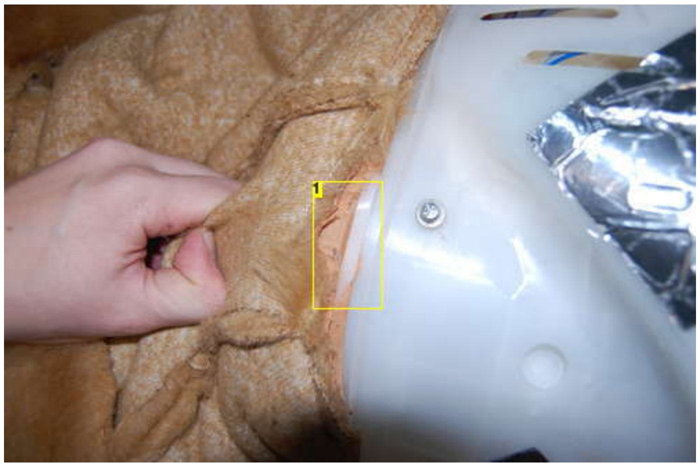


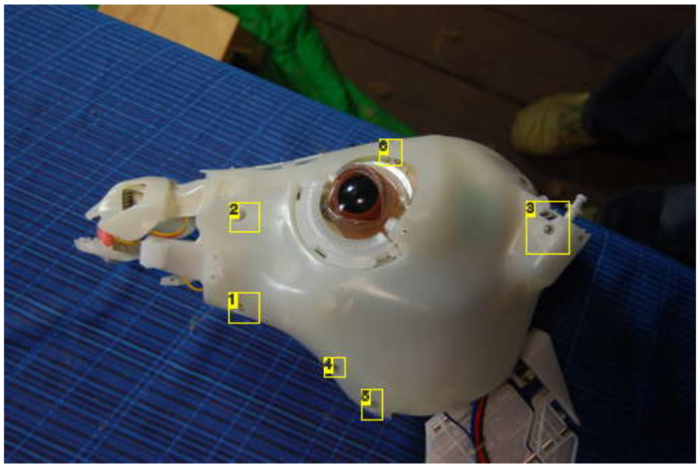
Image Notes
1. the non fur fabric looped around a zip tie.

Image Notes
1. What you should have at the end of this step

Step 9: Removing the face

To get to the main circuit board in the head, you need to removed the left side of the face (the pony's left, not yours) There are quite a few screws that you will need to remove to do this. I tried to mark all of the screws in the images below, but if it will not come loose after removing those, just look around for extras I may have missed. There is also a clip that holds the snout onto the rest of the face. This clip was difficult to remove and I ended up marring the face a bit with a screwdriver and wire snips.

Under the face you will find that the circuit board is held in place by 4 screws. Remove these screws, as we will do most of our work from the lower side.



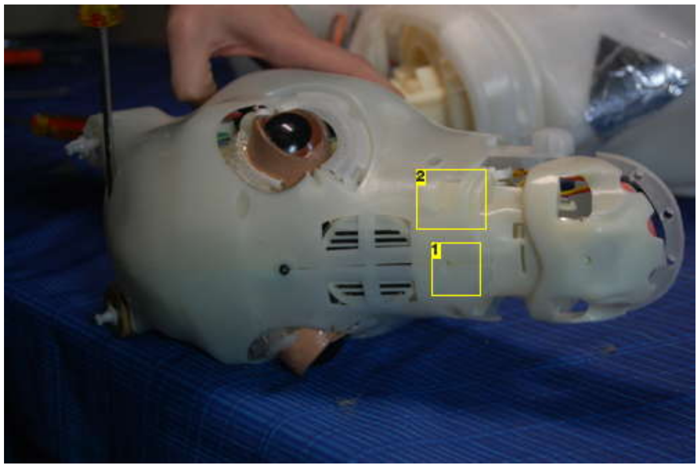
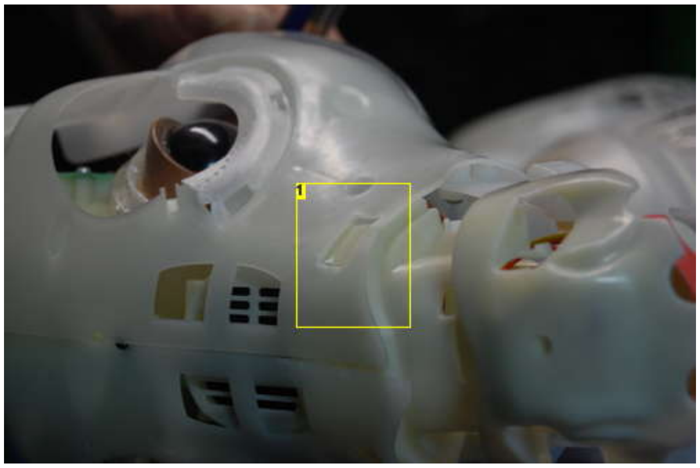


Image Notes
1. Screw
2. Screw
3. Screw on the outside and inside of the ear mount
4. Screw
5. Screw
6. Screw

Image Notes
1. Screw
2. Clip



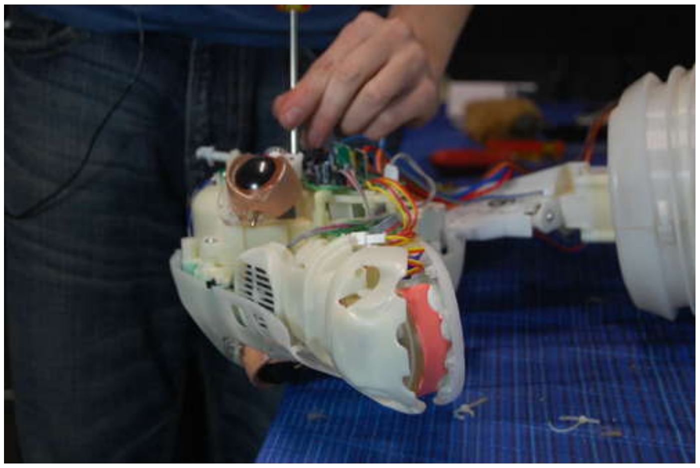
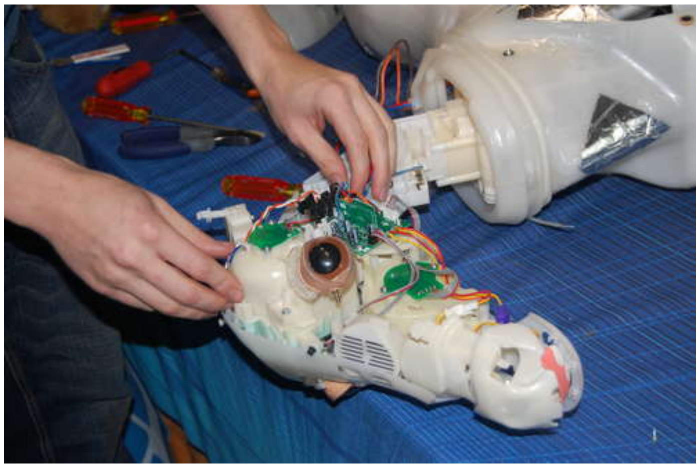
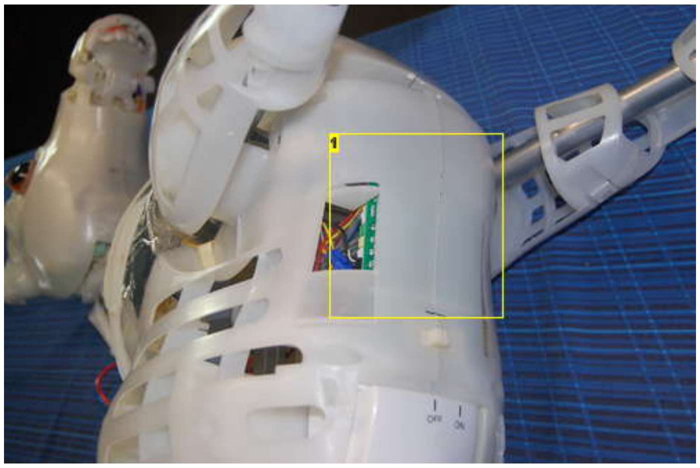


Image Notes
1. Clip



Step 10: Getting access to the Circuit board in the lower body.

I removed every screw I could find from the pony and still could not get the body open. I could not see any clips I could open, or anything else I could remove to release it, so I did the next best thing and dremeled a hole in the stomach. This ended up working out in the end as it provided a good place to put the fuel for the flame thrower. You will want the hole large enough to allow your PVC tube to just slide in.



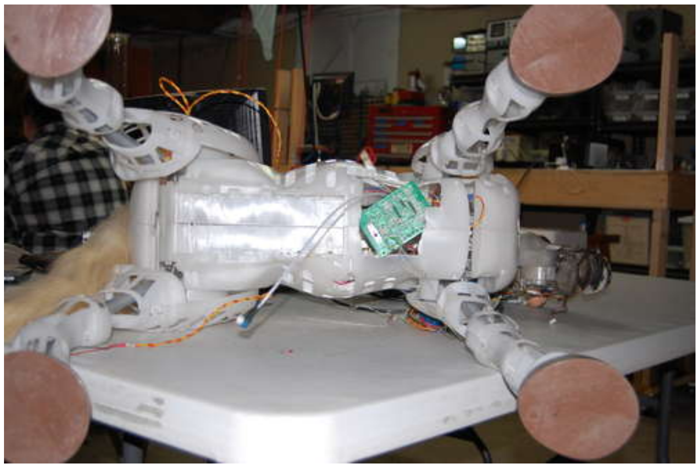


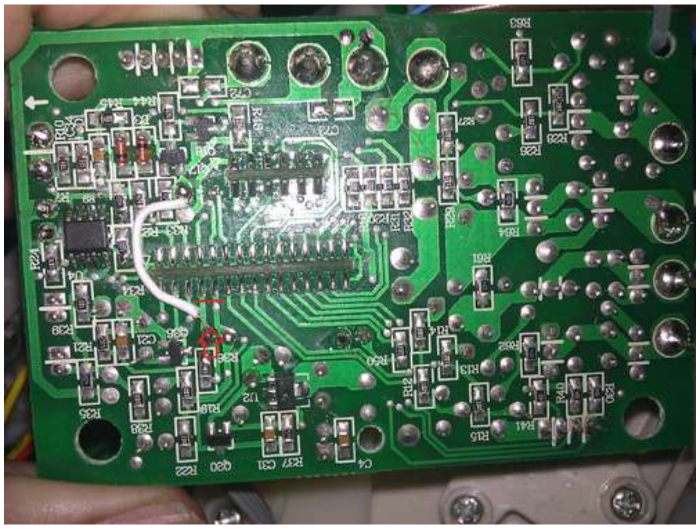
Image Notes

1. Circuit board for the body. location where we will load the fuel.

Step 11: Cutting the power to the Microcontroler

To take control of the pony we will cut the power and the ground to the Micro-controller that is currently controlling the pony. There are two controllers in the pony, one in the head, and one in the body. The one in the head sends commands to the one in the body so we will only be cutting power to the head, and this will take care of both.

To do this, cut the trace going to the 4th and 5th pin on the larger of the two boards sticking out at a right angle. The traces will be on the back side of the board. The 4th pin should have a white wire soldered to it. Using a razor you should be able to cut the trace without a problem.



Step 12: Tapping power for the Arduino

Now we need to power our Arduino, but there is no need to add another battery when we already have the 6 C cells powering the pony itself. Tapping into the power being pulled into the circuit in the pony's head will give us around 9v. I had a few 9v wall warts laying around so I cut the cable off of one with 5.5mm/2.1mm barrel jack on it. I tied this into the connectors going to the head. You can also purchase an adapter from adafruit ,as that will be much easier.

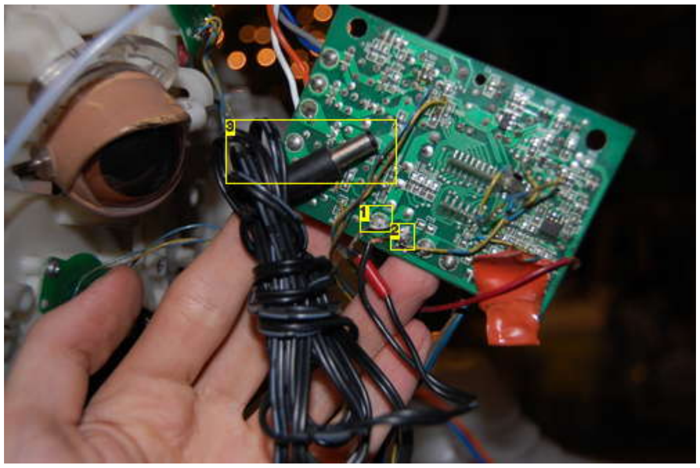


Image Notes
1. V
2. V-
3. 5.5mm/2.1mm barrel connector

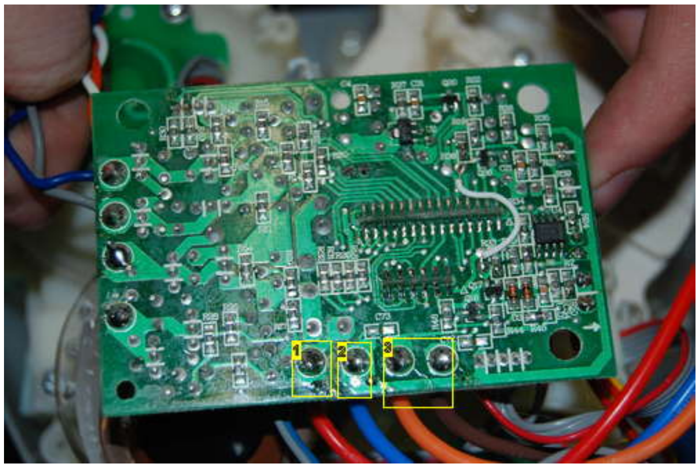


Image Notes
1. Power
2. Ground
3. Data lines

Step 13: Tapping the lines into the motor control circuit.

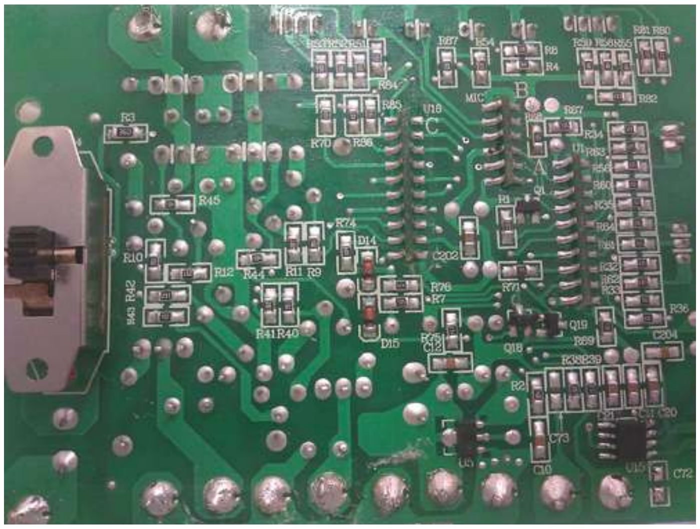
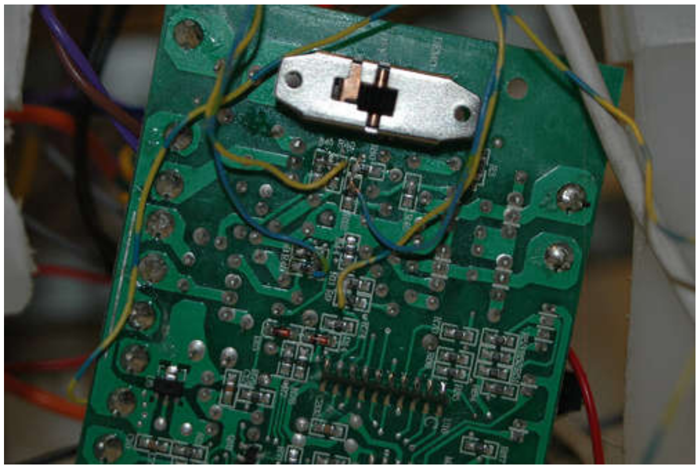
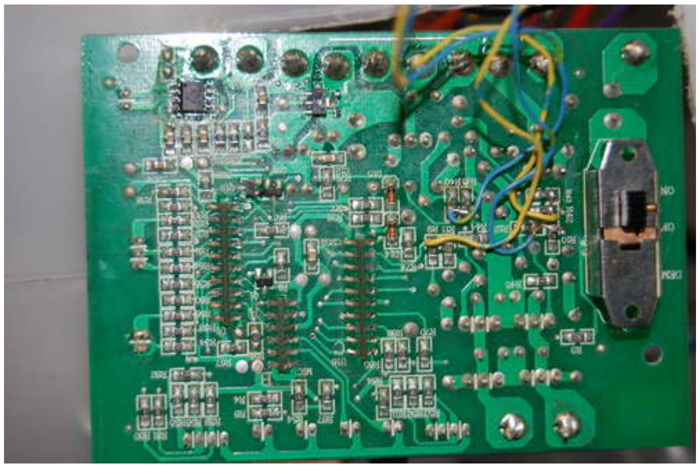
You will need to tap into the the lines coming out of the motor control circuit coming out of the micro-controllers. We will do this at 4 spots on each of the Circuit boards.

On the board in the pony's head you will need to solder your wires into R14 ,R15, R27, and R28

R14 and R15 move the head up and down plus open and close the mouth.
R27 and R28 move the head move left or right as well as move the eyes and ears.

On the board in the body, you will want to solder you wires onto R10,R42 and R11,R41

R10/42 move the head left and right
R11 Moves the tail (only one way)
R41 bobs the head up and down at the neck (moves one way around in a circle like the tail)

Step 14: Taping into the encoders.

There are 4 encoders that will tell you the position of the head. Two of them are located in the head and 2 of them are located in the body. The two in the head are easy since you can see them when you take the pony's face off.

Solder a ~2' long wire to each of these encoders. I used 18g wire.

For the encoders in the body, I was unable to find an easy way to get to them, so I cut the end off of the wire. We will solder the wires from this cable right into the bread board so strip them and you are finished with them for now. Try to leave these wires as long as you can.

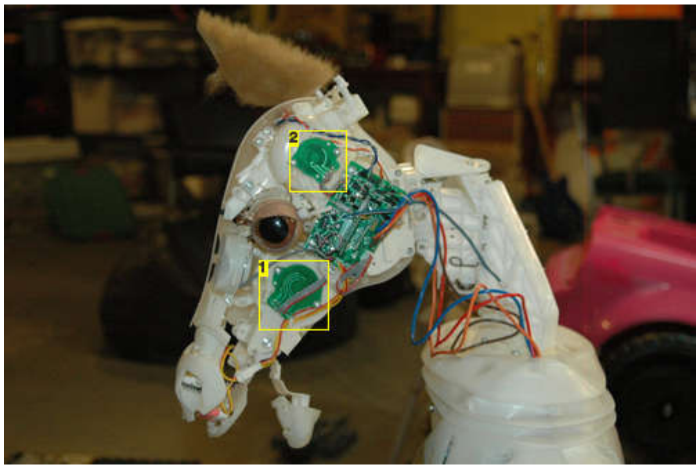


Image Notes
1. Reads the state of the head up and down as well as the mouth open and closed. Listed as Wipe1 on the board
2. Reads the state of the head tilt left and right, plus the movement of the eyes and ears. Listed as Wipe2 on the board.

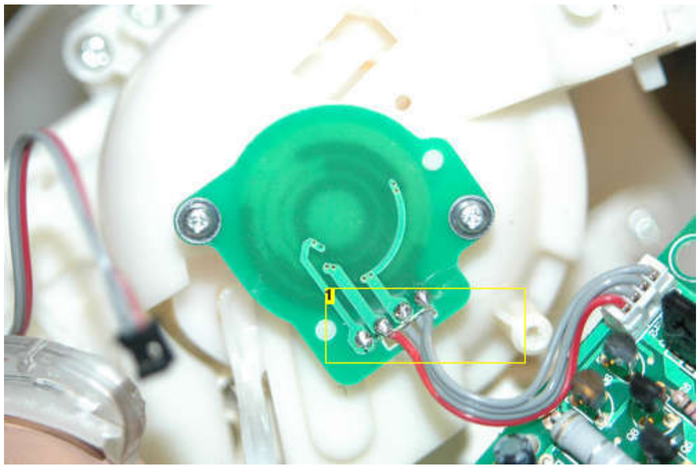
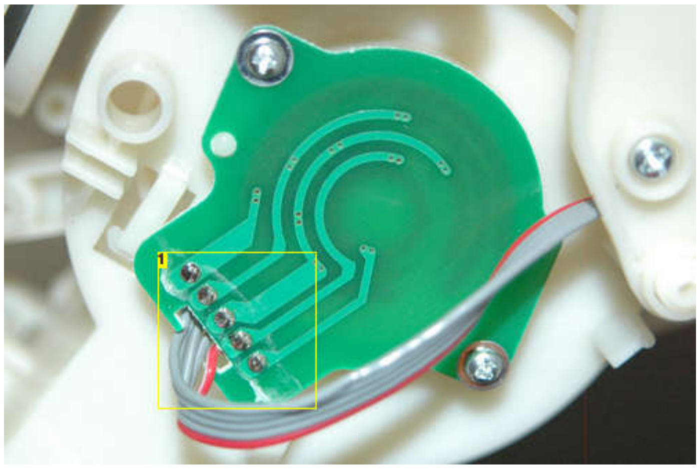


Image Notes
1. Solder onto the encoder rather then cutting the cable.



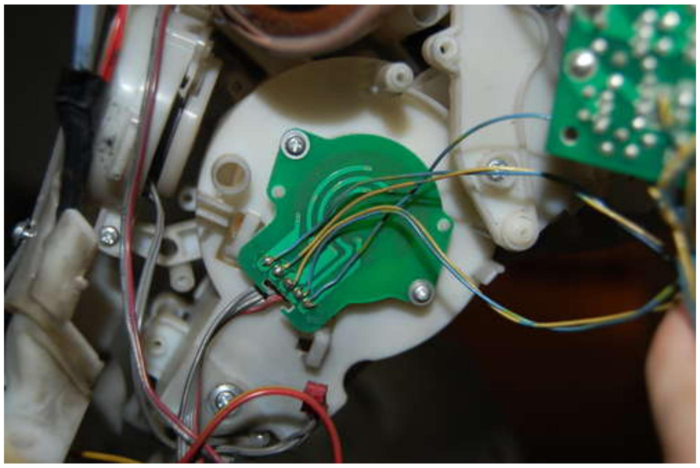
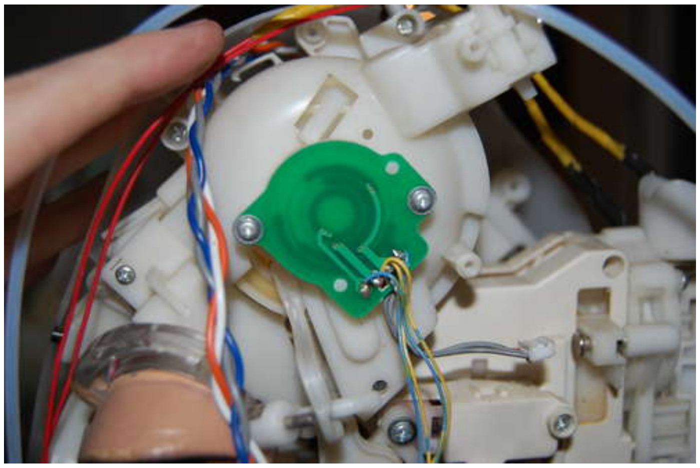


Image Notes
1. Solder to the pads rather then cutting the cable.



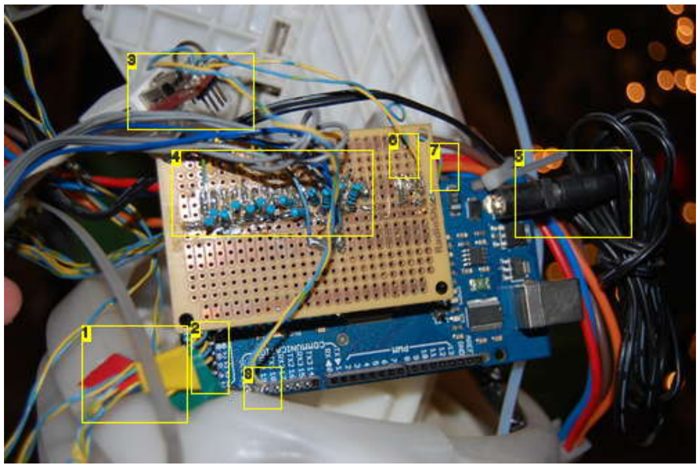
Step 15: Getting the morors and sensors connected to the arduino.

For the current code you will want to have the pins as such

Resistor label --- Pin on the Arduino

R14 Pin 23
R15 Pin 25
R27 Pin 27
R28 Pin 29
R10 Pin 37
R11 Pin 35
R41 Pin 31
R42 Pin 33

To get the pins connected I soldered them to the end of .1 male header.



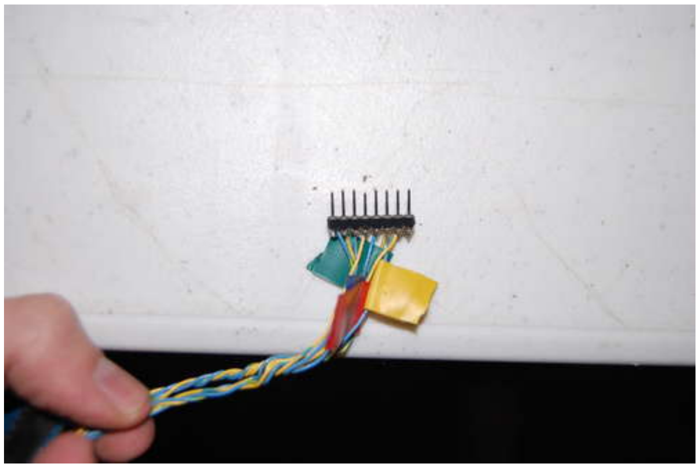
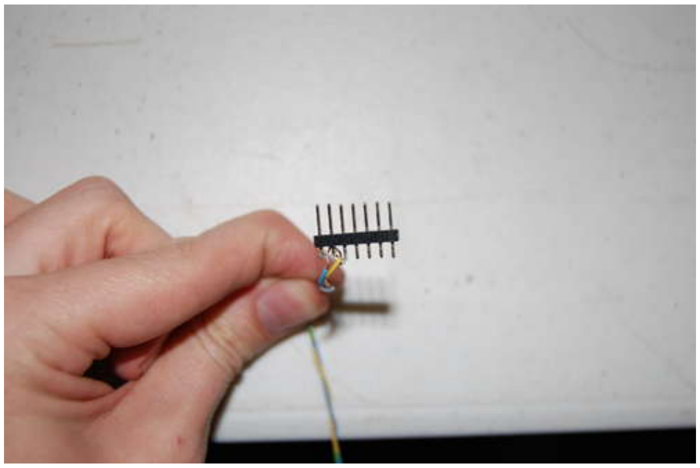


Image Notes
1. Electrical tape so that I know what pin is connected to what motor.
2. Going to the motor control
3. Nunchuck breakout board.
4. Pull downs for the encoder.
5. Plugged into the pony's power supply.
6. Ground pin for nunchuck
7. 3.3v going to nunchuck
8. I2C pins for the nunchick

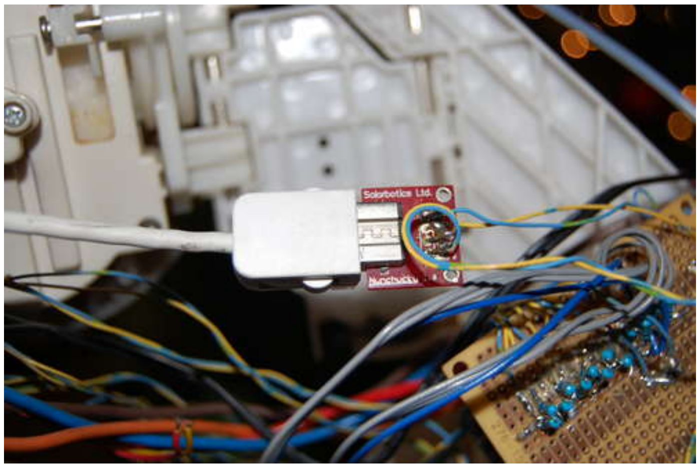


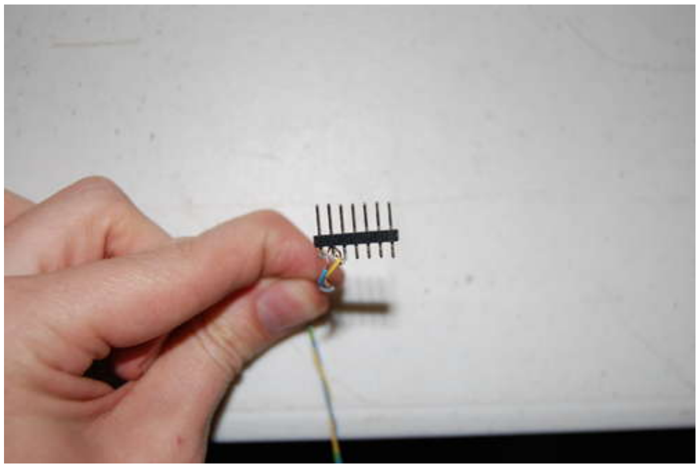
Step 16: Connecting a wii nunchuck into the system.

Now you will need to connect the Wii nunchuck breakout board from adafruit. If you want to run the wii nunchuck at 5v you can just use the .1 pitched pins that are on the breakout board, you can set the the input pin 19 as 5v output and pin 18 as input. I connected mine with wires and chose to play it safe by running it at 3.3v.

On your mega, connect it as such
Gnd: Ground
3.3v: 3.3v
Data: 20
Clk: 21

You will want to check the arduino wiring documentation if you are using something other then a mega





Step 17: The Arduino Code.

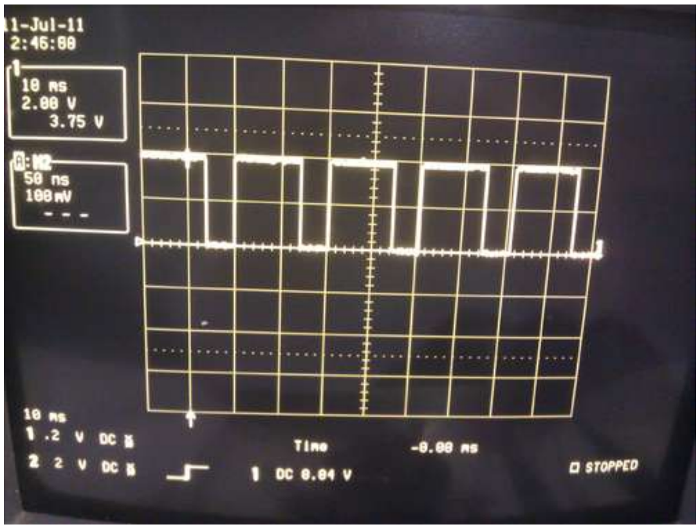
The code should be loaded on the arduino mega using the arduino IDE . Before doing so, you will need to put the modified wii nunchuck file into your arduino libraries folder. It should be in the root of your arduino IDE install. It should look something like "C:\User\joe\arduino-0022\libraries\WiiChuck \WiiChuck.h . Make sure that you put it inside of a folder named WiiChuck so that it can be found by the Arduino. After you load this on your Arduino, you should be ready to start moving the pony around.

The sketch has to bitbang the PWM sent to the motor controls as there are to many pins to do it on PWM pins. I think it runs too slow for the Arduino to do it with hardware anyways.

Current controls work like this:
Push the joystick one way and the head will start moving that way from a dead stop.
Move it the opposite direction from the way it is currently moving and it will stop moving.
Move it up or down and it will move that way from a dead stop.
Move the opposite direction then the head is moving and it will stop moving.

C moves the Tail
Z shakes the head

The mouth moves when the head is moving up and down.
The ears and eyes move when the head is shaking.



File Downloads

FirePony.pde (8 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'FirePony.pde']

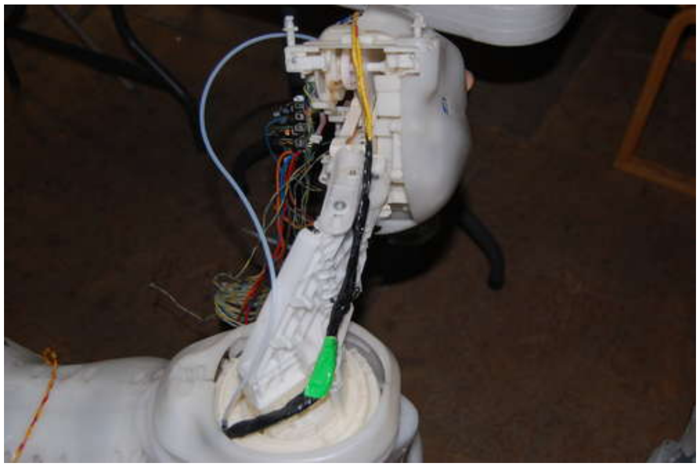
WiiChuck.h (6 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'WiiChuck.h']

Step 18: Getting the fuel to the head

The fuel we are using for the fire is automobile starter fuel. This can be purchased at most any store that sells automotive supplies.

To attach the fuel source, we will connect one spray caps from the starter fluid to a teflon tube that runs from the hole in the belly up along the neck and then out just over the nose where we will place our igniter. The tube will not be the right size to fit into the cap so use a drill bit that is the same diameter as the tube to bore it out until it fits.

Be prepared for some frustration as Teflon is very hard to push into things do to its extremely low Coefficient of friction. Once you get it in, put some hot glue around the joint to hold them together.



Step 19: Building an ignition system.

The ignition system is an electric grill ignitor that has had the leads on it extended. Just over the mouth (or wherever you want the fire to come out) we have attached a metal plate that is connected to one lead of the ignitor. The Teflon fuel tube is epoxied right over this lead. Finally, the other lead off of the ignitor is connected to a wire on the other side if the fuel tube, so that the aprk will arc across the tube.

The fuel has nothing to burn so if the ignitor is not running it will blow out in a small breeze. When modifying the ignitor you will want to connect one 6 foot wire to each lead. Ours had 4 leads, if you have the same type you can put electrical tape over the other two so that they do not spark inside of the body. You will also need to insulate the leads that you extended with tape, shrink tube, or hot glue to keep it from sparking at the ignitor rather than at your extension leads.

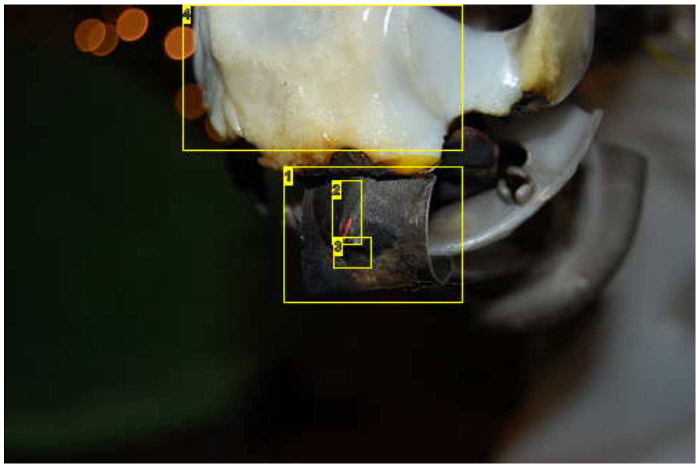
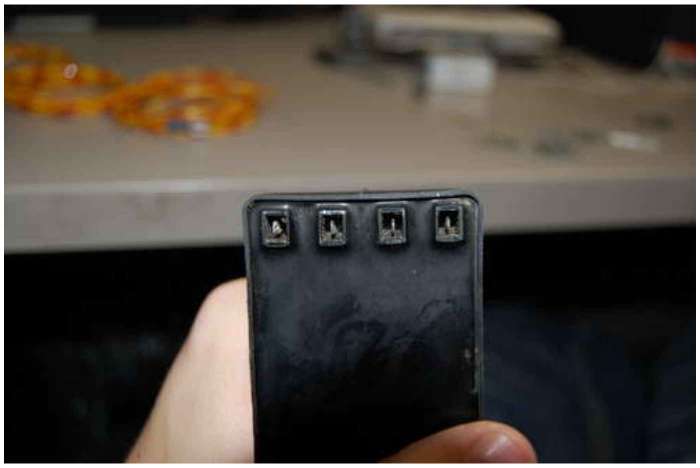
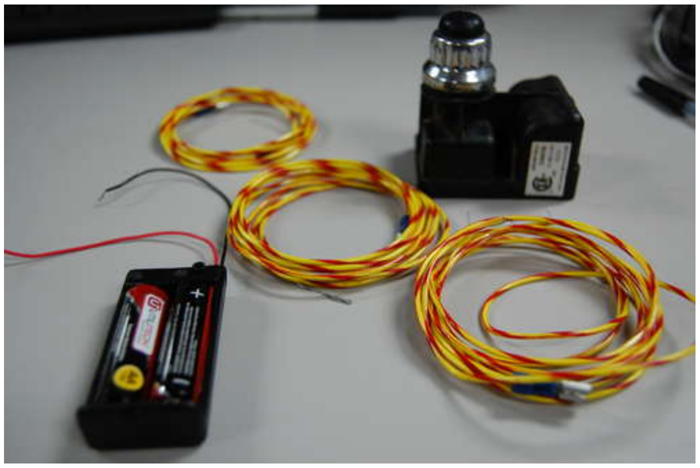


Image Notes
1. Aluminum sheet hooked to one half of the spark generator.
2. Wire hooked to the other side of the spark generator.
3. Exit point for the fuel.
4. What happens when the plastic gets too hot.





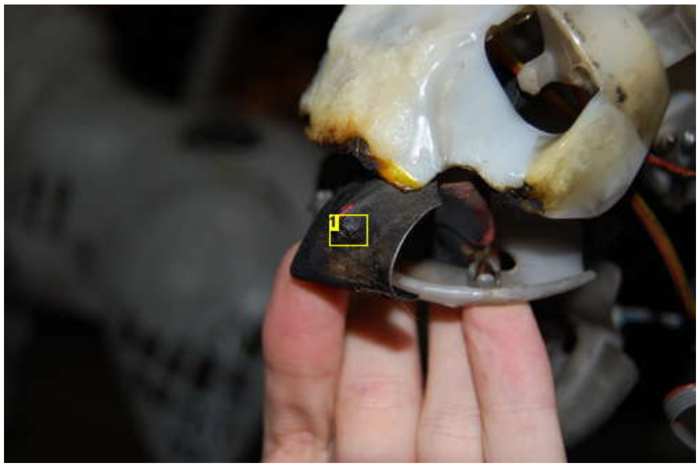


Image Notes
1. Fuel exit tube. Hard to see.

Step 20: Remote fuel trigger

The Remote fuel trigger uses a bowden cable to pull down on a sheet of hinged plexi glass, which pushes the spray nozzle on the fuel can. A bowden cable is the type of cable that is used in bike brakes. This allows it to be flexible in the middle, but still allows it to transfer force in relation to its end points.

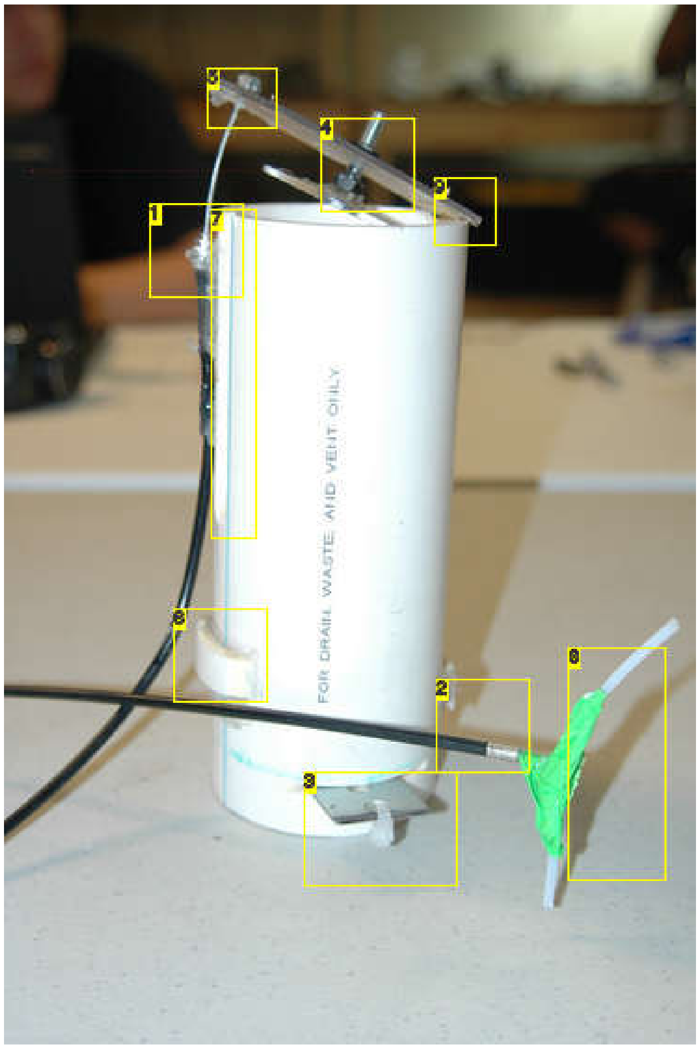


Image Notes
1. bowden cable
2. bowden cable
3. Plate to hold the fuel can inside the tube.
4. two bolts and a few nuts threaded into a piece of plexi glass. This let you change it slightly for variable sized fuel containers.
5. bowden cable Connecting to the chunk of plexy glass to pull/push down on the fuel.
6. this locks on ether side of the tube so that it can lock inside of the pony.
7. Slit in the side of the container so that the fuel tube can come out of the side.
8. This is just a handle to make the cable easier to pull.
9. hinge

Step 21: Follow up

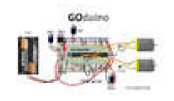
I know that I covered a lot and I am sure I missed something. If something was not clear feel free to ask a question and I will try to clear it up or update the instructable. I also have a short video from not long after maker fair that explains some of the operation of the pony.

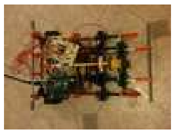
Just remember to keep it safe and have fun.

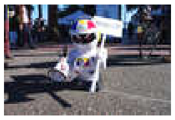
Related Instructables

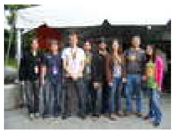
Ardu-pong! the Arduino based pong console by kyle brinkerhoff

arduino Internet PC robot controlled using iphone (video) by Avadhut.Deshmukl

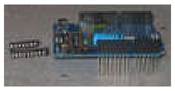

GOduino - The Arduino Uno + Motor Driver clone (Photos) by techbitar


Arduino Robot for lowest cost by doncrush


OCCU(PI) Bot by randofo


Maker Faire NYC 2011 (Photos) by caitlinsdad

How to Make an Obstacles Avoiding Robot-Arduino Style by robotkid249


Tiny Programmer (Photos) by maxhirez

Tweet-a-watt - How to make a twittering power meter...

by **adafruit** on March 29, 2009

Author:adafruit Adafruit Industries
All-original DIY electronics kits - Adafruit Industries is a New York City based company that sells kits and parts for original, open source hardware electronics projects featured on www.adafruit.com as well as other cool open source tronix' that we think are interesting and well-made.

Intro: Tweet-a-watt - How to make a twittering power meter...

Tweet-a-watt - How to make a twittering power meter...

This project documents my adventures in learning how to wire up my home for wireless power monitoring. I live in a rented apartment so I don't have hacking-access to a meter or breaker panel. Since I'm still very interested in measuring my power usage on a long term basis, I built wireless outlet reporters. Building your own power monitor isn't too tough and can save money but I'm not a fan of sticking my fingers into 120V power. Instead, I'll used the existing Kill-a-watt power monitor, which works great and is available at my local hardware store.

My plan is to have each room connected to a 6-outlet power strip which powers all the devices in that room (each kill-a-watt can measure up to 15A, or about 1800W, which is plenty!). That way I can track room-by-room usage, for example "kitchen", "bedroom", "workbench", and "office".

Each wireless outlet/receiver can be built for ~$55 with a few easily-available electronic parts and light soldering, no microcontroller programming or high voltage engineering is necessary!

You can see my setup including graphs and reports at http://twitter.com/tweetawatt

If you'd like to build one for yourself

1. Buy a kit : get all the parts you need, there's a starter kit at the adafruit webshop
2. Make: turn each Kill-a-Watt into a wireless power level transmitter
3. Software: Download & run it on your computer to get data and save it to a file and/or publish it

If you want to know how it was made, check out:

1. Listen: write simple software for my computer (or Arduino, etc) to listen for signal and compute the current power usage
2. Store: Create a database backend that will store the power usage for long-term analysis at http://wattcher.appspot.com
3. View: Graph and understand trends in power usage

Check out the latest readings at http://wattcher.appspot.com

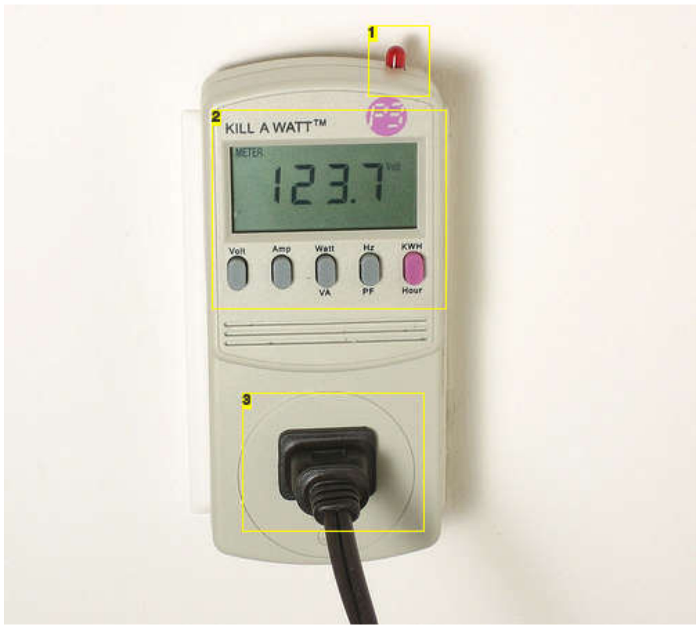
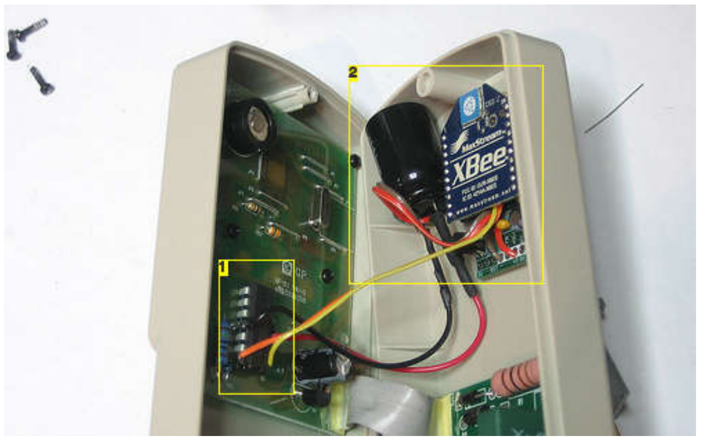


Image Notes
1. Wireless transmission indicator
2. Uses a Kill-a-Watt, available at many hardware or electronic stores
3. Measures up to 15A (1800 Watts!)



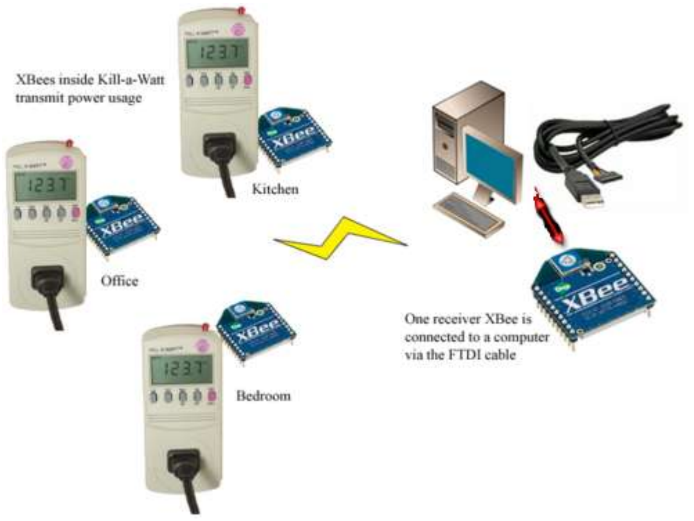

XBees inside Kill-a-Watt transmit power usage
Kitchen
Office
Bedroom
One receiver XBee is connected to a computer via the FTDI cable

Image Notes
1. XBee listens to data coming from the Kill-a-Watt measurement sensors
2. XBee wireless module in an adapter and a few passive components to keep things running smooth

Step 1: Make it!

Before you start...

You should only attempt this project if you are comfortable and competent working with high voltage electricity, electronics and computers. Once the project is complete it is enclosed and there are no exposed high voltages. However, you must only work on the project when its not plugged in and **never ever attempt to test, measure, open, or probe the circuitboards while they are attached to a wall socket.** If something isn't working: stop, remove it from the wall power, then open it up and examine. Yes it takes a few more minutes but it's a lot safer!

Your safety is your own responsibility, including proper use of equipment and safety gear, and determining whether you have adequate skill and experience. Power tools, electricity, and other resources used for this projects are dangerous, unless used properly and with adequate precautions, including safety gear. Some illustrative photos do not depict safety precautions or equipment, in order to show the project steps more clearly. This projects is not intended for use by children.

Use of the instructions and suggestions is at your own risk. Adafruit Industries LLC, disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with applicable laws.

OK, if you agree we can move on!

Make a tweet-a-watt

To make the tweet-a-watt setup, we will have to go through a few steps

1. Prepare by making sure we have everything we need and know the skills necessary to build the project
2. Build the receiver setup by soldering up one of the adapter kits
3. Configure the XBee wireless modems
4. Build the transmitter setup by modifying a Kill-a-Watt to transmit via the XBee
5. Run the software, which will retrieve data and save it to a file, upload it to a database and/or twitter

Step 2: Prep

Tutorials

Learn how to solder with tons of tutorials!
Don't forget to learn how to use your multimeter too!

Tools

There are a few tools that are required for assembly. None of these tools are included. If you don't have them, now would be a good time to borrow or purchase them. They are very very handy whenever assembling/fixing/modifying electronic devices! I provide links to buy them, but of course, you should get them wherever is most convenient/inexpensive. Many of these parts are available in a place like Radio Shack or other (higher quality) DIY electronics stores.

I recommend a "basic" electronics tool set for this kit, which I describe here.

Soldering iron . One with temperature control and a stand is best. A conical or small 'screwdriver' tip is good, almost all irons come with one of these.

A low quality (ahem, $10 model from radioshack) iron may cause more problems than its worth!

Do not use a "ColdHeat" soldering iron, they are not suitable for delicate electronics work and can damage the kit (see here)

Solder . Rosin core, 60/40. Good solder is a good thing. Bad solder leads to bridging and cold solder joints which can be tough to find. Dont buy a tiny amount, you'll run out when you least expect it. A half pound spool is a minimum.

Multimeter/Oscilloscope . A meter is helpful to check voltages and continuity.

Flush/diagonal cutters . Essential for cutting leads close to the PCB.

Desoldering tool . If you are prone to incorrectly soldering parts.

'**Handy Hands' with Magnifying Glass** . Not absolutely necessary but will make things go much much faster.

Check out my recommended tools and where to buy.

Good light. More important than you think.

Step 3: Make the Receiver

Overview

We'll start with the receiver hardware, that's the thing that plugs into the computer and receives data from the wireless power plug. The receiver hardware does 'double duty', it also is used to update the XBee's modems' firmware (which, unfortunately, is necessary because they come from the factory with really old firmware) and configure the modems.

What you'll need

The receiver is essentially, an XBee, with a USB connection to allow a computer to talk to it the XBee.

Name **FTDI cable**
Description A USB-to-serial converter. Plugs in neatly into the Adafruit XBee adapter to allow a computer to talk to the XBee.
Datasheet TTL-232R 3.3V or 5.0V
Distributor Mouser
Qty 1

Name **Adafruit XBee Adapter kit**
Description I'll be using my own design for the XBee breakout/carrier board but you can use nearly any kind as long as you replicate any missing parts such as the3.3V supply and LEDs
You will have 2 adapter kits but you should only assemble one for this part! The other one needs different instructions so just hold off!
Datasheet Webpage
Distributor Adafruit
Qty 1

Name **XBee module**
Description We'll be using the XBee "series 1" point-to-multipoint 802.15.4 modules with a chip antenna part # XB24-ACI-001. They're inexpensive and work great. This project most likely won't work with any other version of the XBee, and certainly not any of the 'high power' Pro types!
Datasheet
Distributor Adafruit
Qty 1

Solder the adapter together!

This step is pretty easy, just go over to the XBee adapter webpage and solder it together according to the instructions!

Remember: You will have 2 adapter kits but you should only solder one of them at this point! The other one needs different instructions so just hold off!

Connect to the XBee

Now its time to connect to the XBees

Find your FTDI cable - use either 3.3V or 5V. These cables have a USB to serial converter chip molded into them and are supported by every OS. Thus configuring or upgrading or connecting is really trivial. Simply plug the cable into the end of the module so that the black wire lines up with GND. There is a white outline showing where the cable connects.

You'll need to figure out which serial port (COM) you are using. Plug in the FTDI cable, USB adapter, Arduino, etc. Under Windows, check the device manager, look for "USB Serial Port"

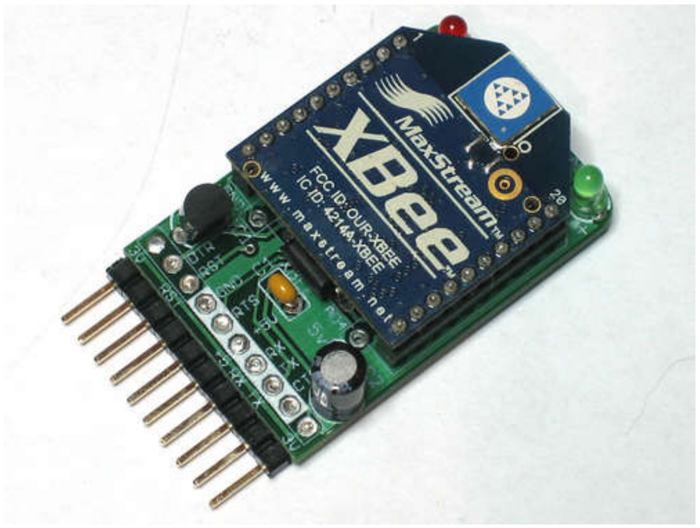
Digi/Maxstream wrote a little program to help configure XBees, its also the only way I know of to upgrade them to the latest firmware. Unfortunately it only runs on Windows. Download X-CTU from Digi and install it on your computer

After installing and starting the program, select the COM port (COM4 here) and baud rate (9600 is default). No flow control, 8N1. Make sure the connection box looks just like the image (other than the com port which may be different)

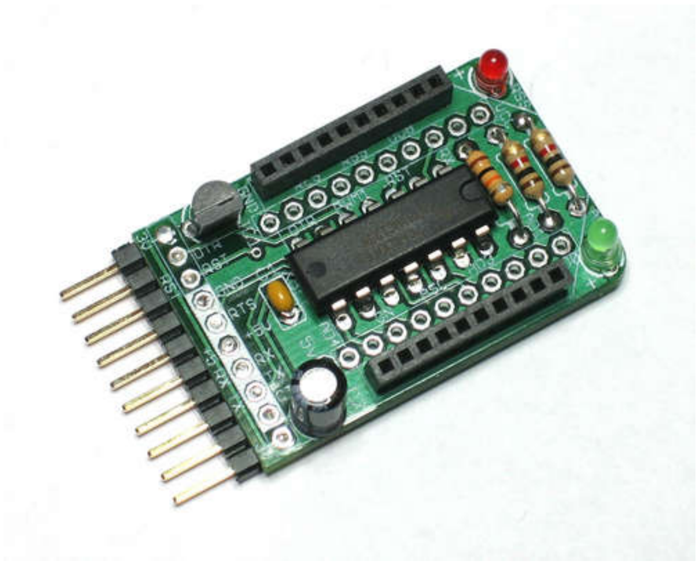
To verify, click **Test** / **Query**

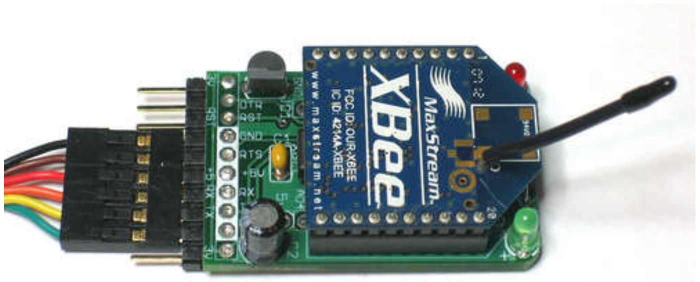
Hopefully the test will succeed. If you are having problems: check that the XBee is powered, the green LED on the adapter board should be blinking, the right COM port & baud rate is selected, etc.

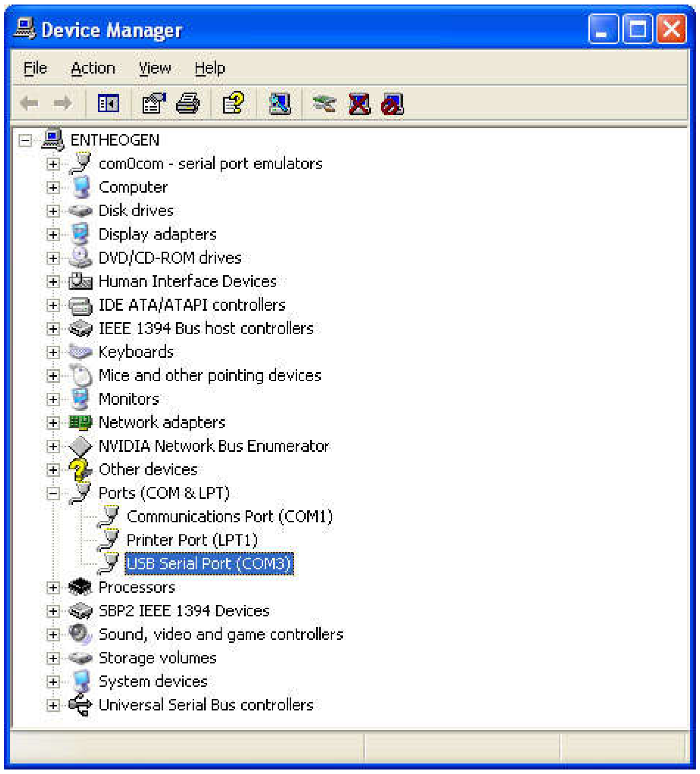
Now unplug the adapter from the FTDI cable, carefully replace the first XBee with the other one and make sure that one is talking fine too. Once you know both XBees are working with the adapter, its time to upgrade and configure them, the next step!


MaxStream
XBee
FCC ID: OUR-XBEE
IC ID: 4214A-XBEE
www.maxstream.net

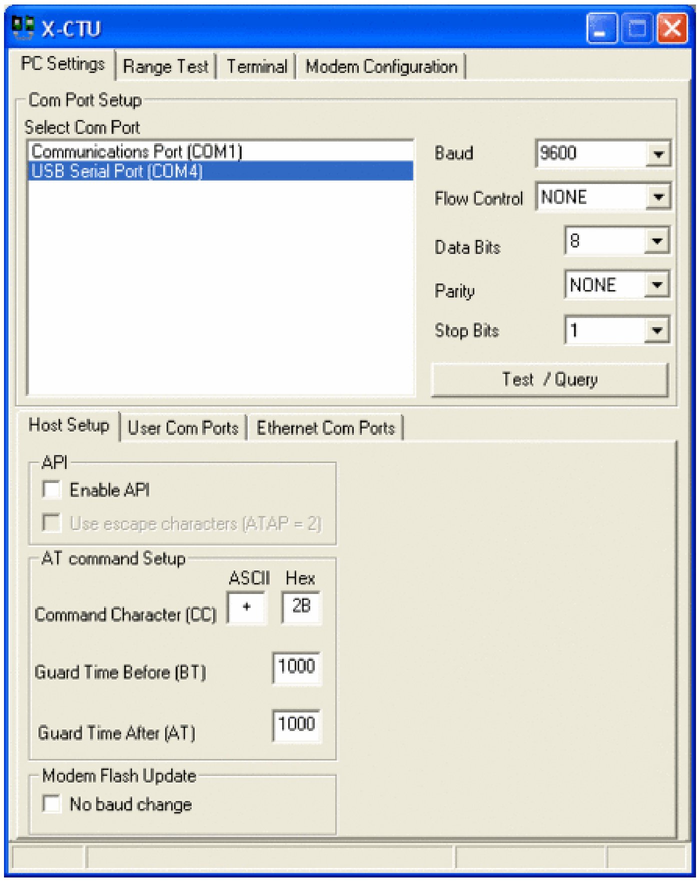
Digi
XBee
FCC ID: OUR-XBEE
IC ID: 4214A-XBEE




XBee
MaxStream
FCC ID: OUR-XBEE
IC ID: 4214A-XBEE
www.maxstream.net



ENTHEOGEN
- com0com - serial port emulators
- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Human Interface Devices
- IDE ATA/ATAPI controllers
- IEEE 1394 Bus host controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- NVIDIA Network Bus Enumerator
- Other devices
- Ports (COM & LPT)
  - Communications Port (COM1)
  - Printer Port (LPT1)
  - USB Serial Port (COM3)
- Processors
- SBP2 IEEE 1394 Devices
- Sound, video and game controllers
- Storage volumes
- System devices
- Universal Serial Bus controllers



Select Com Port
Communications Port (COM1)
USB Serial Port (COM4)

Baud 9600
Flow Control NONE
Data Bits 8
Parity NONE
Stop Bits 1

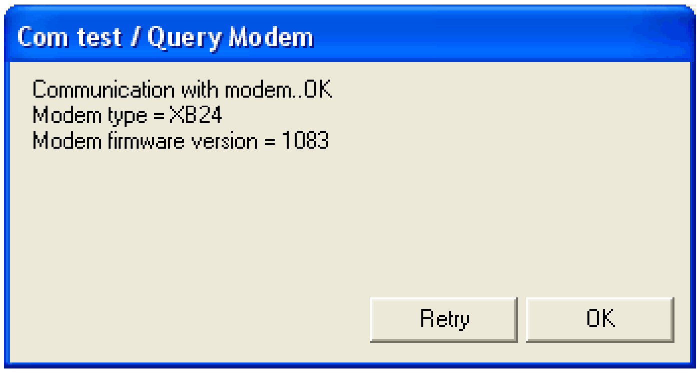
API
Enable API
Use escape characters (ATAP = 2)

AT command Setup

	ASCII	Hex
Command Character (CC)	+	2B

Guard Time Before (BT) 1000
Guard Time After (AT) 1000

Modem Flash Update
No baud change



Communication with modem..OK
Modem type = XB24
Modem firmware version = 1083

Step 4: Configure

Overview

OK so far you have assembled **one** of the XBee adapter boards and connected it to your computer using the FTDI cable. (The other adapter is for later so don't do anything with it yet!) The XBees respond to the X-CTU software and are blinking just fine. Next we will update the firmware

Upgrading the firmware

There's a good chance your XBees are not running the latest firmware & there's a lot of features added, some of which we need to get this project running. So next up is upgrading!

Go to the Modem Configuration tab. This is where the modem is configured and updated

Click **Download new versions** ... and select to download the latest firmwares from the Web

Once you have downloaded the newest firmware, its time to upgrade!

Click on **Modem Parameters** -> "**Read** " to read in the current version and settings

Now you will know for sure what function set, version and settings are stored in the modem

Select from the **Version** dropdown the latest version available

Check the **Always update firmware** checkbox

And click **Write** to initialize and program the new firmware in!

That's it, now you have the most recent firmware for your modem. You should now uncheck the **Always update firmware** checkbox. If you have problems, like for example timing out or not being able to communicate, make sure the RTS pin is wired up correctly as this pin is necessary for upgrading. FTDI cables are already set up for this so you shouldn't have a problem

Rinse & Repeat

Upgrade the firmware on both of the XBees so they are both up to date

At this point it might be wise to label the two XBees in a way that lets you tell them apart. You can use a sharpie, a sticker or similar to indicate which one is the receiver and which is the transmitter

Configure the transmitter XBee

Both XBee's need to be upgraded with the latest firmware but only the transmitter (which is going to be put inside a Kill-a-Watt) needs to be configured. The configure process tells the XBee what pins we want to read the sensor data off of. It also tells the XBee how often to send us data, and how much.

Plug the **transmitter** XBee into the USB connection (put the receiver XBee away) and start up X-CTU or a Terminal program. Connect at 9600 baud, 8N1 parity.Then configure each one as follows:

1. Set the **MY** address (the identifier for the XBee) to **1** (increment this for each transmitter so you can tell them apart, we'll assume you only have one for now)
2. Set the Sleep Mode **SM** to 4 (Cyclic sleep)
3. Set the Sleep Time **ST** to 3 (3 milliseconds after wakeup to go back to sleep)
4. Set the Sleep Period **SP** to **C8** (0xC8 hexadecimal = 200 x 10 milliseconds = 2 seconds between transmits)
5. Set ADC 4 **D4** to **2** (analog/digital sensor enable pin **AD4**)
6. Set ADC 0 **D0** to **2** (analog/digital sensor enable pin **AD0**)
7. Set Samples to TX **IT** to 13 (0x13 = 19 A/D samples per packet)
8. Set Sample Rate **IR** to 1 (1 ms between A/D samples)

if you think there will be more XBee's in the area that could conflict with your setup you may also want to

1. Set the PAN **ID** to a 4-digit hex number (its **3332** by default)

You can do this with X-CTU or with a terminal program such as hyperterm, minicom, zterm, etc. with the command string
ATMY=1,SM=4,ST=3,SP=C8,D4=2,D0=2,IT=13,IR=1
You'll need to start by getting the modem's attention by waiting 10 seconds, then typing in +++ quickly, then pausing for another 5 seconds. Then use **AT** to make sure its paying **AT** tention to your commands

Basically what this means is that we'll have all the XBees on a single PAN network, each XBee will have a unique identifier, they'll stay in sleep mode most of the time, then wake up every 2 seconds to take 19 samples from ADC 0 and 4, 1ms apart. If you're having difficulty, make sure you upgraded the firmware!

Make sure to WRITE the configuration to the XBee's permanent storage once you've done it. If you're using X-CTU click the "Write" button in the top left. If you're using a terminal, use the command **ATWR** !

Note that once the XBee is told to go into sleep mode, you'll have to reset it to talk to it because otherwise it will not respond and X-CTU will complain. You can simply unplug the adapter from the FTDI cable to reset or touch a wire between the RST and GND pins on the bottom edge of the adapter.

Now that the transmitters are all setup with unique **MY** number ID's, make sure that while they are powered from USB the green LED blinks once every 2 seconds (indicating wakeup and data transmit)

Configure the receiver XBee

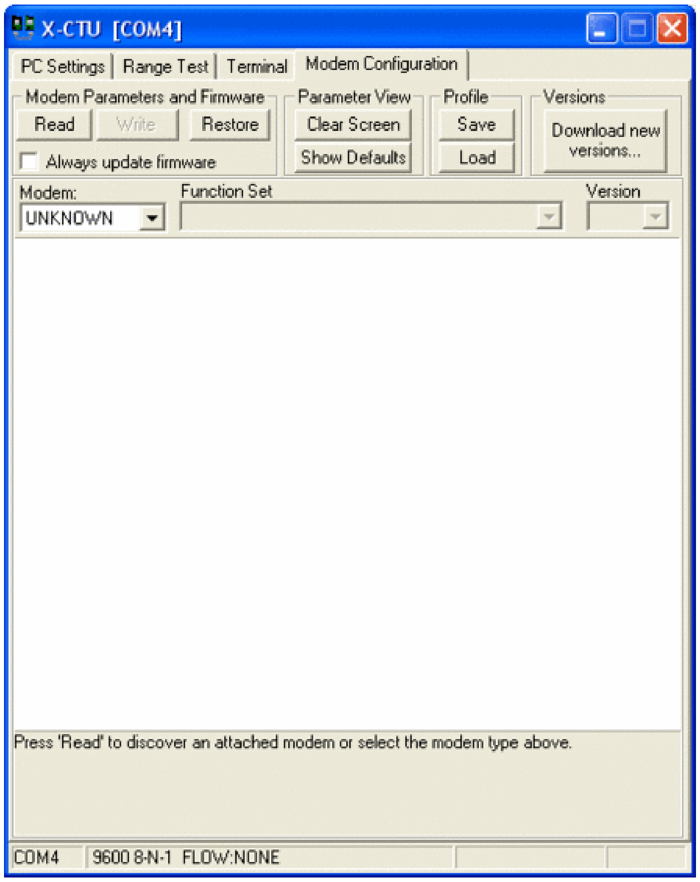
Plug the receiver XBee into the USB connection (put the receiver XBee away) and start up X-CTU . If you set the PAN ID in the previous step, you will have to do the same here

- Set the PAN **ID** to the same hex number as above

If you didn't change the PAN above, then there's nothing for you to do here, just skip this step

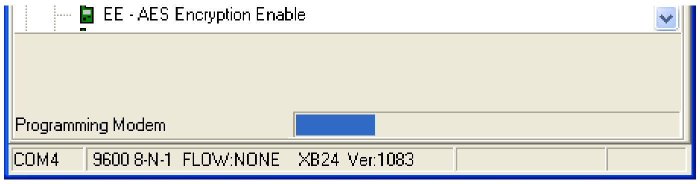
Next!

Now that the XBees are configured and ready, its time to go to the next step where we make the Kill-a-Watt hardware

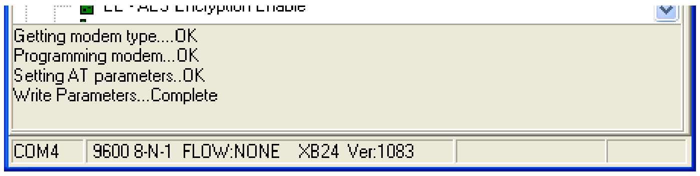


Press 'Read' to discover an attached modem or select the modem type above.

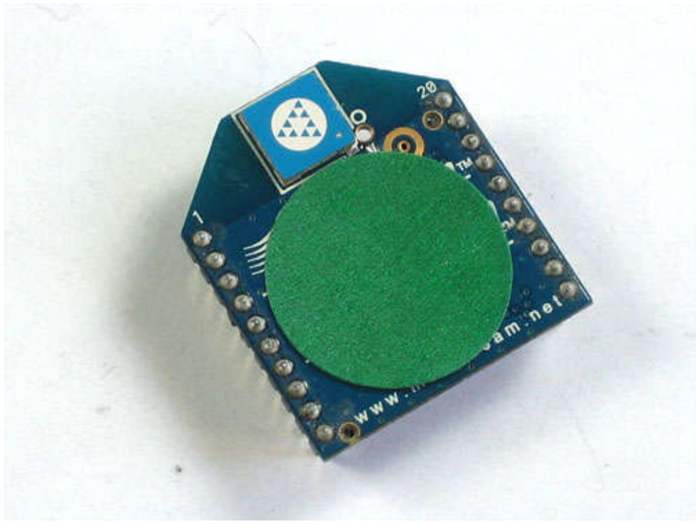
```
COM4  9600 8-N-1  FLOW:NONE
```

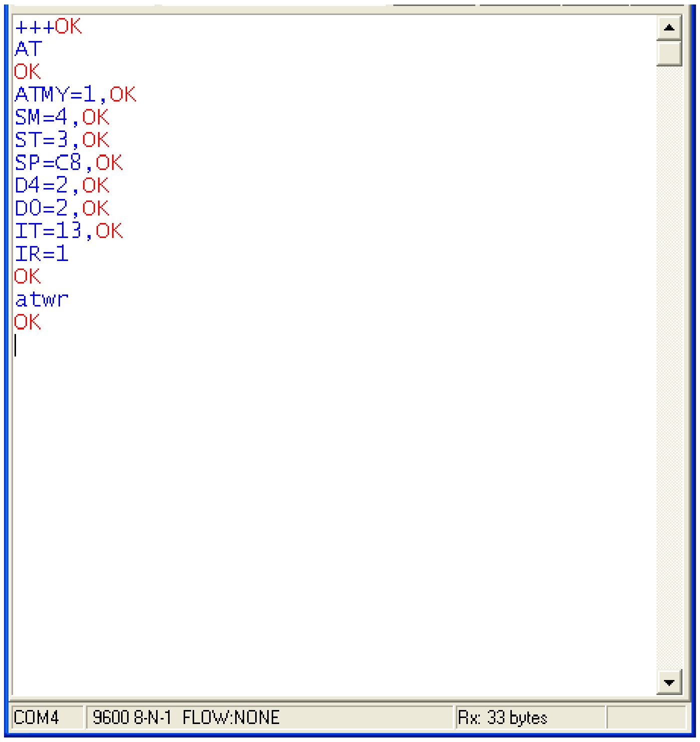


```
Programming Modem
COM4  9600 8-N-1  FLOW:NONE  XB24  Ver:1083
```



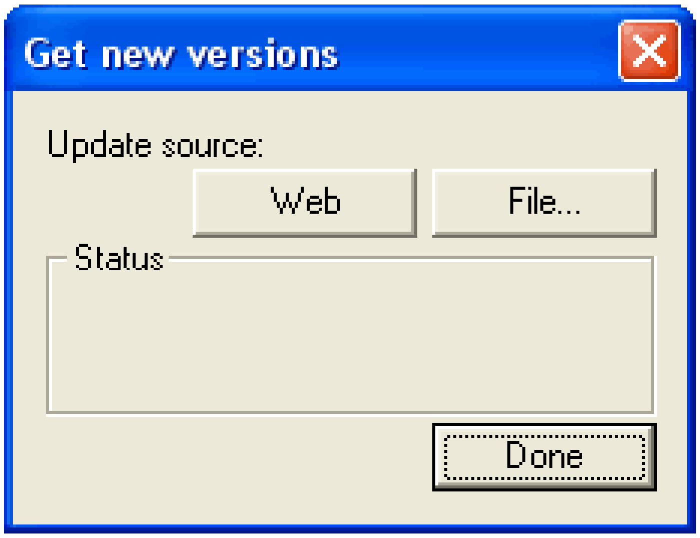
```
Getting modem type....OK
Programming modem...OK
Setting AT parameters..OK
Write Parameters...Complete
COM4  9600 8-N-1  FLOW:NONE  XB24  Ver:1083
```

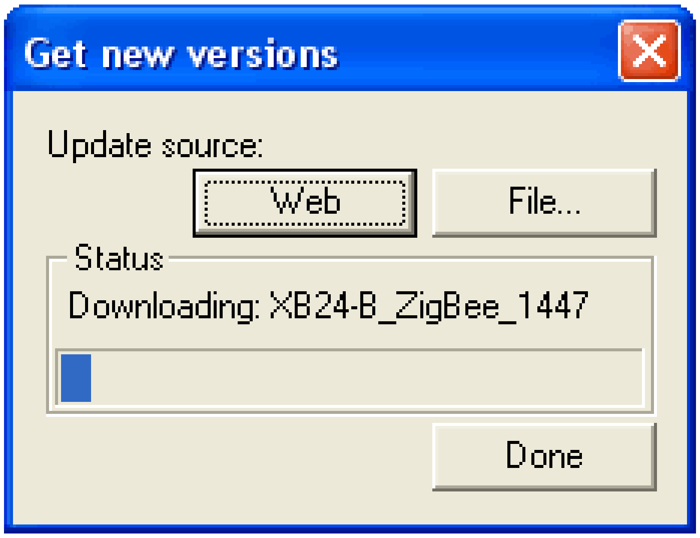




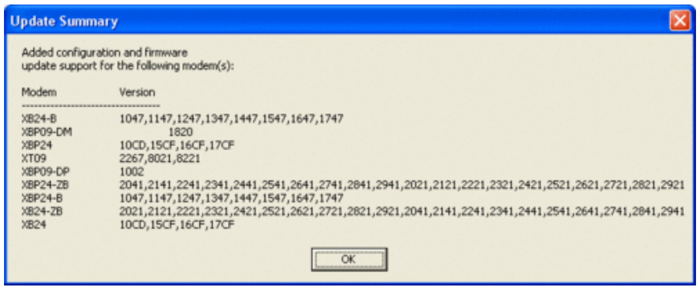
```
+++OK
AT
OK
ATMY=1,OK
SM=4,OK
ST=3,OK
SP=C8,OK
D4=2,OK
D0=2,OK
IT=13,OK
IR=1
OK
atwr
OK
```

```
COM4  9600 8-N-1  FLOW:NONE  Rx: 33 bytes
```



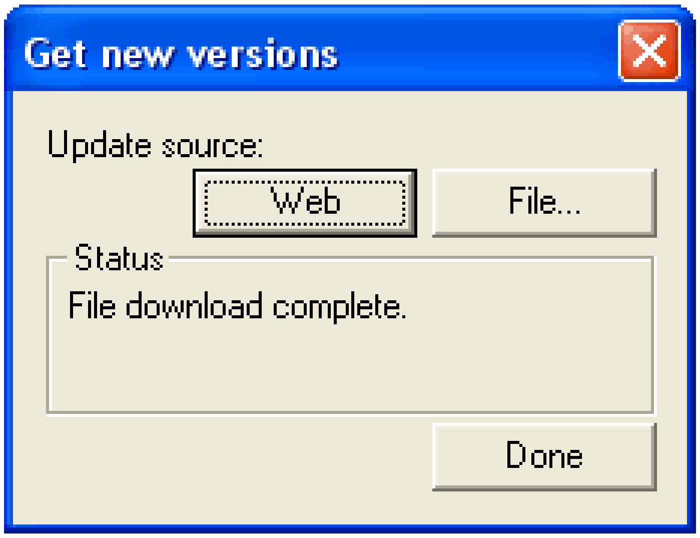


Downloading: XB24-B_ZigBee_1447

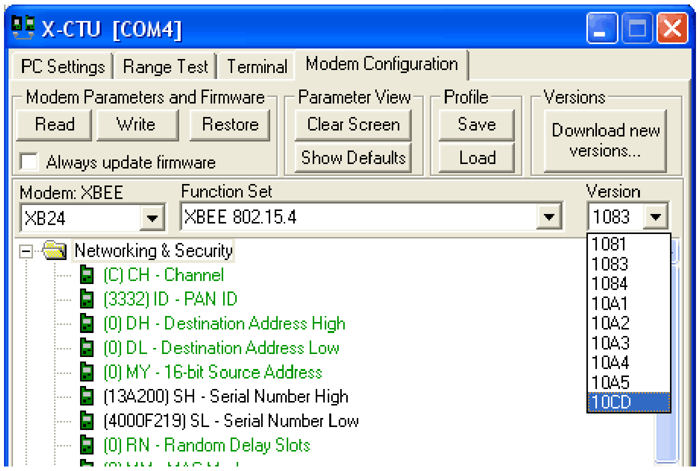


Added configuration and firmware
update support for the following modem(s):

```
Modem           Version
--------------------------------
XB24-B          1047,1147,1247,1347,1447,1547,1647,1747
XBP09-DM                  1820
XBP24           10CD,15CF,16CF,17CF
XT09            2267,8021,8221
XBP09-DP        1002
XBP24-ZB        2041,2141,2241,2341,2441,2541,2641,2741,2841,2941,2021,2121,2221,2321,2421,2521,2621,2721,2821,2921
XBP24-B         1047,1147,1247,1347,1447,1547,1647,1747
XB24-ZB         2021,2121,2221,2321,2421,2521,2621,2721,2821,2921,2041,2141,2241,2341,2441,2541,2641,2741,2841,2941
XB24            10CD,15CF,16CF,17CF
```



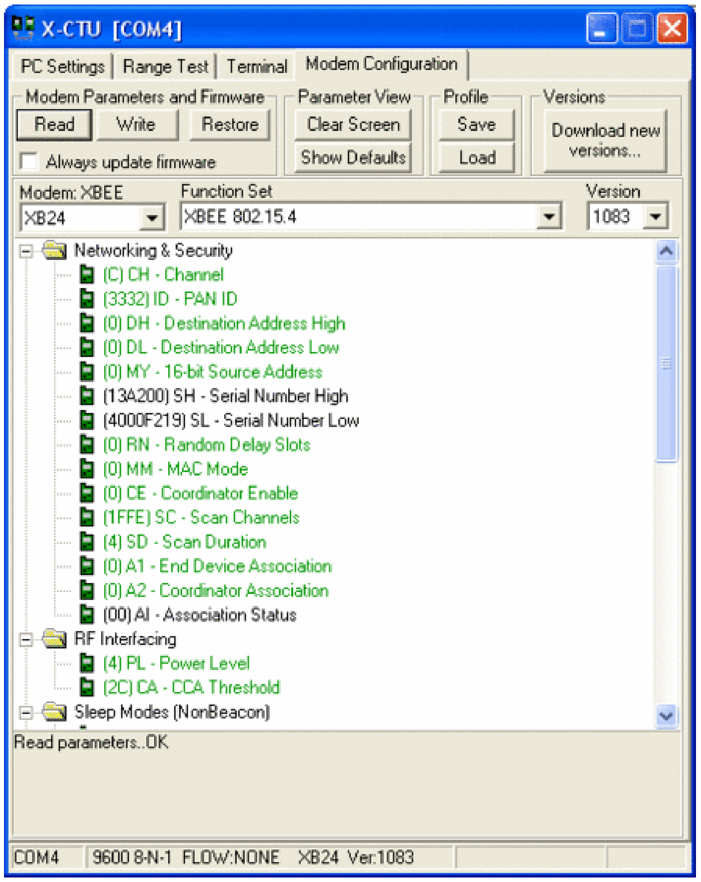
File download complete.



Modem: XBEE: XB24
Function Set: XBEE 802.15.4
Version: 1083

1081
1083
1084
10A1
10A2
10A3
10A4
10A5
10CD

Networking & Security
- (C) CH - Channel
- (3332) ID - PAN ID
- (0) DH - Destination Address High
- (0) DL - Destination Address Low
- (0) MY - 16-bit Source Address
- (13A200) SH - Serial Number High
- (4000F219) SL - Serial Number Low
- (0) RN - Random Delay Slots


X-CTU [COM4]

PC Settings | Range Test | Terminal | Modem Configuration

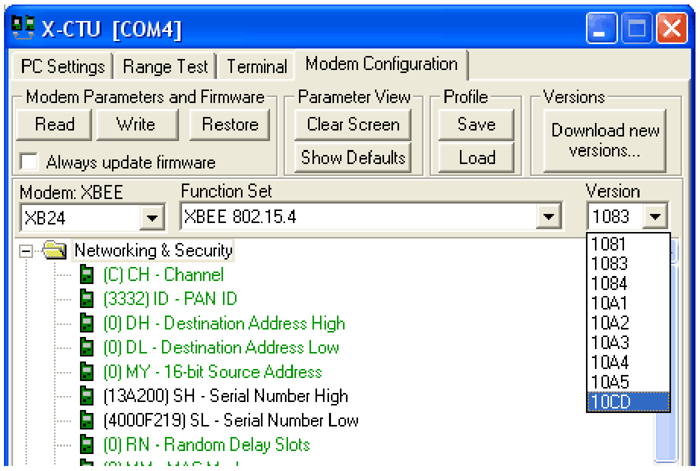
Modem Parameters and Firmware: Read | Write | Restore — Always update firmware
Parameter View: Clear Screen | Show Defaults
Profile: Save | Load
Versions: Download new versions...

Modem: XBEE: XB24 — Function Set: XBEE 802.15.4 — Version: 1083

```
Networking & Security
    (C) CH - Channel
    (3332) ID - PAN ID
    (0) DH - Destination Address High
    (0) DL - Destination Address Low
    (0) MY - 16-bit Source Address
    (13A200) SH - Serial Number High
    (4000F219) SL - Serial Number Low
    (0) RN - Random Delay Slots
    (0) MM - MAC Mode
    (0) CE - Coordinator Enable
    (1FFE) SC - Scan Channels
    (4) SD - Scan Duration
    (0) A1 - End Device Association
    (0) A2 - Coordinator Association
    (00) AI - Association Status
RF Interfacing
    (4) PL - Power Level
    (2C) CA - CCA Threshold
Sleep Modes (NonBeacon)
```

Read parameters..OK

COM4 | 9600 8-N-1 FLOW:NONE | XB24 Ver:1083

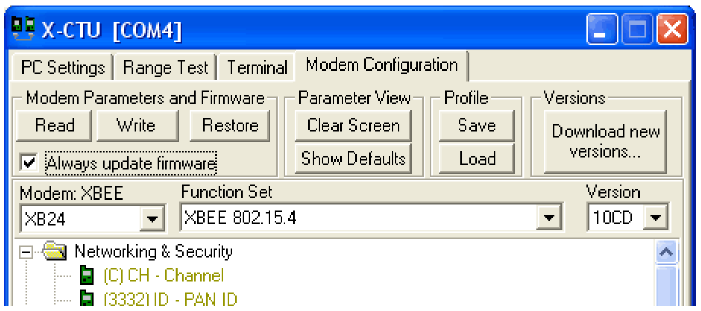

X-CTU [COM4]

PC Settings | Range Test | Terminal | Modem Configuration

Modem: XBEE: XB24 — Function Set: XBEE 802.15.4 — Version: 1083

```
Networking & Security
    (C) CH - Channel
    (3332) ID - PAN ID
    (0) DH - Destination Address High
    (0) DL - Destination Address Low
    (0) MY - 16-bit Source Address
    (13A200) SH - Serial Number High
    (4000F219) SL - Serial Number Low
    (0) RN - Random Delay Slots
```

Version list: 1081, 1083, 1084, 10A1, 10A2, 10A3, 10A4, 10A5, 10CD

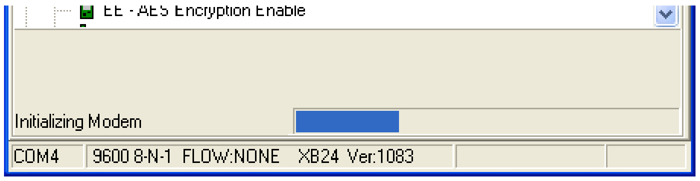

X-CTU [COM4]

PC Settings | Range Test | Terminal | Modem Configuration

☑ Always update firmware

Modem: XBEE: XB24 — Function Set: XBEE 802.15.4 — Version: 10CD

```
Networking & Security
    (C) CH - Channel
    (3332) ID - PAN ID
```


EE - AES Encryption Enable

Initializing Modem

COM4 | 9600 8-N-1 FLOW:NONE | XB24 Ver:1083

Step 5: Solder the Transmitter - parts list

Before you start...

You should only attempt this project if you are comfortable and competent working with high voltage electricity, electronics and computers. Once the project is complete it is enclosed and there are no exposed high voltages. However, you must only work on the project when its not plugged in and **never ever attempt to test, measure, open, or probe the circuitboards while they are attached to a wall socket.** If something isn't working: stop, remove it from the wall power, then open it up and examine. Yes it takes a few more minutes but it's a lot safer!

Your safety is your own responsibility, including proper use of equipment and safety gear, and determining whether you have adequate skill and experience. Power tools, electricity, and other resources used for this projects are dangerous, unless used properly and with adequate precautions, including safety gear. Some illustrative photos do not depict safety precautions or equipment, in order to show the project steps more clearly. This projects is not intended for use by children.

Use of the instructions and suggestions is at your own risk. Adafruit Industries LLC, disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with applicable laws.

OK, if you agree we can move on!

Transmitter partslist

For each outlet you want to monitor, you'll need:

Name: **Kill-a-Watt**
Description: "Off the shelf" model P4400 power monitor
Datasheet: P3 Kill-a-watt
Distributor: Lots! Also check hardware/electronics stores
Qty: 1

Name: **Adafruit XBee Adapter**
Description: I'll be using my own design for the XBee breakout/carrier board but you can use nearly any kind as long as you replicate any missing parts such as the3.3V

supply and LEDs
Datasheet: Webpage
Distributor: Adafruit
Qty: 1

Name: **XBee module**
Description: We'll be using the XBee "series 1" point-to-multipoint 802.15.4 modules with a chip antenna part # XB24-ACI-001. They're inexpensive and work great. This project most likely won't work with any other version of the XBee, and certainly not any of the 'high power' Pro types!
Distributor: Adafruit
Qty: 1

Name: **D3**
Description: 1N4001 diode. Any power diode should work fine. Heck, even a 1n4148 or 1n914 should be OK. But 1N4001 is suggested and is in the kit.
Datasheet: Generic 1N4001
Distributor: Digikey Mouser
Qty: 1

Name: **D2**
Description: Large diffused LED, for easy viewing. The kit comes with green.
Qty: 1

Name: **C3**
Description: 220uF, 4V or higher (photo shows 100uF)
Datasheet: Generic
Distributor: Digikey Mouser
Qty: 1

Name: **C4**
Description: 10,000uF capacitor (wow!) / 6.3V (photo shows a mere 2200uF) Try to get 16mm diameter, 25mm long
Datasheet: Generic
Distributor: Digikey [Mouser]
Qty: 1

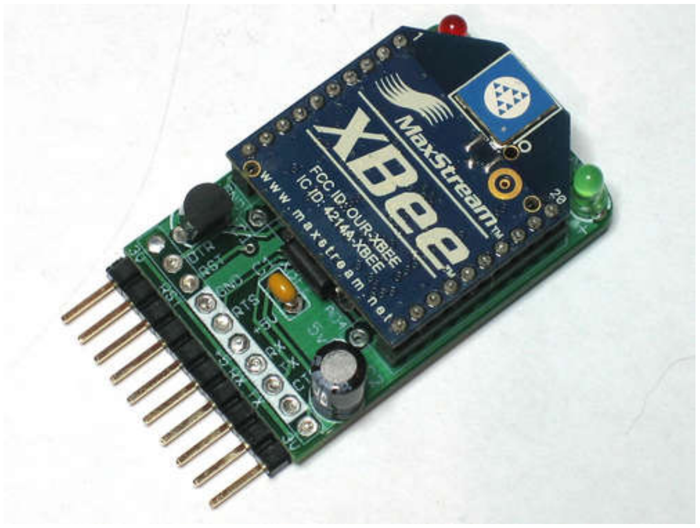
Name: **R4 R6**
Description: 10K 1/4W 1% resistor (brown black black red gold) or 10K 1/4W 5% resistor (brown black orange gold). 1% is preferred but 5% is OK
Datasheet: Generic
Distributor: Mouser Digikey
Qty: 2

Name: **R3 R5**
Description: 4.7K 1/4W 1% resistor (yellow violet black brown gold) or 4.7K 1/4W 5% resistor (yellow violet red gold). 1% is preferred but 5% is OK.
Datasheet: Generic
Distributor: Mouser Digikey
Qty: 2

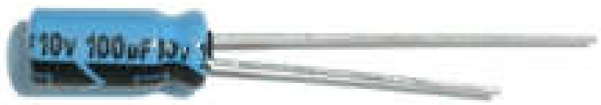
Name: **Ribbon cable**
Description: Ribbon cable, or other flexible wire, at least 6 conductors, about 6" long
Datasheet: Generic Ribbon
Distributor: Digikey
Qty: 6"

Name: **Heat shrink**
Description: Heat shrink! A couple inches of 1/8" and 1/16" each
Datasheet: Generic

It will run you about $50-$60 for each outlet



Digi
XBee
FCC ID: OUR-XBEE
IC ID: 4214A-XBEE


10v 100uF 10v

16v 2200uF 16v

Step 6: Transmitter Schematic

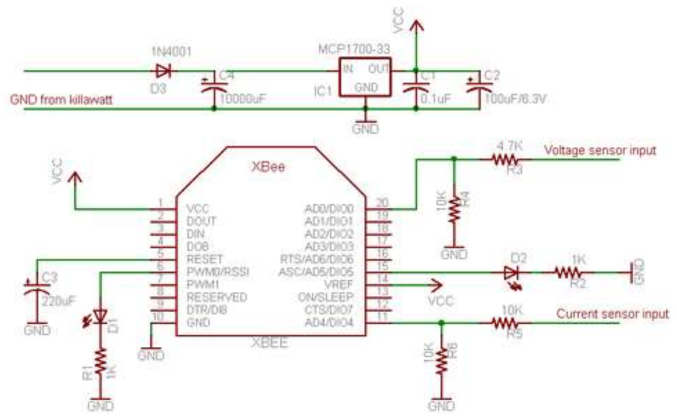
The XBee radio does all of the hard work, it listens on two analog input ports (AD0 and AD4) for voltage and current data. Then it transmits that information wirelessly to the host computer receiver XBee. There are a few we have to engineer around to make it Work:

1. We want to run the XBee off the Kill-a-Watt's internal power supply. However its current limited and wont provide 50mA in a burst when the XBee transmits. We solve this by adding a simple 'rechargeable battery' in the form of a really large capacitor **C4** .

2. The Kill-a-Watt runs at 5V but XBees can only run at 3.3V so we have a voltage regulator **IC1** and two capacitors two stabilize the 3.3V supply, **C1** and **C2** .

3. The XBee will transmit every few seconds, even while the capacitor is charging. This means that it will keep draining the capacitor, resetting, and trying again, basically freaking out while the power supply is still building. We prevent this by adding another fairly big capacitor **C3** on the reset line. This slows down the XBee, delaying the startup by a few seconds & keeps the XBee from starting up till we have solid power.

4. The XBee analog sensors run at 3.3V but the Kill-a-Watt sensors run at 5V. We use simple voltage dividers **R3** /**R4** and **R5** /**R6** to reduce the analog signal down to a reasonable level



Step 7: Assemble and create the transmitter - 1

Open up your kit and get out the parts for the transmitter. Remember that we'll be using most of but not all of an XBee adapter kit. The two small LEDs, the 74HC125N chip, a 10K and 1K resistor are not used and you should put them aside for a future project so you don't accidentally use them here.

Check to make sure you've got everything you need. The only thing not shown here is the XBee radio and Kill-a-Watt.

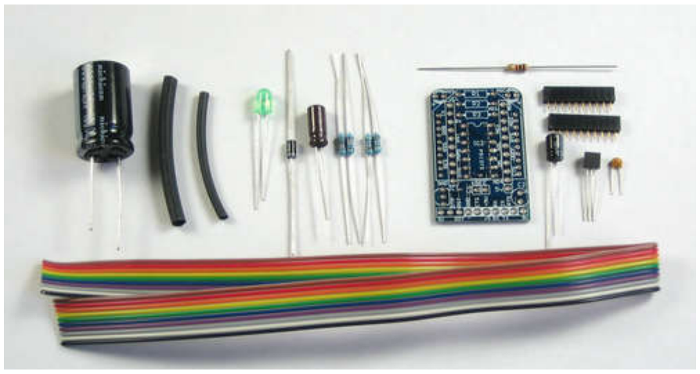
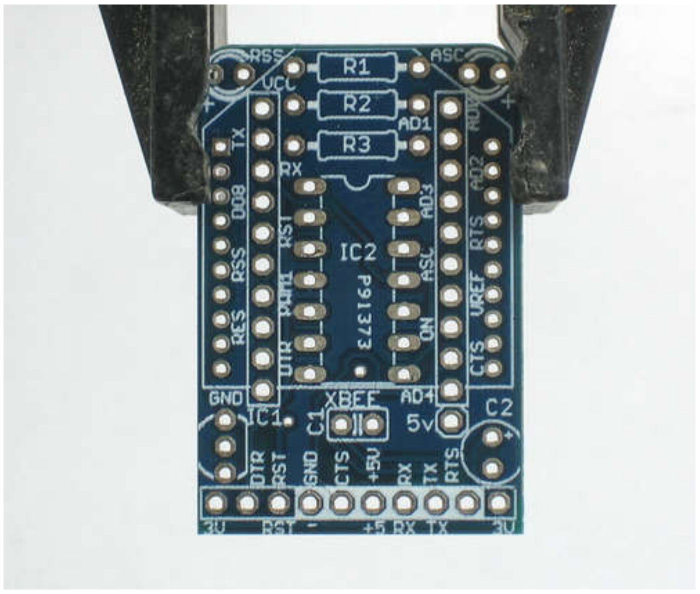
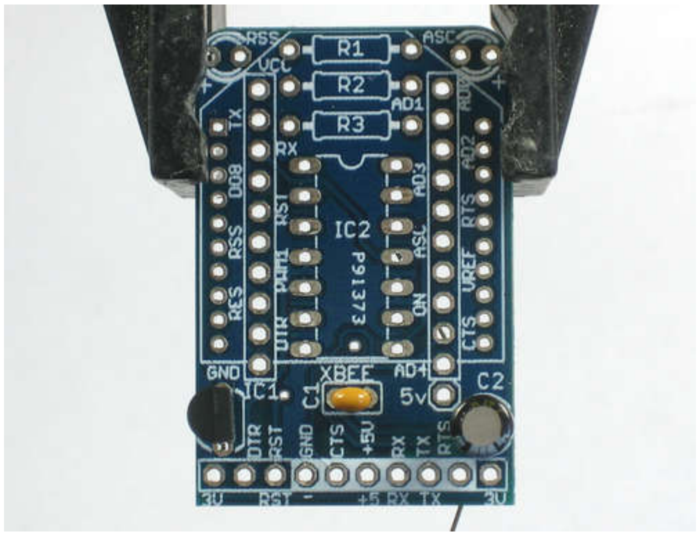
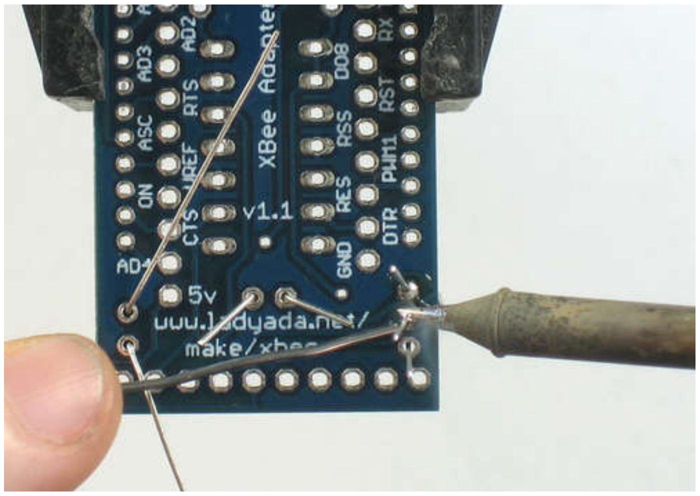
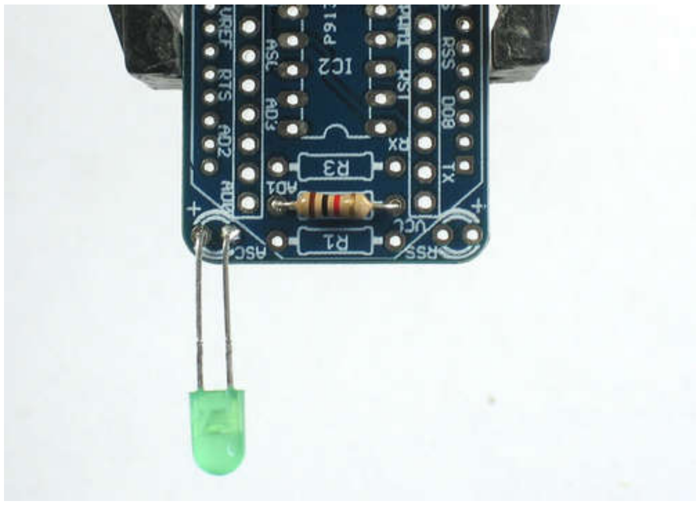
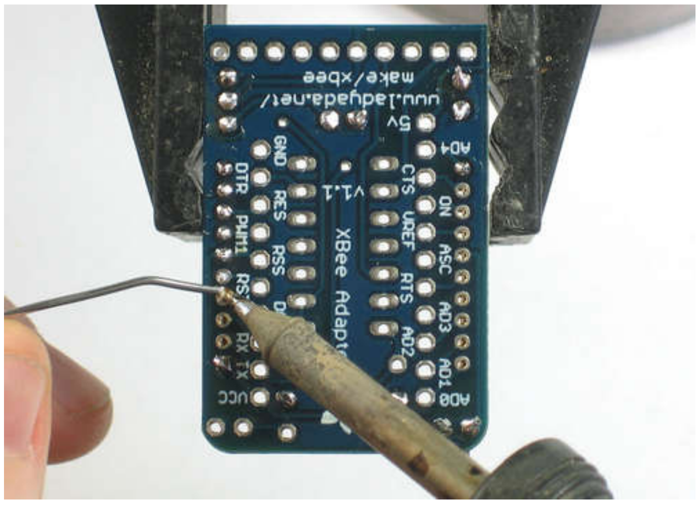
Place the PCB of adapter kit and get ready to solder by heating up your soldering iron, and preparing your hand tools

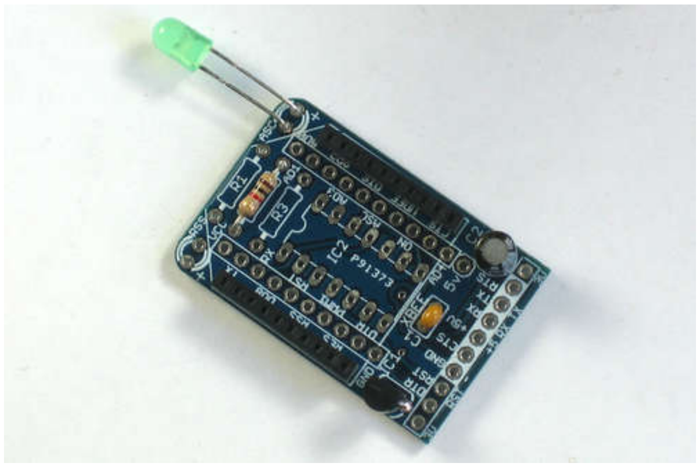
We'll start by soldering in the 3.3V regulator, which is identical to the standard XBee Adapter kit you made in the receiver instructions. Don't forget to check the polarity of **C2** and that **IC1** is in the right way. Then solder and clip the three components.

Now we will veer from the standard XBee adapter instructions and add a much larger LED on the ASC line so that we can easily see it blinking when its in the Kill-a-Watt. Make sure to watch for the LED polarity, because a backwards LED will make debugging very difficult. The longer lead goes in the + marked solder hole.

Give the LED about half an inch of space beyond the end of the PCB as shown. Also solder in the matching 1K resistor **R2**

Solder in the two 2mm 10pin female headers in the adapter kit. Be careful with the solder so that you don't accidentally fill the female header. Use a sparing amount to make sure there's a connection but its not overflowing



Step 8: Assemble and create the transmitter - 2

Now its time to prepare the wires we need for the next few stops. Use your diagonal cutters to notch off the brown, red, orange and yellow wires from the end of the rainbow ribbon cable in the kit.

Then tear off the four wires from the rest of the cable.

Do the same for the black and white wires and the single green wire. Then cut the green wire so its only about 1.5" long. You should now have 3 strips of wire, one 6" with 4 conductors, one 6" with 2 conductors and one 1.5" with 1 conductor

Use wirestrippers to strip the ends of the green wire, 1/4" from the ends

Then tin the green wire by heating the ends of the wire and applying a little solder to bind together the stranded wire.

Use the green wire to create a jumper between the **VREF** pin, 7th from the top on the right and the **VCC** pin on the top left.

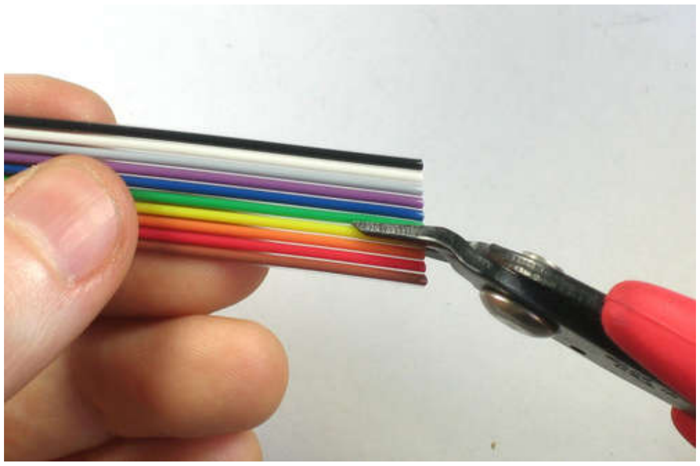
Double check to make sure you get this right! Then solder it in place. This will set the reference point of the analog converter to 3.3V

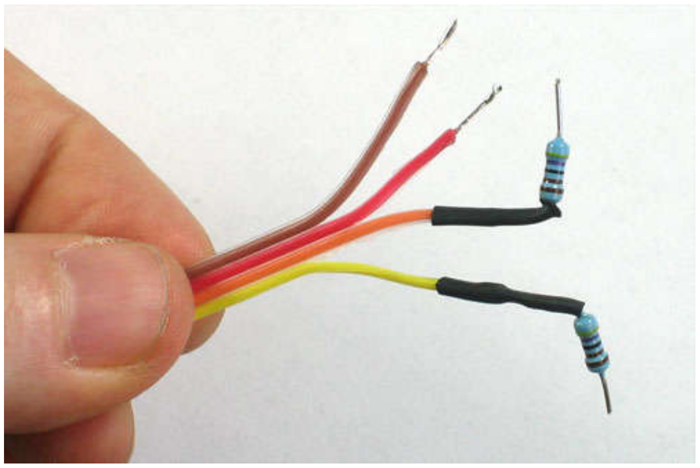
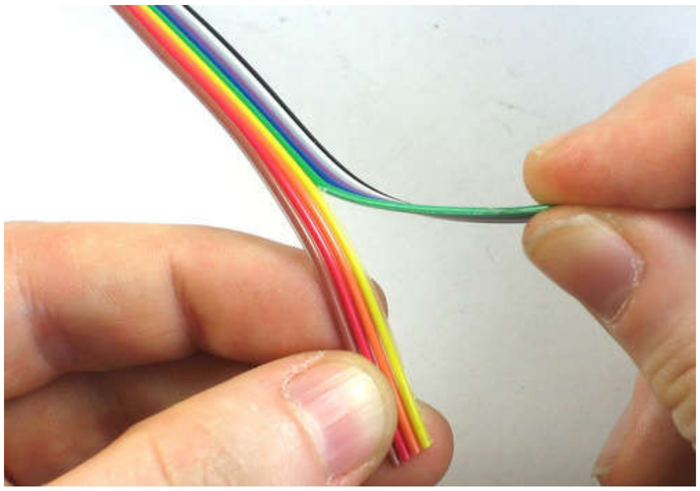
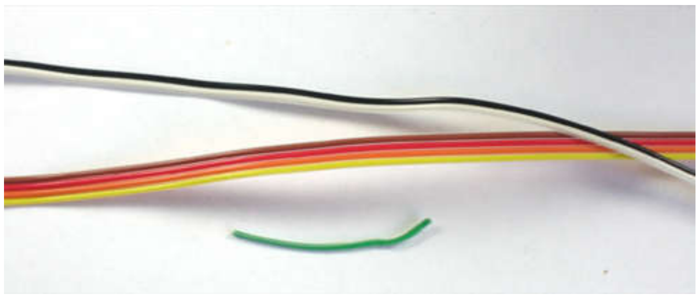
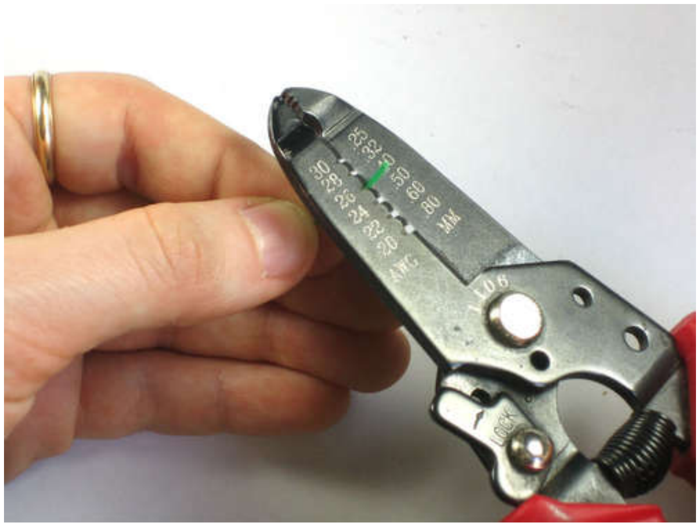
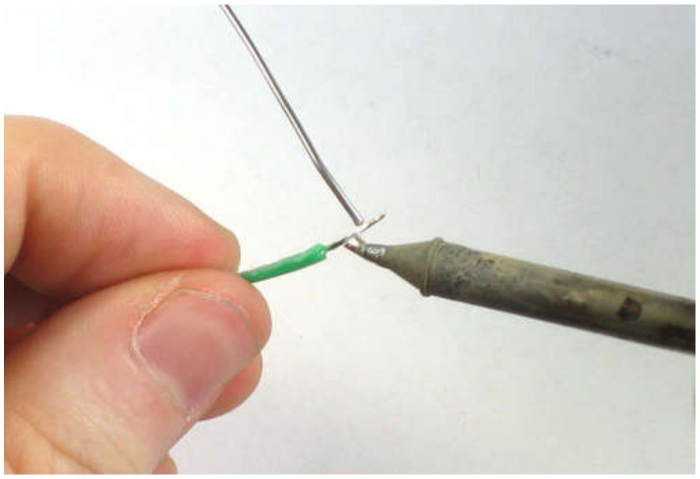
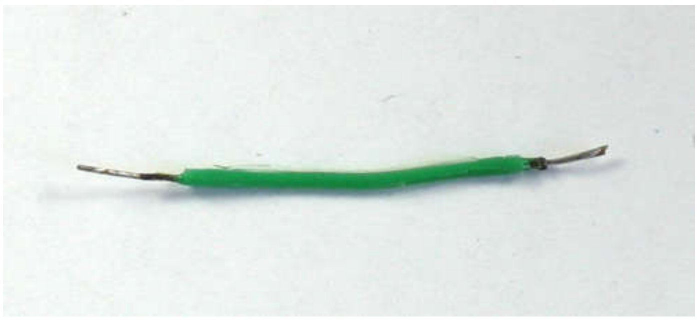
Go back to the 4-piece ribbon cable. Split the ends with the diagonal cutter, then strip and tin all 8 ends.

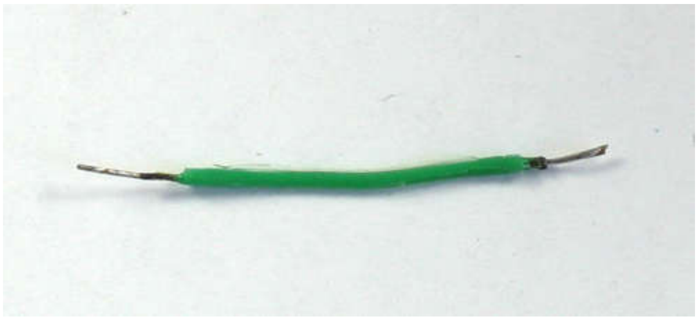
Put a 4.7K resistor in a vise or holder, then clip one end off and tin it just like the wires.

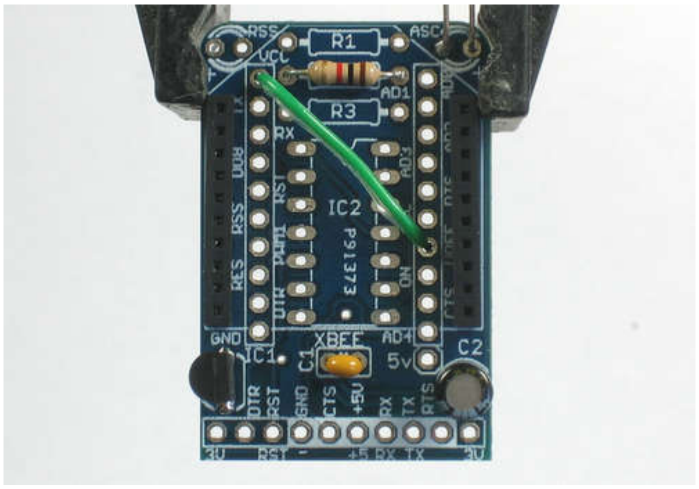
Cut a 1/2" piece of 1/16" heat shrink and slip it onto the yellow wire, making sure there's clearance between the heatshrink and the end of the wire. Then solder the yellow wire to the 4.7k resistor.

Do the same for the orange wire and the other 4.7K resistor. Use a heat source (a heat gun or hair drier is perfect) to shrink the heatshrink over the soldered wire/resistor joint.Then bend the resistor 90degrees and clip the other end of the 4.7k resistors







AWG
MM
LOCK







Step 9: Assemble and create the transmitter - 3

Now we will build the voltage divider. Take the two 10K resistors and connect them as shown. One goes from **AD0** and one from **AD4** . Both then connect to ground. Conveniently, the chip we are not using had grounded pins so we can 'reuse' those pins.

Now comes the tricky part. We want to connect the other end of the 4.7K resistor to the AD0 pin but the 10K resistor is already there. Use your soldering iron to melt a blob of solder onto the top of the 10K resistor and then piggyback the 4.7K resistor by soldering to the top of the 10K resistor.

Solder the orange wire to the **AD0** pin, the yellow to the **AD4**

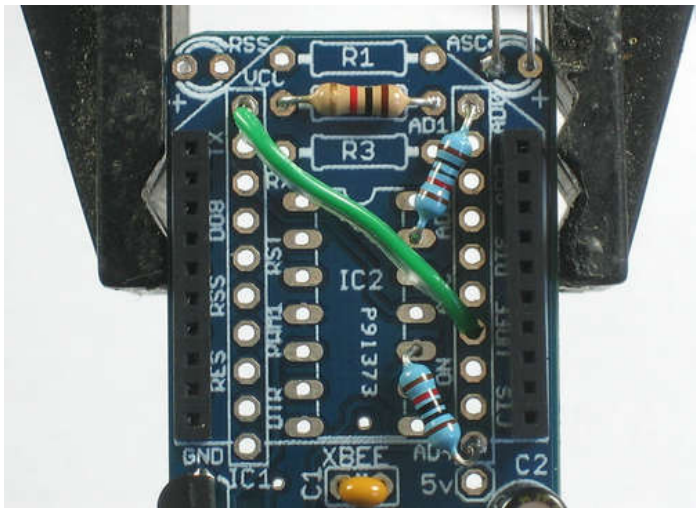
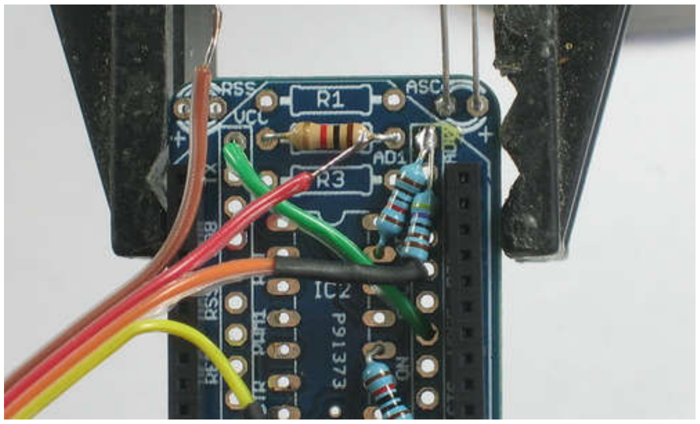
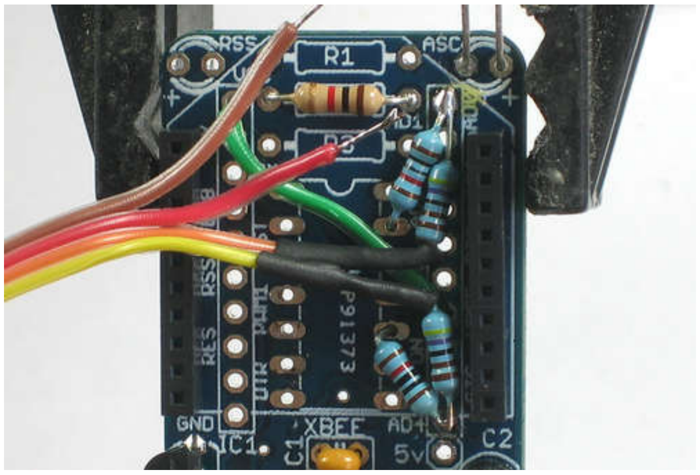
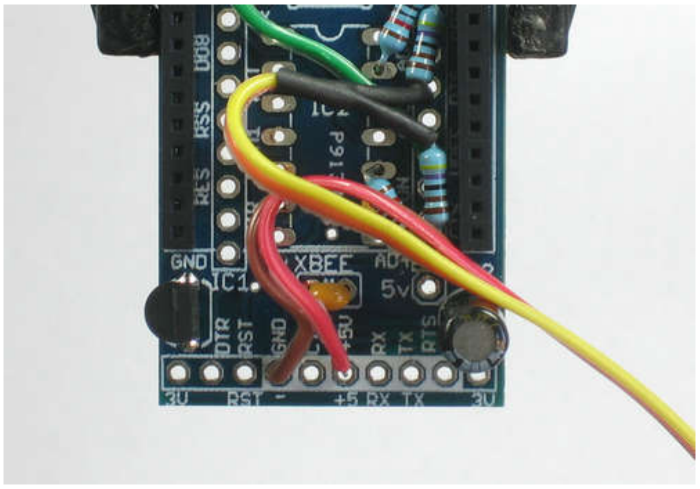
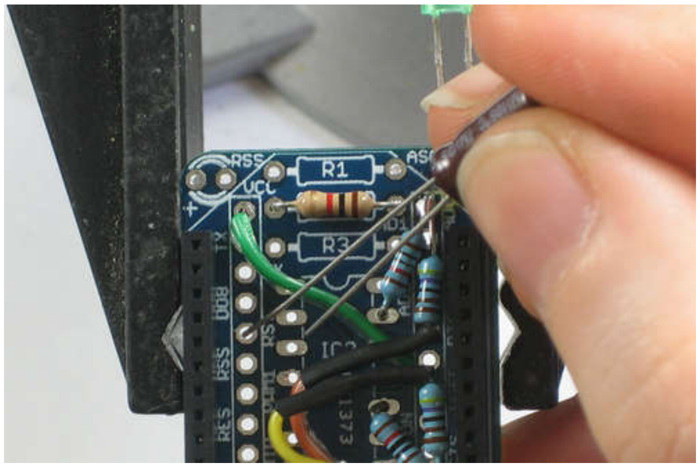
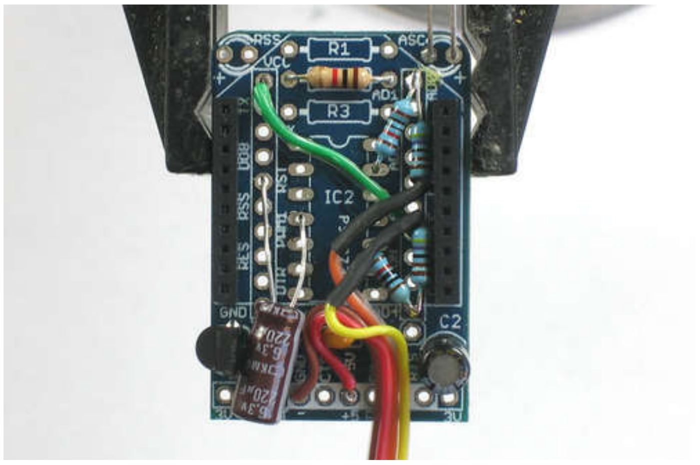
The other two wires are for carrying power. The red wire should be soldered to the +5V pin on the bottom of the adapter PCB. The brown wire to the **GND** pin.

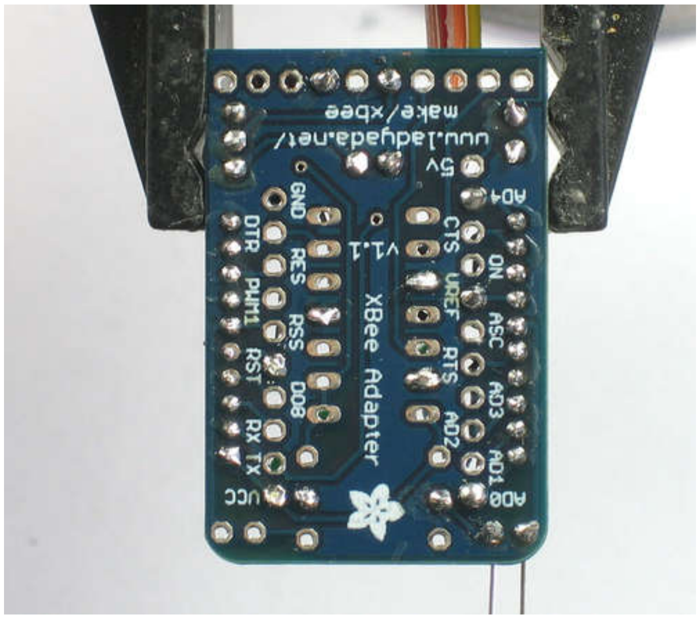
We're nearly done with the adapter soldering. Lastly is the 220uF reset capacitor. We'll connect this to the RST pin, 5th from the top on the left. Make sure the long lead is connected to the RST pin and the shorter lead goes to the 4th pin of where the chip would go. Check the photo on the left to make sure you've got it in right.

The capacitor wont fit underneath the XBee module so give it some lead length so that the cylindrical bulk is next to the 3.3V regulator.

For reference, the images below show what the back should look like.

... and what it should look like with the XBee modem installed. Make sure the pins on the XBee line up with the header.



Step 10: Assemble and create the transmitter - 4

Now replace the PCB with the huge capacitor.

Clip the long leads down. You'll need to use the "-" stripe to keep track of which pin is negative and which is positive.

Tin both leads with solder.

Solder the other end of the red ribbon wire (that goes to +5V on the XBee adapter) to the positive pin of the capacitor.

Then solder the brown wire (that goes to GND on the XBee adapter) to the negative pin.

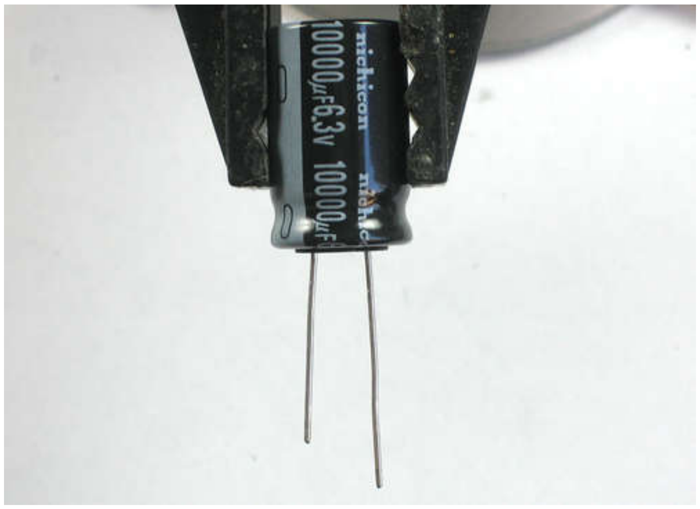
Clip the cathode lead down of the 1N4001 diode, that's the end with the white stripe. on it. Solder the diode so that the white-stripe side is connected to the positive pin of the big capacitor.

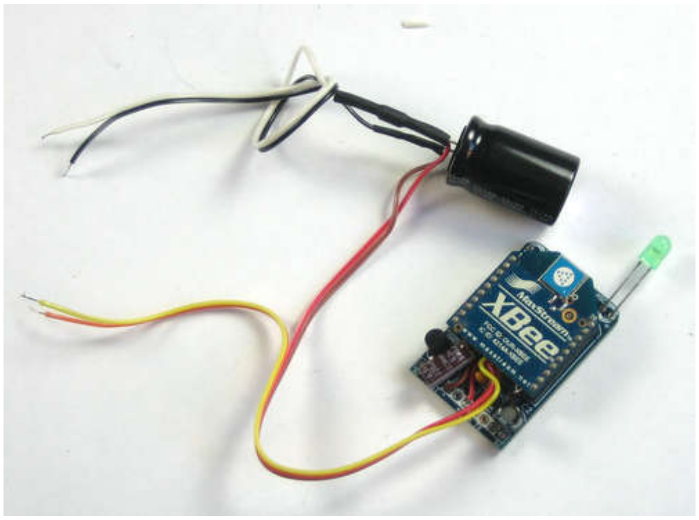
Take the black and white ribbon from earlier. Split, strip and tin the four ends. Cut a 1" piece of 1/8" heatshrink and slip it onto the white wire. Slip a 1/2" piece of 1/16" heat shrink onto the black wire.

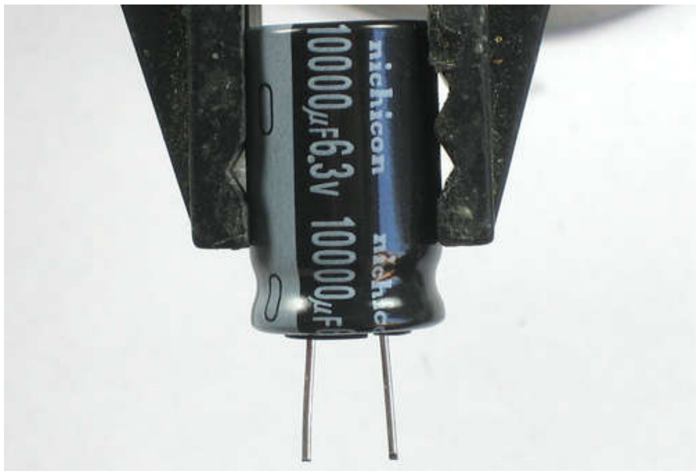
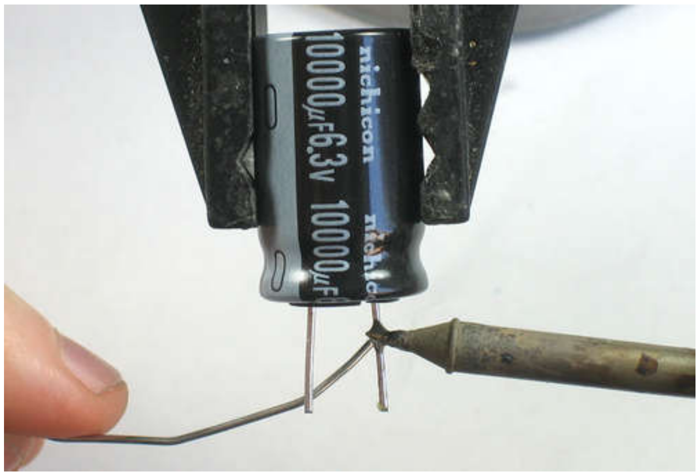
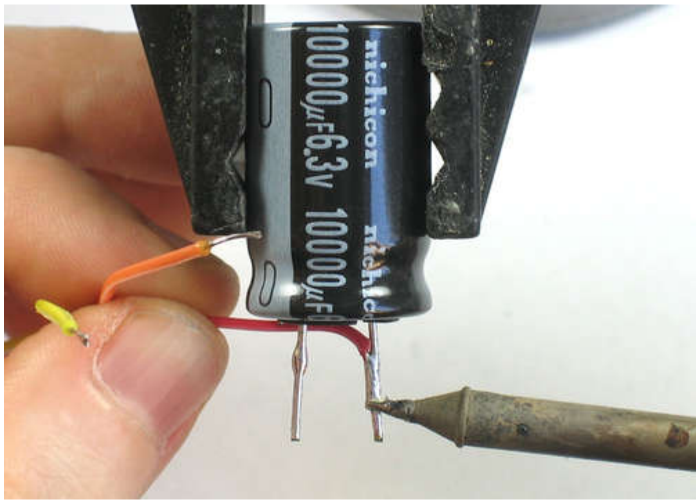
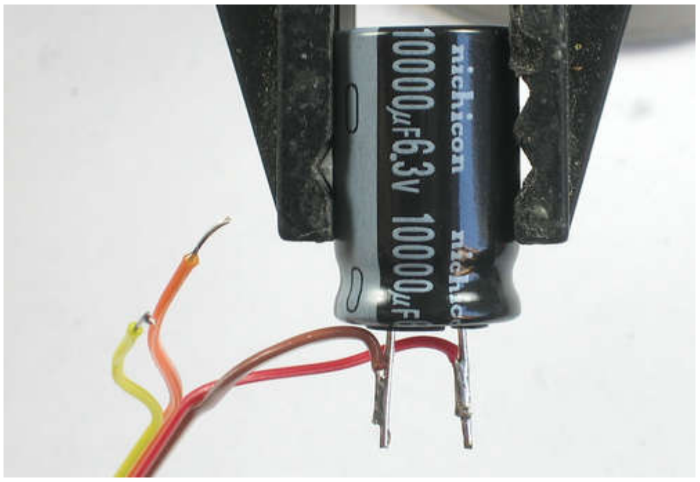
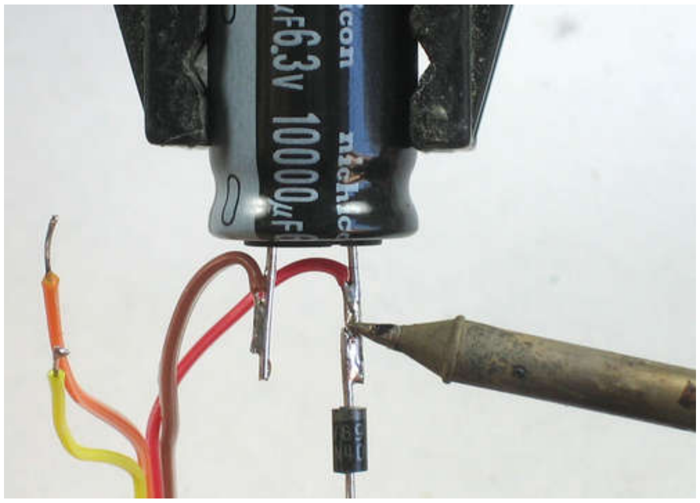
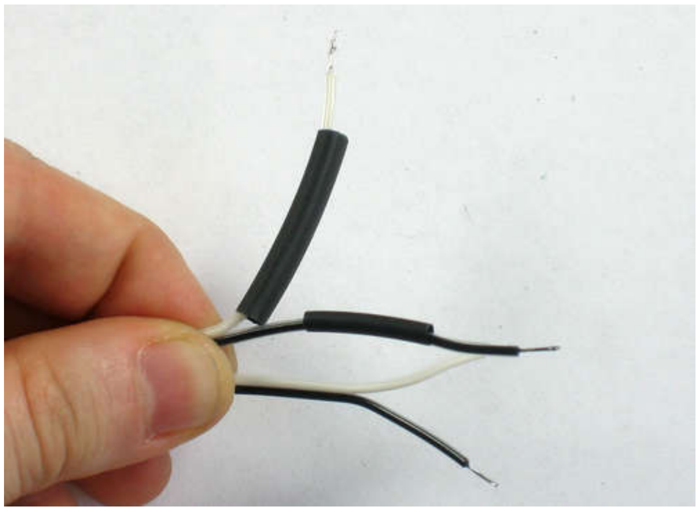
Clip the other end of the diode (the side without a white stripe) and solder the white wire to it. Solder the black wire to the negative pin of the big capacitor.

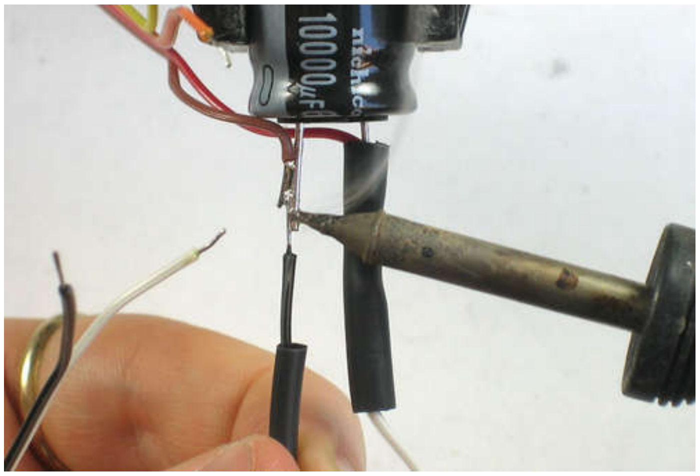
Now shrink the heatshrink so that the capacitor leads and diode are covered.

All right, here is what you should have, an adapter with two sensor lines (orange and yellow) hanging off and two power lines (red and brown) that are connected to the big capacitor. Then there are two black&white wires connected to the capacitor, the white one through a diode.






nichicon
10000µF6.3V 10000µF

nichicon
10000µF6.3V 10000µF

nichicon
10000µF6.3V 10000µF

nichicon
10000µF6.3V 10000µF

µF6.3V 10000µF




Step 11: Assemble and create the transmitter - 5

Now its time to open the Kill-a-Watt! There are only 3 screws that hold it together, and they are found on the back.

Now its time to open the Kill-a-Watt! There are only 3 screws that hold it together, and they are found on the back.

Use a 3/8 drill bit to make a hole near the right corner of the case back. This is what the LED will stick out of. (Ignore the white tape and #4, this is a recycled kill-a-watt :)

Now find the LM2902N chip. This is a quad op-amp that senses the power line usage. We're going to piggy-back right on top of it, and borrow the ground, 5V power and 2 sensor outputs!

With your soldering iron, melt a bit of solder on pin 1, 4, 11 and 14 of the chip. Make sure you have the chip oriented correctly, the notch indicates where pins 1 and 14 are

Solder the white wire (5V to the XBee) to pin 4. Solder the black wire (ground) to pin 11 directly across.

Now solder the yellow wire to pin 1 and the orange wire to pin 14.

Use two small pieces of sticky foam and stick them onto the back of the case.

Then place the XBee adapter and capacitor on the tape so that the LED sticks out of the hole drilled earlier

Tuck the excess ribbon cable out of the way so that they are not near the 120V connections which could make them go poof.

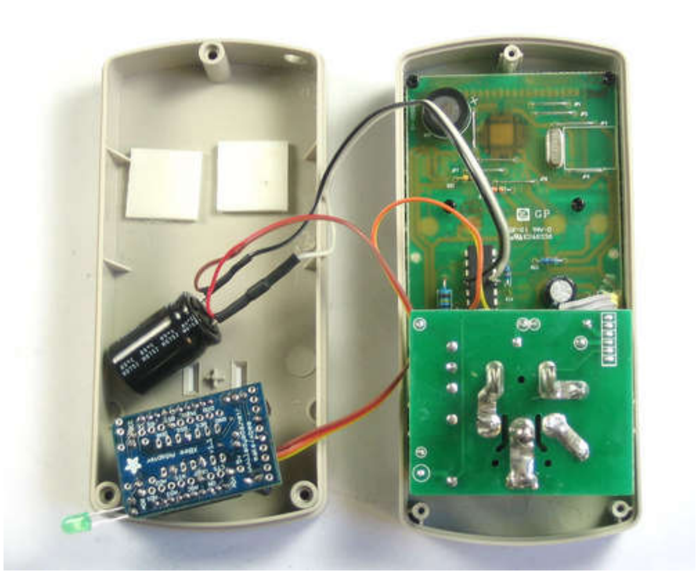
Close it up and plug it in.

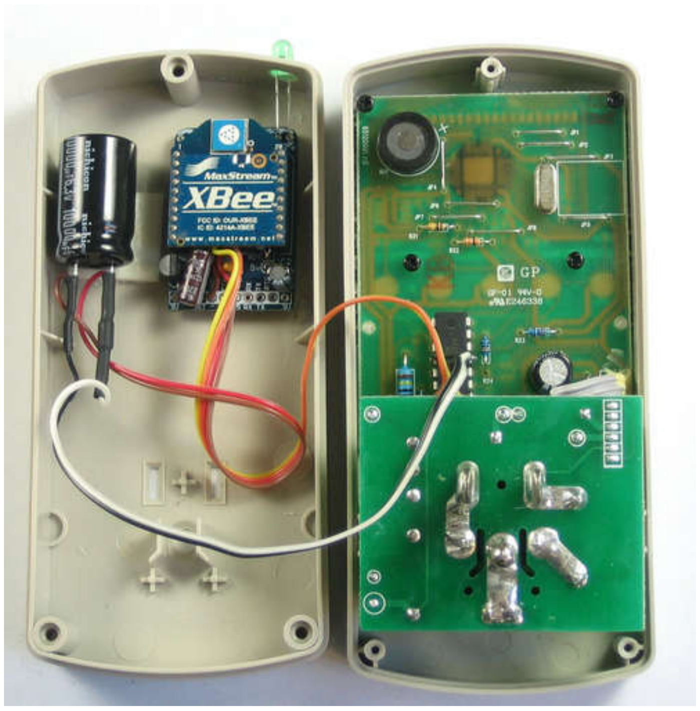
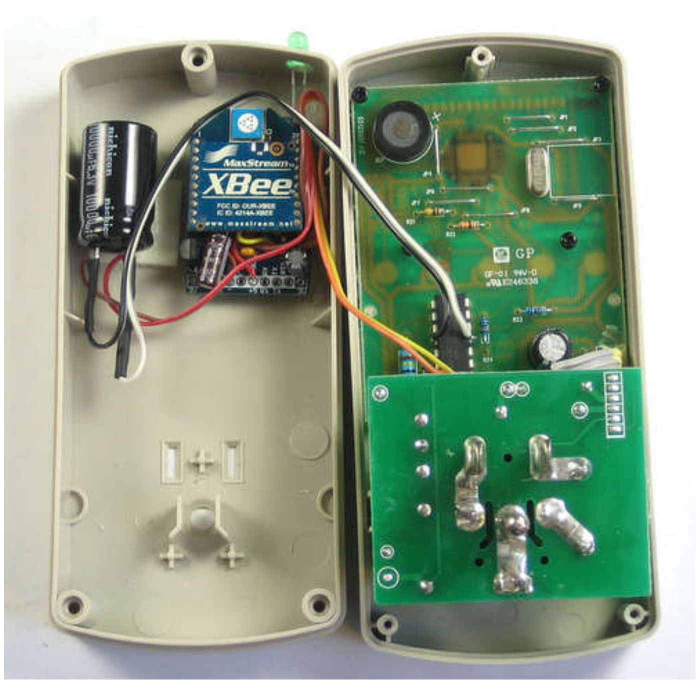
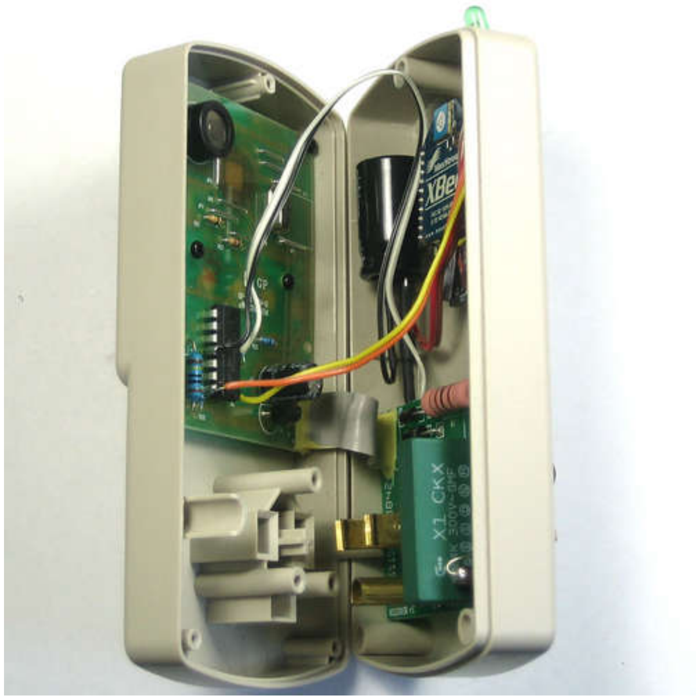
You'll notice its a bit finicky for a few seconds as the big capacitor charges up. The display may not come up for 15-30 seconds, and it may fade in and out at first. The numbers may also be wrong for a bit as it powers up. Within about 30 seconds, you should see the display stabilize and the indicator LED blinking every 2 seconds!

Go back to your computer, plug the receiver XBee into the USB adapter and make sure it has the latest firmware uploaded and set it to the same PAN ID as the transmitters. You will see the RSSI LED (red LED) light up. That means you have a good link!

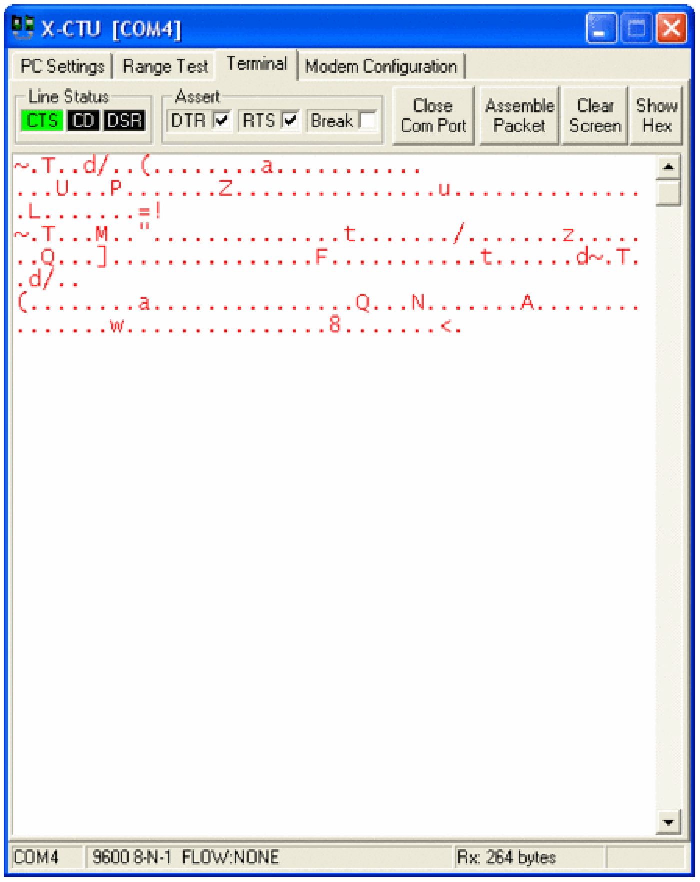
Open up the Terminal in X-CTU (or another terminal program) and connect at 9600 baud 8N1 parity and you'll see a lot of nonsense. Whats important is that a new chunk of nonsense gets printed out once every 2 seconds, indicating a packet of data has been received.

The hardware is done. Good work!



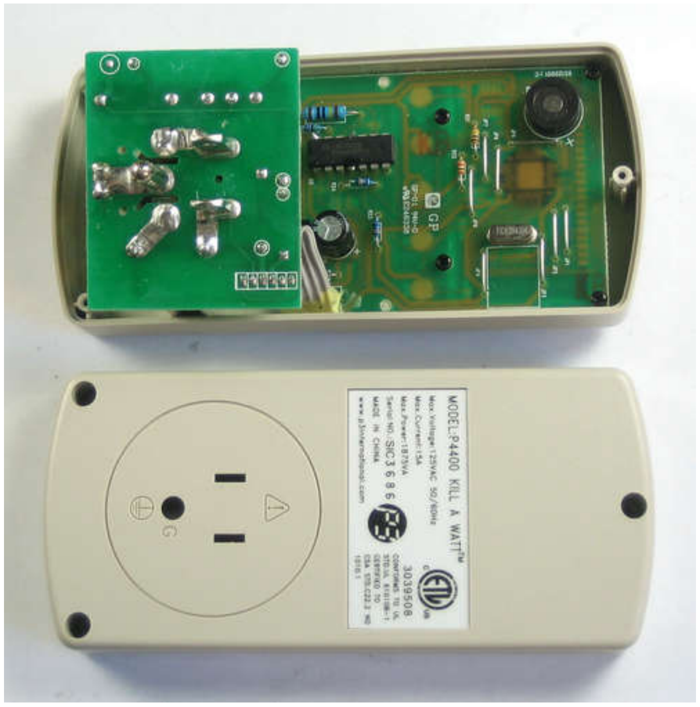

MaxStream
XBee

MaxStream
XBee


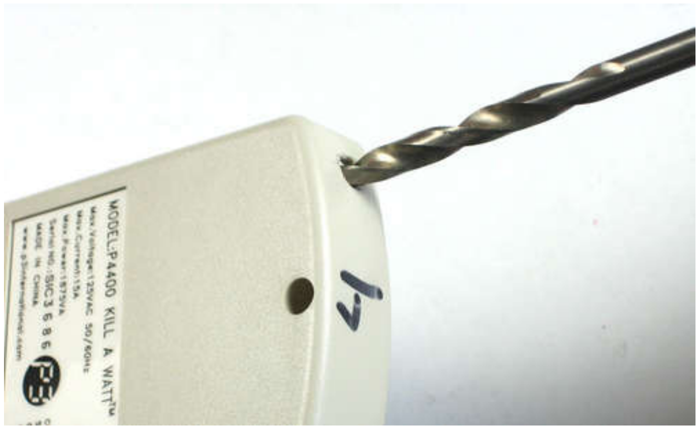
MODEL:P4400 KILL A WATT™

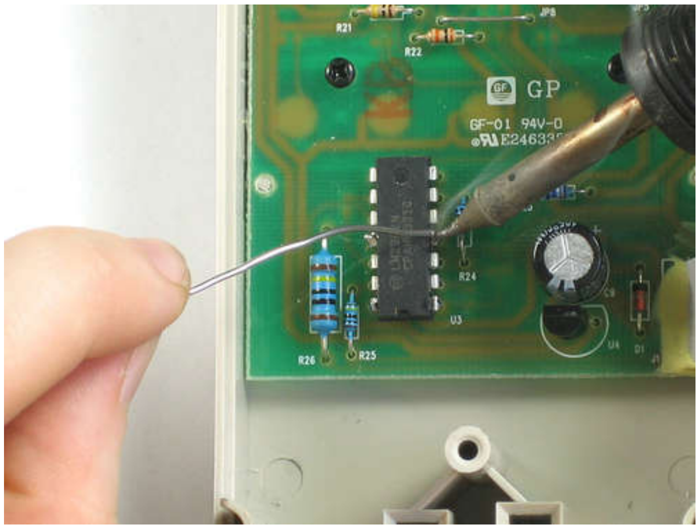
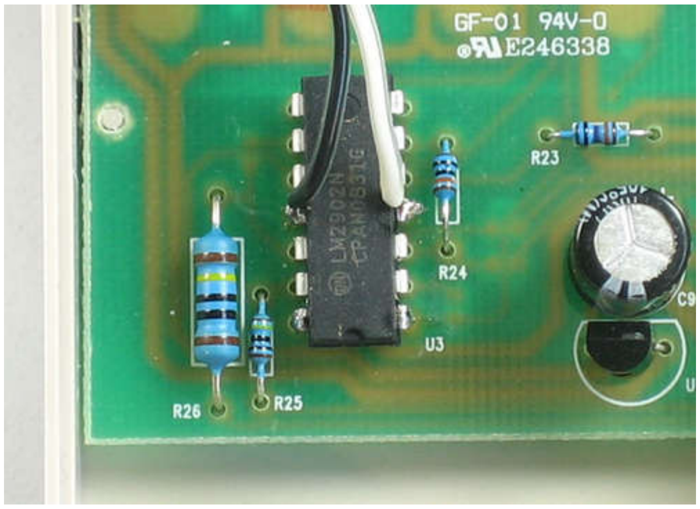
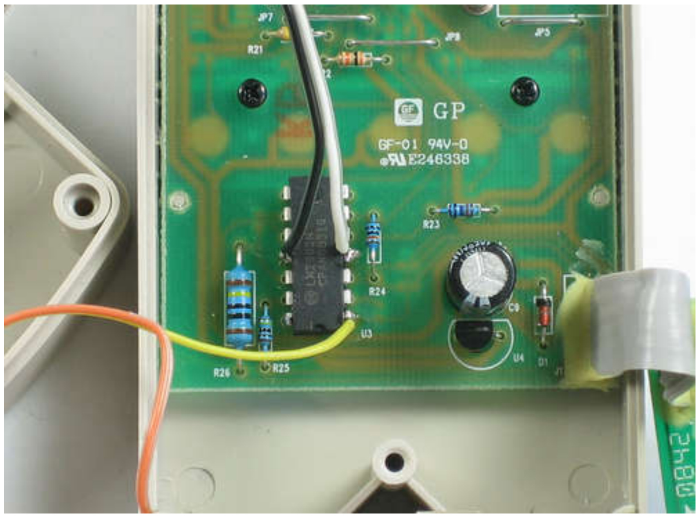
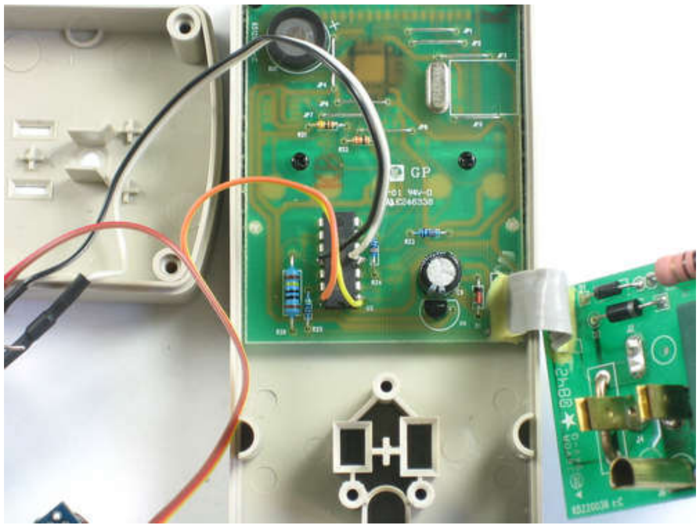
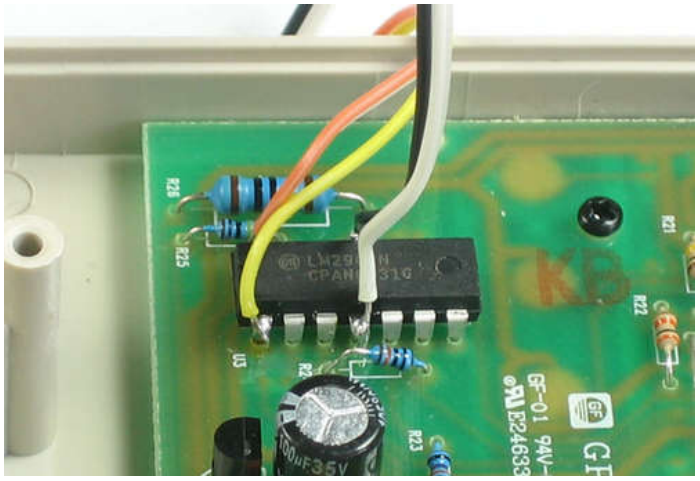


```
~.T..d/..(........a............
...U...P.......Z...............u..............
.L.......=!
~.T...M.."...............t......./.......z.....
..Q...]...............F...........t......d~.T.
.d/..
(.......a...............Q...N.......A.........
.......w...............8.......<.
```

COM4 | 9600 8-N-1 FLOW:NONE | Rx: 264 bytes





Step 12: Software

Introduction

Now that the hardware is complete, we come to the exciting part: running the software that retrieves the data from our receiver XBee and saves it to our computer or uploads it to a database or updates our twitter feed or....whatever you'd like!

Here is how it works, the XBee inside the Kill-a-Watt is hooked up to two analog signals. One is the voltage signal which indicates the AC voltage read. In general this is a sine wave that is 120VAC. One tricky thing to remember is that 120V is the 'RMS' voltage, and the 'true voltage' is +-170VDC. (You can read more about RMS voltage at wikipedia basically its a way to indicate how much 'average' voltage there is.) The second reading is the AC current read. This is how much current is being drawn through the Kill-a-Watt. If you multiply the current by the voltage, you'll get the power (in Watts) used!

The XBee's Analog/Digital converter is set up to take a 'snapshot' of one sine-cycle at a time. Each double-sample (voltage and current) is taken 1ms apart and it takes 17 of them. That translates to a 17ms long train of samples. One cycle of power-usage is 1/60Hz long which is 16.6ms. So it works pretty well!

Lets look at some examples of voltage and current waveforms as the XBee sees them.

For example this first graph is of a laptop plugged in. You'll see that its a switching supply, and only pulls power during the peak of the voltage curve.

Now lets try plugging in a 40W incandescent light bulb. You'll notice that unlike the switching supply, the current follows the voltage almost perfectly. That's because a lightbulb is just a resistor!

Finally, lets try sticking the meter on a dimmable switch. You'll see that the voltage is 'chopped' up, no longer sinusoidal. And although the current follows the voltage, its still matching pretty well.

The XBee sends the raw data to the computer which, in a python script, figures out what the (calibrated) voltage and amperage is at each sample and multiplies each point together to get the Watts used in that cycle. Since there's almost no device that changes the power-usages from cycle-to-cycle, the snapshot is a good indicator of the overall power usage that second. Then once every 2 seconds, a single snapshot is sent to the receiver XBee

Install python & friends

The software that talks to the XBee is written in **python** . I used python because its quick to develop in, has multi-OS support and is pretty popular with software and hardware hackers. The XBees talk over the serial port so literally any programming language can/could be used here. If you're a software geek and want to use perl, C, C#, tcl/tk, processing, java, etc. go for it! You'll have to read the serial data and parse out the packet but its not particularly hard.

However, most people just want to get on with it and so for you we'll go through the process of installing **python** and the libraries we need.

1. Download and install python 2.5 from http://www.python.org/download/ I suggest 2.5 because that seems to be stable and well supported at this time. If you use another version there may be issues
2. Download and install pyserial from the package repository (this will let us talk to the XBee thru the serial port)
3. If you're running windows download and install win32file for python 2.5 (this will add file support)
4. Download and install the simplejson python library (this is how the twitter api likes to be spoken to)

Now you can finally download the Wattcher script we will demonstrate here! We're going to download it into the **C:\wattcher** directory, for other OS's you can of course change this directory

Basic configure

We'll have to do a little bit of setup to start, open up the **wattcher.py** script with a text editor and find the line

SERIALPORT = "COM4" # the com/serial port the XBee is connected to

change COM4 into whatever the serial port you will be connecting to the XBee with is called. Under windows its some **COMx** port, under linux and mac its something like **/dev/cu.usbserial-xxxx** check the /dev/ directory and/or **dmesg**

Save the script with the new serial port name

Test it out

Once you have installed python and extracted the scripts to your working directory, start up a terminal (under linux this is just **rxvt** or **xterm** , under mac its **Terminal** , under windows, its a **cmd** window)

I'm going to assume you're running windows from now on, it shouldn't be tough to adapt the instructions to linux/mac once the terminal window is open.

Run the command **cd C:\wattcher** to get to the place where you uncompressed the files. By running the **dir** command you can see that you have the files in the directory

Make sure your transmitter (Kill-a-Watt + Xbee) is plugged in, and blinking once every 2 seconds. Remember it takes a while for the transmitter to charge up power and start transmitting. The LCD display should be clear, not fuzzy. Make sure that there's nothing plugged into the Kill-a-Watt, too. The RSSI (red) LED on the receiver connected to the computer should be lit indicating data is being received. Don't continue until that is all good to go.

Now run **python** by running the command **C:\python25\python.exe wattcher.py**

You should get a steady print out of data. The first number is the XBee address from which it received data, following is the estimated current draw, wattage used and the Watt-hours consumed since the last data came in. Hooray! We have wireless data!

Calibrating

Now that we have good data being received, its time to tweak it. For example, its very likely that even without an appliance or light plugged into the Kill-a-Watt, the script thinks that there is power being used. We need to calibrate the sensor so that we know where 'zero' is. In the Kill-a-Watt there is an autocalibration system but unfortunately the XBee is not smart enough to do it on its own. So, we do it in the python script. Quit the script by typing in Control-C and run it again this time as **C:\python25\python.exe wattcher.py -d** note the **-d** which tells the script to print out debugging information

Now you can see the script printing out a whole mess of data. The first chunk with lots of -1's in it is the raw packet. While its interesting we want to look at the line that starts with **ampdata** :

```
ampdata: [498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 497, 498, 498, 498]
```

Now you'll notice that the numbers are pretty much all the same. That's because there's nothing plugged into the tweetawatt and so each 1/60 Hz cycle has a flat line at 'zero'. The A/D in the XBee is 10 bits, and will return values between 0 and 1023. So, in theory, if the system is perfect the value at 'zero' should be 512. However, there are a bunch of little things that make the system imperfect and so zero is only close to 512. In this case the 'zero' calibration point is really 498. When its off there is a 'DC offset' to the Amp readings, as this graph shows:

See how the Amp line (green) is steady but its not at zero, its at 0.4 amps? There is a 'DC offset' of 0.4 amps

OK, open up the wattcher.py script in a text editor.

vrefcalibration = [492, # Calibration for sensor #0]
492, # Calibration for sensor #1
489, # Calibration for sensor #2
492, # Calibration for sensor #3
501, # Calibration for sensor #4
493] # etc... approx ((2.4v * (10Ko/14.7Ko)) / 3

See the line that says **# Calibration for sensor #1** ? Change that to **498**

vrefcalibration = [492, # Calibration for sensor #0]
498, # Calibration for sensor #1
489, # Calibration for sensor #2
492, # Calibration for sensor #3
501, # Calibration for sensor #4
493] # etc... approx ((2.4v * (10Ko/14.7Ko)) / 3

Save the file and start up the script again, this time without the **-d**

Now you'll see that the Watt draw is 2W or less, instead of 40W (which was way off!) The reason its not 0W is that, first off, there's a little noise that we're reading in the A/D lines, secondly there's power draw by the Kill-a-Watt itself and finally, the XBee doesn't have a lot of samples to work with. However <2W is pretty good considering that the full sensing range is 0-1500W

Note the graph with the calibrated sensor:

See how the Amps line is now at 0 steady, there is no DC offset

Logging data

Its nice to have this data but it would be even nicer if we could store it for use. Well, thats automatically done for you! You can set the name of the log file in the wattcher.py script. By default it's **powerdatalog.csv** . The script collects data and every 5 minutes writes a single line in the format **Year Month Day, Time, Sensor#, Watts** for each sensor.As you can see, this is an example of a 40W incandescent lightbulb plugged in for a few hours. Because of the low sample rate, you'll see some minor variations in the Watts recorded. This data can be easily imported directly into any spreadsheet program

Tweeting

Finally we get to the tweeting part of the tweet-a-watt. First open up the wattcher.py script and set

# Twitter username & password
twitterusername = "username"
twitterpassword = "password"

to your username and password on twitter. You can make an account on twitter.com if you don't have one.

Then run the script as usual. Every 8 hours (midnight, 8am and 4pm) the script will sent a twitter using the Twitter API

Then check it out at your account:

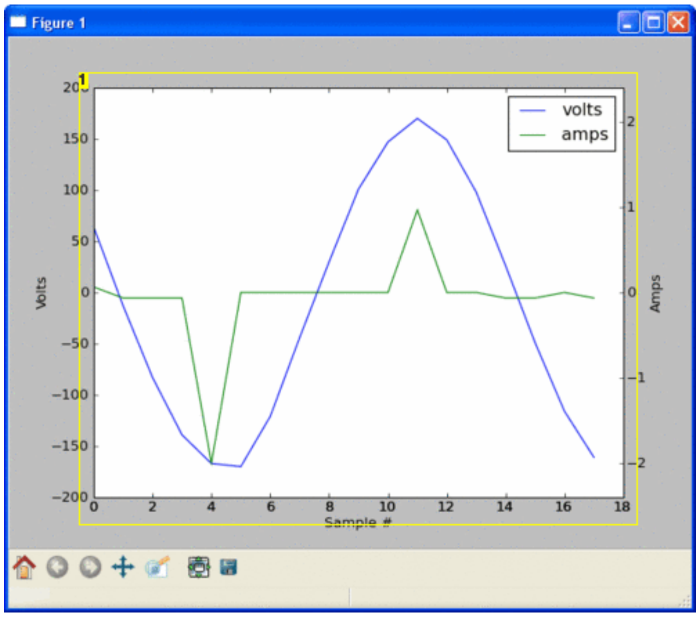


Image Notes
1. A laptop plugged in, switching power supply

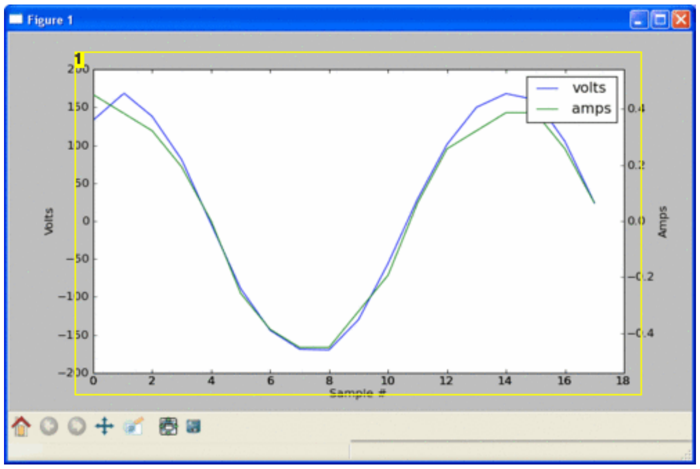
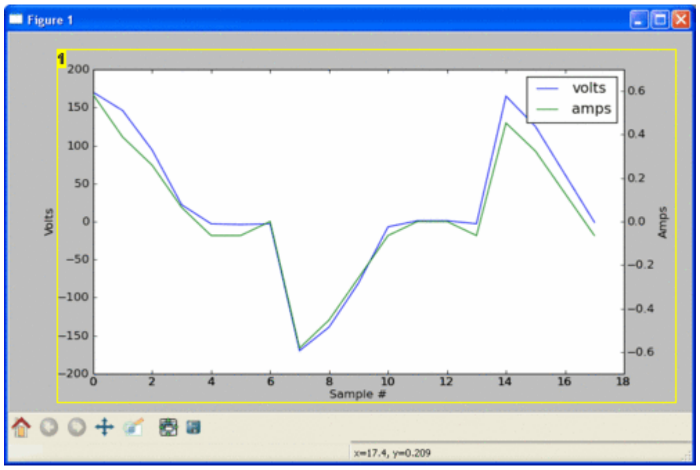
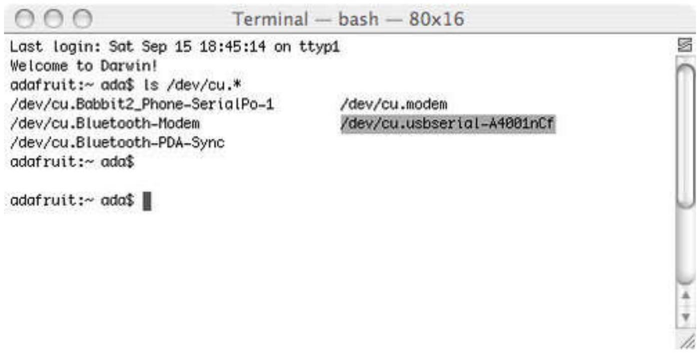


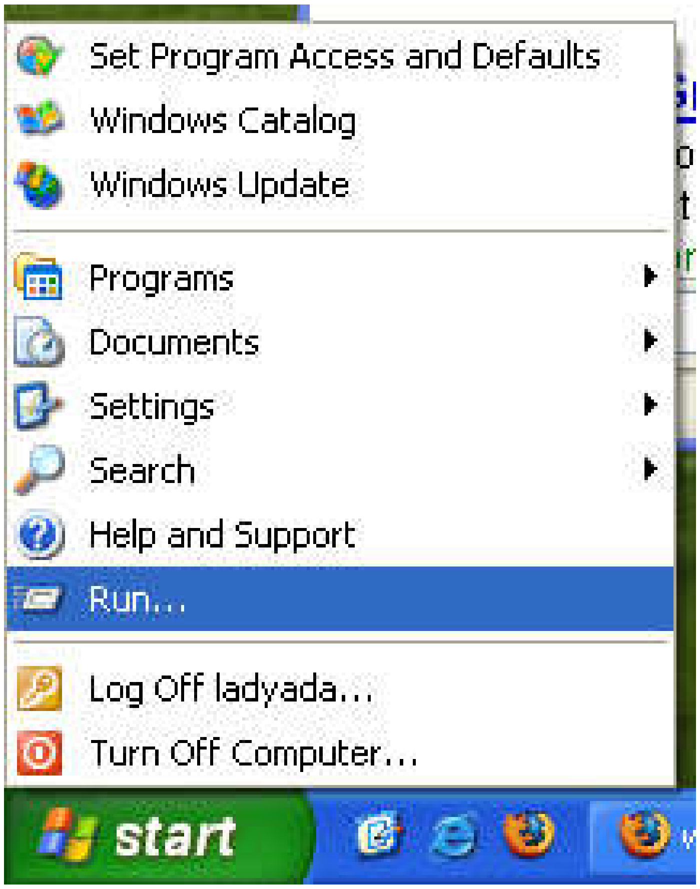
Image Notes
1. 40W lightbulb

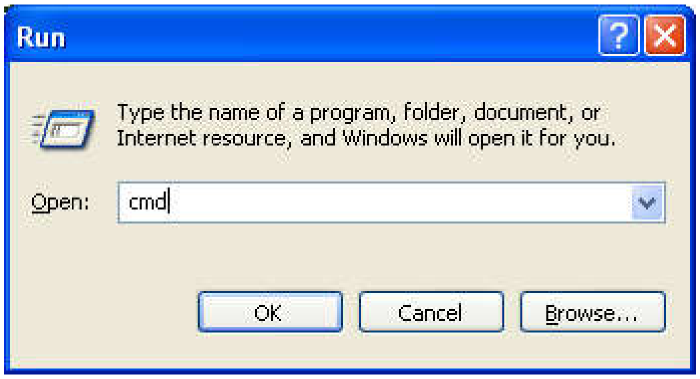


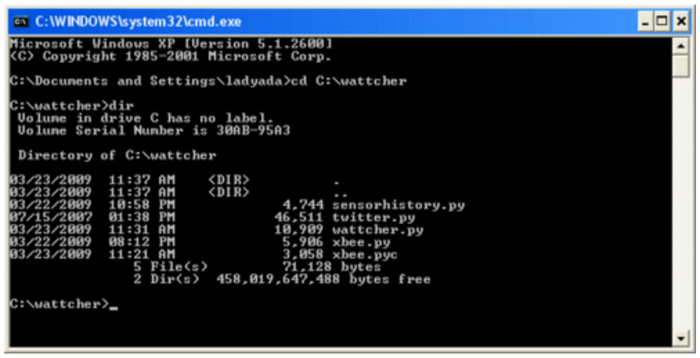


```
Terminal — bash — 80x16
Last login: Sat Sep 15 18:45:14 on ttyp1
Welcome to Darwin!
adafruit:~ ada$ ls /dev/cu.*
/dev/cu.Bobbit2_Phone-SerialPo-1        /dev/cu.modem
/dev/cu.Bluetooth-Modem                 /dev/cu.usbserial-A4001nCf
/dev/cu.Bluetooth-PDA-Sync
adafruit:~ ada$

adafruit:~ ada$
```

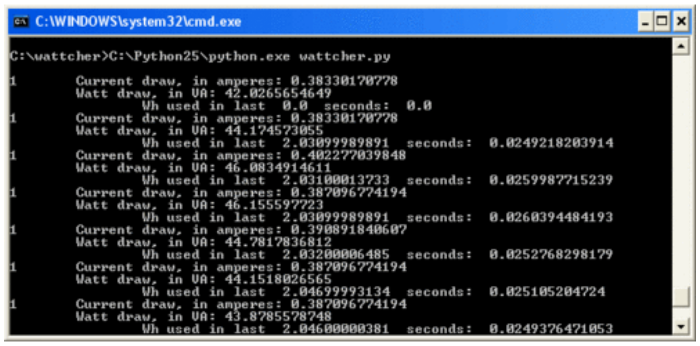
Image Notes
1. Light bulb on dimmer switch



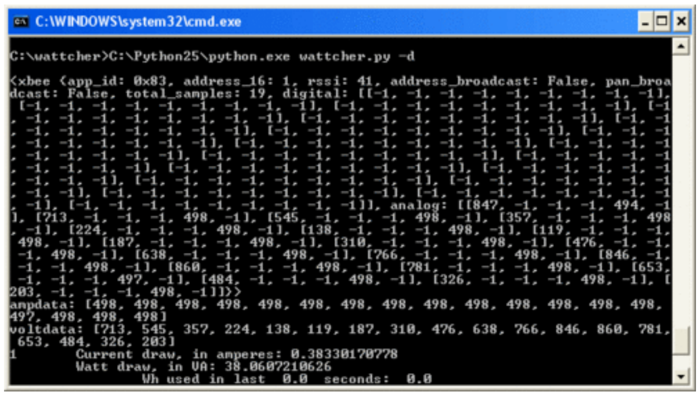

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
cmd
OK
Cancel
Browse...



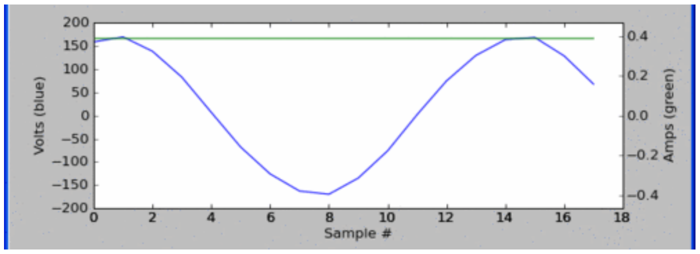
```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ladyada>cd C:\wattcher

C:\wattcher>dir
 Volume in drive C has no label.
 Volume Serial Number is 30AB-95A3

 Directory of C:\wattcher

03/23/2009  11:37 AM    <DIR>          .
03/23/2009  11:37 AM    <DIR>          ..
03/22/2009  10:58 PM             4,744 sensorhistory.py
07/15/2007  01:38 PM            46,511 twitter.py
03/23/2009  11:31 AM            10,909 wattcher.py
03/22/2009  08:12 PM             5,906 xbee.py
03/23/2009  11:21 AM             3,058 xbee.pyc
               5 File(s)         71,128 bytes
               2 Dir(s)  458,019,647,488 bytes free

C:\wattcher>_
```

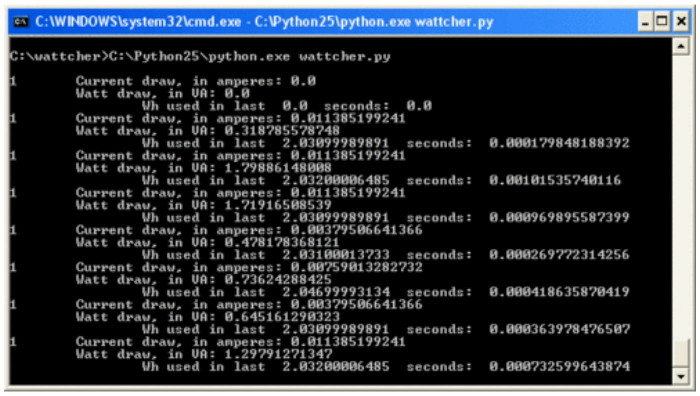


```
C:\WINDOWS\system32\cmd.exe
C:\wattcher>C:\Python25\python.exe wattcher.py

1       Current draw, in amperes: 0.38330170778
        Watt draw, in VA: 42.0265654649
                Wh used in last  0.0  seconds:  0.0
1       Current draw, in amperes: 0.38330170778
        Watt draw, in VA: 44.174573055
                Wh used in last  2.03099989891  seconds:  0.0249218203914
1       Current draw, in amperes: 0.402277039848
        Watt draw, in VA: 46.0834914611
                Wh used in last  2.03100013733  seconds:  0.0259987715239
1       Current draw, in amperes: 0.387096774194
        Watt draw, in VA: 46.155597723
                Wh used in last  2.03099989891  seconds:  0.0260394484193
1       Current draw, in amperes: 0.390891840607
        Watt draw, in VA: 44.7817836812
                Wh used in last  2.03200006485  seconds:  0.0252768298179
1       Current draw, in amperes: 0.387096774194
        Watt draw, in VA: 44.1518026565
                Wh used in last  2.04699993134  seconds:  0.025105204724
1       Current draw, in amperes: 0.387096774194
        Watt draw, in VA: 43.8785578748
                Wh used in last  2.04600000381  seconds:  0.0249376471053
```

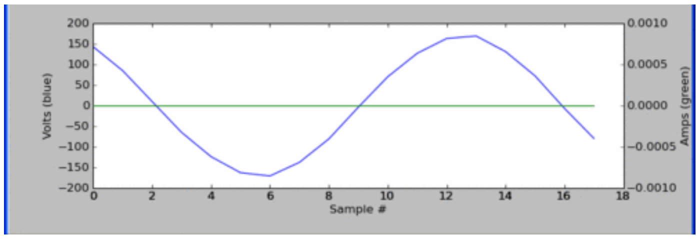


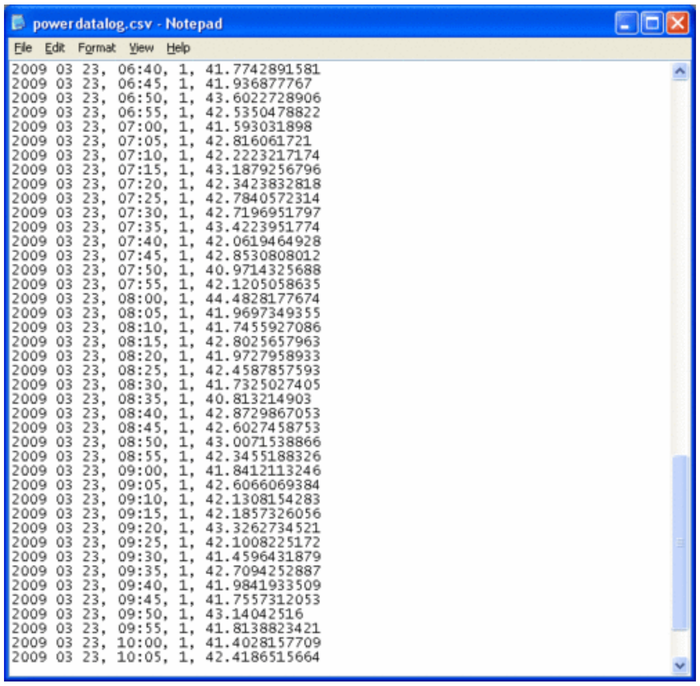
```
C:\wattcher>C:\Python25\python.exe wattcher.py -d

<xbee {app_id: 0x83, address_16: 1, rssi: 41, address_broadcast: False, pan_broa
dcast: False, total_samples: 19, digital: [[-1, -1, -1, -1, -1, -1, -1, -1, -1],
 [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1
, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1
, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1
, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1
, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1
, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1
, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1
, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1]], analog: [[847, -1, -1, -1, 494, -1
], [713, -1, -1, -1, 498, -1], [545, -1, -1, -1, 498, -1], [357, -1, -1, -1, 498
, -1], [224, -1, -1, -1, 498, -1], [138, -1, -1, -1, 498, -1], [119, -1, -1, -1,
 498, -1], [187, -1, -1, -1, 498, -1], [310, -1, -1, -1, 498, -1], [476, -1, -1,
 -1, 498, -1], [638, -1, -1, -1, 498, -1], [766, -1, -1, -1, 498, -1], [846, -1,
 -1, -1, 498, -1], [860, -1, -1, -1, 498, -1], [781, -1, -1, -1, 498, -1], [653,
 -1, -1, -1, 497, -1], [484, -1, -1, -1, 498, -1], [326, -1, -1, -1, 498, -1], [
203, -1, -1, -1, 498, -1]]}>
ampdata: [498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498,
497, 498, 498, 498]
voltdata: [713, 545, 357, 224, 138, 119, 187, 310, 476, 638, 766, 846, 860, 781,
 653, 484, 326, 203]
1        Current draw, in amperes: 0.38330170778
         Watt draw, in VA: 38.0607210626
                  Wh used in last  0.0  seconds:  0.0
```



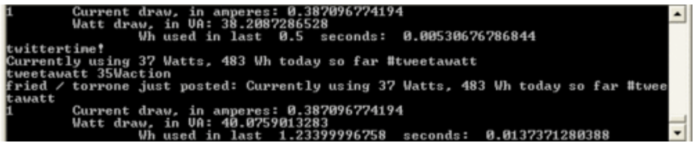

C:\WINDOWS\system32\cmd.exe - C:\Python25\python.exe wattcher.py

```
C:\wattcher>C:\Python25\python.exe wattcher.py

1        Current draw, in amperes: 0.0
         Watt draw, in VA: 0.0
                  Wh used in last  0.0  seconds:  0.0
1        Current draw, in amperes: 0.011385199241
         Watt draw, in VA: 0.3187855578748
                  Wh used in last  2.03099989891  seconds:  0.000179848188392
1        Current draw, in amperes: 0.011385199241
         Watt draw, in VA: 1.79886148008
                  Wh used in last  2.03200006485  seconds:  0.00101535740116
1        Current draw, in amperes: 0.011385199241
         Watt draw, in VA: 1.71916508539
                  Wh used in last  2.03099989891  seconds:  0.000969895587399
1        Current draw, in amperes: 0.00379506641366
         Watt draw, in VA: 0.478178368121
                  Wh used in last  2.03100013733  seconds:  0.000269772314256
1        Current draw, in amperes: 0.00759013282732
         Watt draw, in VA: 0.73624288425
                  Wh used in last  2.04699993134  seconds:  0.000418635870419
1        Current draw, in amperes: 0.00379506641366
         Watt draw, in VA: 0.645161290323
                  Wh used in last  2.03099989891  seconds:  0.000363978476507
1        Current draw, in amperes: 0.011385199241
         Watt draw, in VA: 1.29791271347
                  Wh used in last  2.03200006485  seconds:  0.000732599643874
```

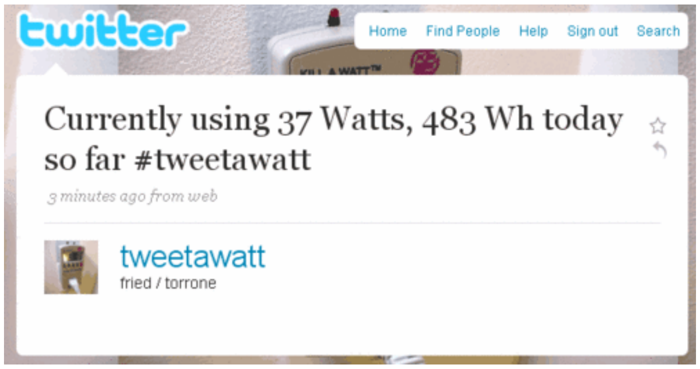




```
2009 03 23, 06:40, 1, 41.7742891581
2009 03 23, 06:45, 1, 41.936877767
2009 03 23, 06:50, 1, 43.6022728906
2009 03 23, 06:55, 1, 42.5350478822
2009 03 23, 07:00, 1, 41.593031898
2009 03 23, 07:05, 1, 42.816061721
2009 03 23, 07:10, 1, 42.2223217174
2009 03 23, 07:15, 1, 43.1879256796
2009 03 23, 07:20, 1, 42.3423832818
2009 03 23, 07:25, 1, 42.7840572314
2009 03 23, 07:30, 1, 42.7196951797
2009 03 23, 07:35, 1, 43.4223951774
2009 03 23, 07:40, 1, 42.0619464928
2009 03 23, 07:45, 1, 42.8530808012
2009 03 23, 07:50, 1, 40.9714325688
2009 03 23, 07:55, 1, 42.1205058635
2009 03 23, 08:00, 1, 44.4828177674
2009 03 23, 08:05, 1, 41.9697349355
2009 03 23, 08:10, 1, 41.7455927086
2009 03 23, 08:15, 1, 42.8025657963
2009 03 23, 08:20, 1, 41.9727958933
2009 03 23, 08:25, 1, 42.4587857593
2009 03 23, 08:30, 1, 41.7325027405
2009 03 23, 08:35, 1, 40.813214903
2009 03 23, 08:40, 1, 42.8729867053
2009 03 23, 08:45, 1, 42.6027458753
2009 03 23, 08:50, 1, 43.0071538866
2009 03 23, 08:55, 1, 42.3455188326
2009 03 23, 09:00, 1, 41.8412113246
2009 03 23, 09:05, 1, 42.6066069384
2009 03 23, 09:10, 1, 42.1308154283
2009 03 23, 09:15, 1, 42.1857326056
2009 03 23, 09:20, 1, 43.3262734521
2009 03 23, 09:25, 1, 42.1008225172
2009 03 23, 09:30, 1, 41.4596431879
2009 03 23, 09:35, 1, 42.7094252887
2009 03 23, 09:40, 1, 41.9841933509
2009 03 23, 09:45, 1, 41.7557312053
2009 03 23, 09:50, 1, 43.14042516
2009 03 23, 09:55, 1, 41.8138823421
2009 03 23, 10:00, 1, 41.4028157709
2009 03 23, 10:05, 1, 42.4186515664
```



```
1        Current draw, in amperes: 0.387096774194
         Watt draw, in VA: 38.2087286528
                  Wh used in last  0.5  seconds:  0.00530676786844
twittertime!
Currently using 37 Watts, 483 Wh today so far #tweetawatt
tweetawatt 35Waction
fried / torrone just posted: Currently using 37 Watts, 483 Wh today so far #twee
tawatt
1        Current draw, in amperes: 0.387096774194
         Watt draw, in VA: 40.0759013283
                  Wh used in last  1.23399996758  seconds:  0.0137371280388
```


Currently using 37 Watts, 483 Wh today so far #tweetawatt

3 minutes ago from web

tweetawatt

fried / torrone

Step 13: Expand

Overview

Once you've got your base system up and running here are some ideas for how to extend, improve or expand it!

Add more outlets

So you can track more rooms, of course

Graphing

If you'd like to play some more with the script, there's some extras built in. For example, you can graph the data as it comes in from the XBee, both Watts used and the actual 'power line' waveform. Simply set **GRAPHIT = True** you'll need to install a mess of python libraries though, including **wxpython** , **numpy** and **pylab**

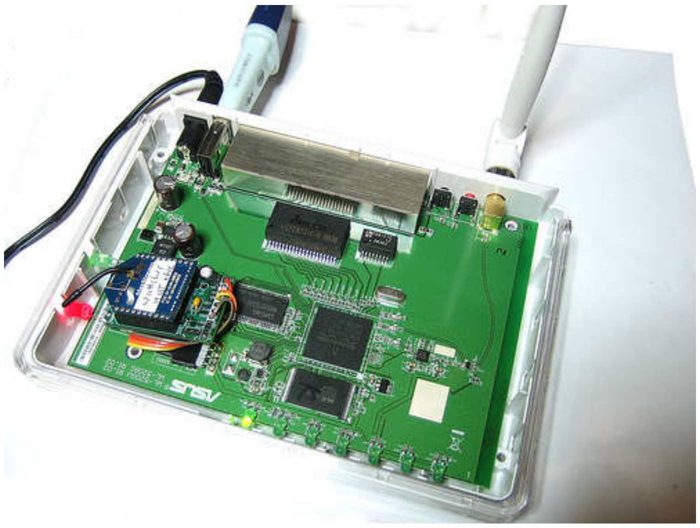
Remove the computer

It took a few hours, but I hacked my Asus wifi router to also log data for me. There'll be more documentation soon but here's some hints:

Do basically everything in [Do basically everything in MightyOhm's tutorial. You can use the FTDI cable to reprogram the router, just move the pins around. Then add a 16mb USB key (I was given one as schwag so look in your drawers) and install python and the openssl library as well as the other libraries needed like pyserial. The code should pretty much just run! (I'll put up more detailed notes later)]

The router still works as my wifi gateway to the cablemodem, and only uses 5W MightyOhm's tutorial]. You can use the FTDI cable to reprogram the router, just move the pins around. Then add a 16mb USB key (I was given one as schwag so look in your drawers) and install python and the openssl library as well as the other libraries needed like pyserial. The code should pretty much just run! (I'll put up more detailed notes later)

The router still works as my wifi gateway to the cablemodem, and only uses 5W



Step 14: Design - overview

Design overview

For those interested in how to build a sensor node system with a Google Appengine backend, here is the process by which I created it. Of course, you should have the hardware part done first!

1. Listen - designing the parser for the computer that grabs XBee packets, and extracts the useful data
2. Store - how to use GAE to store the data in 'the cloud'
3. Graph - using Google Visualizations to make pretty graphs

Step 15: Design - listen

Data listening & parsing

In this section we will work on the receiver software, that will talk to a receiver XBee and figure out what the sensor data means. I'll be writing the code in **python** which is a fairly-easy to use scripting language. It runs on all OS's and has tons of tutorials online. Also, Google AppEngine uses it so its a good time to learn!

This whole section assumes that you only have 1 transmitter and 1 receiver, mostly to make graphing easier to cope with. In the next section we'll tie in more sensors when we get to the datalogging part!

Raw analog input

We'll start by just getting raw data from the XBee and checking it out. The packet format for XBees is published but instead of rooting around in it, I'll just use the handy XBee library written for python. With it, I can focus on the data instead of counting bytes and calculating checksums.

To use the library, first use the **pyserial** module to open up a serial port (ie COM4 under windows, /dev/ttyUSB0 under mac/linux/etc) You can look at the XBee project page for information on how to figure out which COM port you're looking for. We connect at the standard default baudrate for XBees, which is 9600 and look for packets

```
from xbee import xbee
import serial

SERIALPORT = "COM4" # the com/serial port the XBee is connected to
BAUDRATE = 9600 # the baud rate we talk to the xbee

# open up the FTDI serial port to get data transmitted to xbee
ser = serial.Serial(SERIALPORT, BAUDRATE)
ser.open()

while True:
# grab one packet from the xbee, or timeout
packet = xbee.find_packet(ser)
if packet:
xb = xbee(packet)

print xb
```

Running this code, you'll get the following output:

```
<xbee {app_id: 0x83, address_16: 1, rssi: 85, address_broadcast: False, pan_broadcast: False, total_samples: 19, digital: [[-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1,
-1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1 , -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1 , -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -
1, -1, -1], [-1, -1, -1 , -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1 , -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1 , -1, -1, -1, -1], [-1,
-1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1 , -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1 , -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -
1, -1, -1 , -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1]], analog: [[190, -1, -1, -1, 489, -1], [109, -1, -1, -1, 484, -1], [150, -1, -1, -1, 492, -1], [262, -1, -1, -1, 492 , -1], [423, -1, -1, -1,
492, -1], [589, -1, -1, -1, 492, -1], [740, -1, -1, -1, 492, -1], [843, -1, -1, -1, 492, -1], [870, -1, -1, -1, 496, -1], [805, -1, -1, -1, 491, -1], [680, -1, -1, -1, 492, -1], [518, -1, -1,
-1, 492, -1], [349, -1, -1, -1, 491, -1], [199, -1, -1, -1, 491, -1], [116, -1, -1, -1, 468, -1], [108, -1, -1, -1, 492, -1], [198, -1, -1, -1, 492, -1], [335, -1, -1, -1, 492, -1], [523, -1,
-1, -1, 492, -1]]}>
```

which we will reformat to make a little more legible

```
<xbee {
app_id: 0x83,
address_16: 1,
rssi: 85,
address_broadcast: False,
pan_broadcast: False,
total_samples: 19,
digital: [[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1]],
analog: [[190, -1, -1, -1, 489, -1],
[109, -1, -1, -1, 484, -1],
[150, -1, -1, -1, 492, -1],
[262, -1, -1, -1, 492, -1],
[423, -1, -1, -1, 492, -1],
[589, -1, -1, -1, 492, -1],
[740, -1, -1, -1, 492, -1],
[843, -1, -1, -1, 492, -1],
[870, -1, -1, -1, 496, -1],
[805, -1, -1, -1, 491, -1],
[680, -1, -1, -1, 492, -1],
[518, -1, -1, -1, 492, -1],
[349, -1, -1, -1, 491, -1],
[199, -1, -1, -1, 491, -1],
[116, -1, -1, -1, 468, -1],
[108, -1, -1, -1, 492, -1],
[198, -1, -1, -1, 492, -1],
[335, -1, -1, -1, 492, -1],
[523, -1, -1, -1, 492, -1]]

}>
```

OK now its clear whats going on here. First off, we get some data like the transmitter ID (address_16) and signal strength (RSSI). The packet also tells us how many sample are available (19). Now, the digital samples are all -1 because we didn't request any to be sent. The library still fills them in tho so thats why the non-data is there. The second chunk is 19 sets of analog data, ranging from 0 to 1023. As you can see, the first sample (#0) and fifth sample (#4) contain real data, the rest are -1. That corresponds to the hardware section where we setup AD0 and AD4 to be our voltage and current sensors.

We'll tweak our code so that we can extract this data only and ignore the rest of the packet.

This code creates two arrays, voltagedata and ampdata where we will stick the data. We throw out the first sample because usually ADCs are a bit wonky on the first sample and then are good to go after that. It may not be necessary tho

```
#!/usr/bin/env python
import serial
from xbee import xbee

SERIALPORT = "COM4" # the com/serial port the XBee is connected to
BAUDRATE = 9600 # the baud rate we talk to the xbee
CURRENTSENSE = 4 # which XBee ADC has current draw data
VOLTSENSE = 0 # which XBee ADC has mains voltage data

# open up the FTDI serial port to get data transmitted to xbee
ser = serial.Serial(SERIALPORT, BAUDRATE)
ser.open()

while True:
# grab one packet from the xbee, or timeout
packet = xbee.find_packet(ser)
if packet:
xb = xbee(packet)

#print xb
# we'll only store n-1 samples since the first one is usually messed up
voltagedata = [-1] * (len(xb.analog_samples) - 1)
ampdata = [-1] * (len(xb.analog_samples ) -1)
# grab 1 thru n of the ADC readings, referencing the ADC constants
# and store them in nice little arrays
for i in range(len(voltagedata)):
voltagedata[i] = xb.analog_samples[i+1][VOLTSENSE]
ampdata[i] = xb.analog_samples[i+1][CURRENTSENSE]
print voltagedata
print ampdata
```

Now our data is easier to see:

```
Voltage: [672, 801, 864, 860, 755, 607, 419, 242, 143, 108, 143, 253, 433, 623, 760, 848, 871, 811]
Current: [492, 492, 510, 491, 492, 491, 491, 491, 492, 480, 492, 492, 492, 492, 492, 492, 497, 492]
```

Note that the voltage swings from about 100 to 900, sinusoidally.

Normalizing the data

Next up we will 'normalize' the data. The voltage should go from -170 to +170 which is the actual voltage on the line, instead of 100 to 900 which is just what the ADC reads. To do that we will get the average value of the largest and smallest reading and subtract it from all the samples. After that, we'll normalize the Current measurements as well, to get the numbers to equal the current draw in Amperes.

```
#!/usr/bin/env python
import serial
from xbee import xbee

SERIALPORT = "COM4" # the com/serial port the XBee is connected to
BAUDRATE = 9600 # the baud rate we talk to the xbee
CURRENTSENSE = 4 # which XBee ADC has current draw data
VOLTSENSE = 0 # which XBee ADC has mains voltage data

# open up the FTDI serial port to get data transmitted to xbee
ser = serial.Serial(SERIALPORT, BAUDRATE)
ser.open()

while True:
# grab one packet from the xbee, or timeout
packet = xbee.find_packet(ser)
if packet:
xb = xbee(packet)
```

```
#print xb
# we'll only store n-1 samples since the first one is usually messed up
voltagedata = [-1] * (len(xb.analog_samples) - 1)
ampdata = [-1] * (len(xb.analog_samples ) -1)
# grab 1 thru n of the ADC readings, referencing the ADC constants
# and store them in nice little arrays
for i in range(len(voltagedata)):
voltagedata[i] = xb.analog_samples[i+1][VOLTSENSE]
ampdata[i] = xb.analog_samples[i+1][CURRENTSENSE]

# get max and min voltage and normalize the curve to '0'
# to make the graph 'AC coupled' / signed
min_v = 1024 # XBee ADC is 10 bits, so max value is 1023
max_v = 0
for i in range(len(voltagedata)):
if (min_v > voltagedata[i]):
min_v = voltagedata[i]
if (max_v < voltagedata[i]):
max_v = voltagedata[i]

# figure out the 'average' of the max and min readings
avgv = (max_v + min_v) / 2
# also calculate the peak to peak measurements
vpp = max_v-min_v

for i in range(len(voltagedata)):
#remove 'dc bias', which we call the average read
voltagedata[i] -= avgv
# We know that the mains voltage is 120Vrms = +-170Vpp
voltagedata[i] = (voltagedata[i] * MAINSVPP) / vpp

# normalize current readings to amperes
for i in range(len(ampdata)):
# VREF is the hardcoded 'DC bias' value, its
# about 492 but would be nice if we could somehow
# get this data once in a while maybe using xbeeAPI
ampdata[i] -= VREF
# the CURRENTNORM is our normalizing constant
# that converts the ADC reading to Amperes
ampdata[i] /= CURRENTNORM

print "Voltage, in volts: ", voltagedata
print "Current, in amps: ", ampdata
```

We'll run this now to get this data, which looks pretty good, there's the sinusoidal voltage we are expecting and the current is mostly at 0 and then peaks up and down once in a while. Note that the current is sometimes negative but that's OK because we multiply it by the voltage and if both are negative it still comes out as a positive power draw

```
Voltage, in volts: [-125, -164, -170, -128, -64, 11, 93, 148, 170, 161, 114, 46, -39, -115, -157, -170, -150, -99]

Current, in amps: [0.064516129032258063, -1.096774193548387, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.096774193548387,]
0.0, 0.0, 0.0, -0.064516129032258063, 0.0, 0.0, -0.70967741935483875, 0.0, 0.0]
```

Basic data graphing

Finally, I'm going to add a whole bunch more code that will use the **numpy** graphing modules to make a nice graph of our data. Note that you'll need to install **wxpython** as well as **numpy** , and **matplotlib** !

At this point, the code is getting waaay to big to paste here so grab "wattcher.py Mains graph" from the download page!

Run it and you should see a graph window pop up with a nice sinusoidal voltage graph and various amperage data. For example this first graph is of a laptop plugged in. You'll see that its a switching supply, and only pulls power during the peak of the voltage curve.

Now lets try plugging in a 40W incandescent light bulb. You'll notice that unlike the switching supply, the current follows the voltage almost perfectly. Thats because a lightbulb is just a resistor!

Finally, lets try sticking the meter on a dimmable switch. You'll see that the voltage is 'chopped' up, no longer sinusoidal. And although the current follows the voltage, its still matching pretty well.

Graphing wattage!

OK neat, its all fun to watch waveforms but what we -really want- is the Watts used. Remember, P = VI otherwise known as Watts = Voltage * Current. We can calculate total Watts used by multiplying the voltages and currents at each sample point, then summing them up over a cycle & averaging to get the power used per cycle. Once we have Watts, its easy to just multiply that by 'time' to get Watt-hours!

Download and run the wattcher.py - watt grapher script from the download page

Now you can watch the last hour's worth of watt history (3600 seconds divided by 2 seconds per sample = 1800 samples) In the image above you can see as I dim a 40-watt lightbulb. The data is very 'scattered' looking because we have not done any low-pass filtering. If we had a better analog sampling rate, this may not be as big a deal but with only 17 samples to work with, precision is a little difficult

Done!

OK great! We have managed to read data, parse out the analog sensor payload and process it in a way that gives us meaningful graphs. Of course, this is great for instantaneous knowledge but it -would- be nice if we could have longer term storage, and also keep track of multiple sensors. In the next step we will do that by taking advantage of some free 'cloud computing' services!

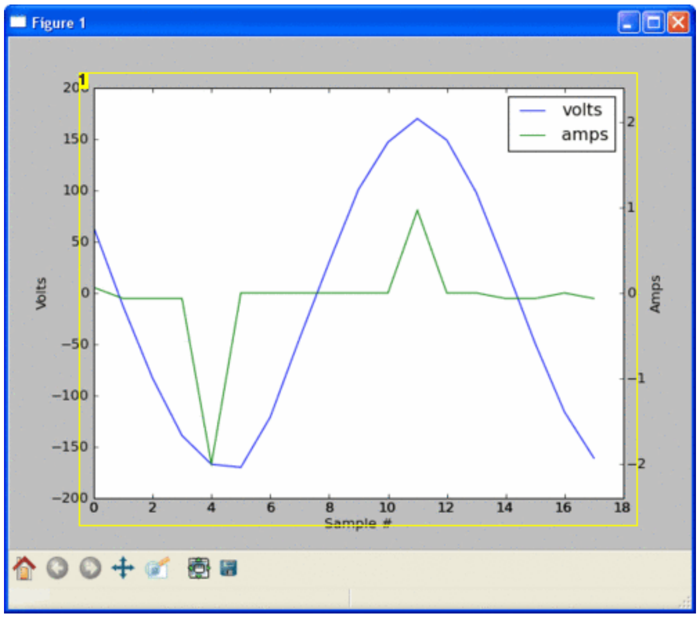


Image Notes
1. A laptop plugged in, switching power supply

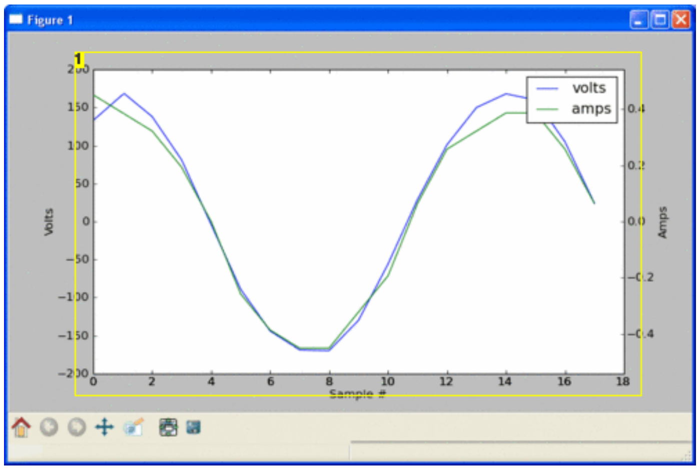


Image Notes
1. 40W lightbulb

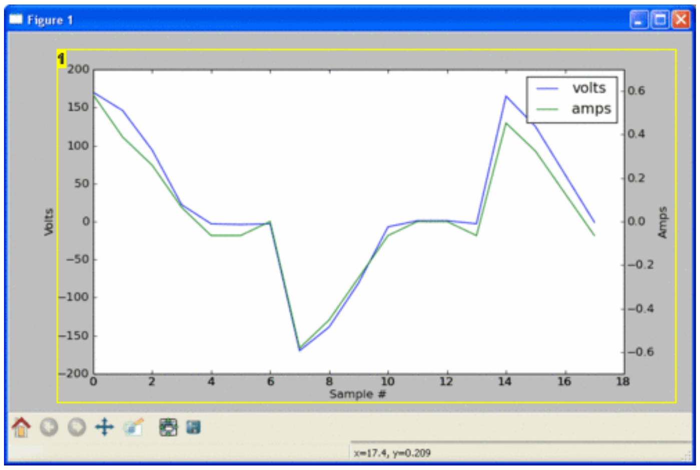
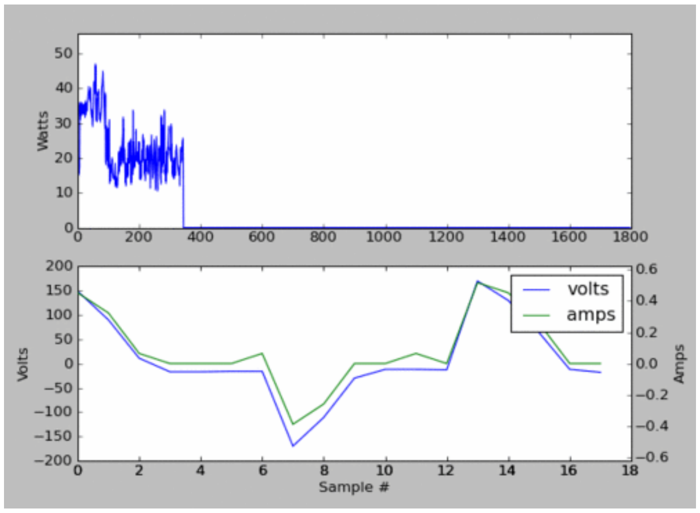


Image Notes
1. Light bulb on dimmer switch



Step 16: Design - store

Introduction

OK we are getting good data from our sensors, lets corral it into more useful chunks and store it in a database. We could make a database on the computer, but since we'd like to share this data, it makes more sense to put it online. There are custom services that are specifically designed to do this sort of thing like Pachube but I'm going to reinvent the wheel and design my own web-app that stores and displays energy data. (Mostly I want to play around with Google App Engine!)

You have 5 minutes!

We get data every few seconds from the XBee modem inside the kill-a-watt. We could, in theory, put data into our database every 2 seconds but that would quickly balloon the amount of storage necessary. It would also make sorting through the data difficult. So instead lets add up all the sensor data for 5 minutes and then take the average.

We'll do this by keeping two timers and one tally. One timer will track how long its been since the last sensor signal was sent, and the other will track if its been 5 minutes. The tally will store up all the Watt-hours (Watt measurements * time since last sensor data). Then at the end we can average by the 5 minutes

This chunk of code goes near the beginning, it creates the timers and tally and initializes them

```
...
fiveminutetimer = lasttime = time.time() # get the current time
cumulativewatthr = 0
...
```

Then later on, after we get our data we can put in this chunk of code:

```
# add up the delta-watthr used since last reading
# Figure out how many watt hours were used since last reading
elapsedseconds = time.time() - lasttime
dwatthr = (avgwatt * elapsedseconds) / (60.0 * 60.0) # 60 seconds in 60 minutes = 1 hr
lasttime = time.time()
print "\t\tWh used in last ",elapsedseconds," seconds: ",dwatthr
cumulativewatthr += dwatthr

# Determine the minute of the hour (ie 6:42 -> '42')
currminute = (int(time.time())/60) % 10
# Figure out if its been five minutes since our last save
if (((time.time() - fiveminutetimer) >= 60.0) and (currminute % 5 == 0)):
# Print out debug data, Wh used in last 5 minutes
avgwattsused = cumulativewatthr * (60.0*60.0 / (time.time() - fiveminutetimer))
print time.strftime("%Y %m %d, %H:%M"),", ",cumulativewatthr,"Wh = ",avgwattsused," W average")

# Reset our 5 minute timer
fiveminutetimer = time.time()
cumulativewatthr = 0
```

Note that we calculate delta-watthours, the small amount of power used every few seconds. Then we can get the average watts used by dividing the watthours by the number of hours that have passed (about 1/12th). Instead of going by exact 5 minutes, I decided to only report on the 5's of the hour (:05, :10, etc) so that its easier to send all the data at once if theres multiple sensors that started up at different times.

Download **wattcher-5minreporter.py** from the Download page. If you run this, you'll get a steady stream

Near the end you can see the timestamp, the Watthrs used in the last few minutes and the average Wattage

Multisensor!

We have good data but so far it only works with one sensor. Multiple sensors will mess it up! Time to add support for more than one XBee so that I can track a few rooms. I'll do that by creating an object class in python, and using the XBee address (remember that from part 1?) to track. I'll replace the code we just wrote with the following:

At the top, instead of the timer variables, I'll have a full class declaration, and create an array to store them:

```
####### store sensor data and array of histories per sensor
class Fiveminutehistory:
def init (self, sensornum):
self.sensornum = sensornum
self.fiveminutetimer = time.time() # track data over 5 minutes
self.lasttime = time.time()
self.cumulativewatthr = 0

def addwatthr(self, deltawatthr):
self.cumulativewatthr += float(deltawatthr)

def reset5mintimer(self):
self.cumulativewatthr = 0
self.fiveminutetimer = time.time()

def avgwattover5min(self):
return self.cumulativewatthr * (60.0*60.0 / (time.time() - self.fiveminutetimer))

def str (self):
return "[id#: %d, 5mintimer: %f, lasttime; %f, cumulativewatthr: %f]" % (self.sensornum, self.fiveminutetimer, self.lasttime, self.cumulativewatthr)

######### an array of histories
sensorhistories = []
```

When the object is initialized with the sensor ID number, it also sets up the two timers and cumulative Watthrs tracked. I also created a few helper functions that will make the code cleaner

Right below that I'll create a little function to help me create and retrieve these objects. Given an XBee ID number it either makes a new one or gets the reference to it

```
####### retriever
def findsensorhistory(sensornum):
for history in sensorhistories:
if history.sensornum == sensornum:
return history
# none found, create it!
history = Fiveminutehistory(sensornum)
sensorhistories.append(history)
return history
```

Finally, instead of the average Watt calculation code written up above, we'll replace it with the following chunk, which retreives the object and tracks power usage with the object timers

```
# retreive the history for this sensor
sensorhistory = findsensorhistory(xb.address_16)
#print sensorhistory

# add up the delta-watthr used since last reading
# Figure out how many watt hours were used since last reading
elapsedseconds = time.time() - sensorhistory.lasttime
dwatthr = (avgwatt * elapsedseconds) / (60.0 * 60.0) # 60 seconds in 60 minutes = 1 hr
sensorhistory.lasttime = time.time()
```

```
print "\t\tWh used in last ",elapsedseconds," seconds: ",dwatthr
sensorhistory.addwatthr(dwatthr)

# Determine the minute of the hour (ie 6:42 -> '42')
currminute = (int(time.time())/60) % 10
# Figure out if its been five minutes since our last save
if (((time.time() - sensorhistory.fiveminutetimer) >= 60.0) and (currminute % 5 == 0)):
# Print out debug data, Wh used in last 5 minutes
avgwattsused = sensorhistory.avgwattover5min()
print time.strftime("%Y %m %d, %H:%M"),", ",sensorhistory.cumulativewatthr,"Wh = ",avgwattsused," W average"

# Reset our 5 minute timer
sensorhistory.reset5mintimer()
```

The code basically acts the same except now it wont choke on multiple sensor data! Below, my two Kill-a-Watts, one with a computer attached (100W) and another with a lamp (40W)

Onto the database!

The App Engine

So we want to have an networked computer to store this data so we can share the data, but we really don't want to have to run a server from home! What to do? Well as mentioned before, you can use Pachube or similar, but I will show how to roll-your-own with Google App Engine (GAE) . GAE is basically a free mini-webserver hosted by Google, that will run basic webapps without the hassle of administrating a database server. Each webapp has storage, some frameworks and can use Google accounts for authentication. To get started I suggest checking out the GAE website, documentation, etc. I'll assume you've gone through the tutorials and jump right into designing my power data storage app called Wattcher (a little confusing I know)

First, the **app.yaml** file which defines my app looks like this:

```
application: wattcher
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
script: wattcherapp.py
```

Pretty simple, just says that the app uses **wattcherapp.py** as the source file

Next, we'll dive into the python code for our webapp. First, the includes and database index. To create a database, we actually define it -in the python file-, GAE then figures out what kind of database to create for you by following those directions (very different than MySQL where you'd create the DB separately)

```
import cgi, datetime

from google.appengine.api import users
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext import db

class Powerusage(db.Model):
author = db.UserProperty() # the user
sensornum = db.IntegerProperty() # can have multiple sensors
watt = db.FloatProperty() # each sending us latest Watt measurement
date = db.DateTimeProperty(auto_now_add=True) # timestamp
```

We use the default includes. We have a single database table called **Powerusage** , and it has 4 entries: one for the user, one for the sensor number, one for the last reported Watts used and one for a datestamp

Each 'page' or function of our webapp needs its own class. Lets start with the function that allows us to store data in the DB. I'll call it PowerUpdate.

```
class PowerUpdate(webapp.RequestHandler):
def get(self):

# make the user log in
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))

powerusage = Powerusage()

if users.get_current_user():
powerusage.author = users.get_current_user()
#print self.request
if self.request.get('watt'):
powerusage.watt = float(self.request.get('watt'))
else:
self.response.out.write('Couldnt find \'watt\' GET property!')
return
if self.request.get('sensornum'):
powerusage.sensornum = int(self.request.get('sensornum'))
else:
powerusage.sensornum = 0 # assume theres just one or something

powerusage.put()
self.response.out.write('OK!')
```

When we send a request to do that with a GET call (ie requesting the webpage), we'll first make sure the user is authenticated and logged in so we know their name. Then we'll create a new database entry by initializing a new instantiation of Powerusage. Then we'll look the GET request for the watt data, which would be in the format watt=39.2 or similar. That's parsed for us, thankfully and we can also get the sensor number which is passed in the format sensornum=3. Finally we can store the data into the permanent database

Next is a useful debugging function, it will simply print out all the data it has received for your account!

```
class DumpData(webapp.RequestHandler):
def get(self):

# make the user log in
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))

self.response.out.write('<html><body>Here is all the data you have sent us:<p>')

powerusages = db.GqlQuery("SELECT * FROM Powerusage WHERE author = :1 ORDER BY date", users.get_current_user())

for powerused in powerusages:
if powerused.sensornum:
self.response.out.write('<b>%s</b>\'s sensor #%d' %
(powerused.author.nickname(), powerused.sensornum))
else:
self.response.out.write(<b>%s</b>' % powerused.author.nickname())

self.response.out.write(' used: %f Watts at %s<p>' % (powerused.watt, powerused.date))
self.response.out.write("</body></html>")
```

This function simply SELECT's (retrieves) all the entries, sorts them by date and prints out each one at a time

Finally we'll make a basic 'front page' that will show the last couple of datapoints sent

```
class MainPage(webapp.RequestHandler):
def get(self):

self.response.out.write('<html><body>Welcome to Wattcher!<p>Here are the last 10 datapoints:<p>')

powerusages = db.GqlQuery("SELECT * FROM Powerusage ORDER BY date DESC LIMIT 10")

for powerused in powerusages:
if powerused.sensornum:
self.response.out.write('<b>%s</b>\'s sensor #%d' %
(powerused.author.nickname(), powerused.sensornum))
else:
self.response.out.write('<b>%s</b>' % powerused.author.nickname())

self.response.out.write(' used: %f Watts at %s<p>' % (powerused.watt, powerused.date))
self.response.out.write("</body></html>")
```

Its very similar to the DataDump function but its only 10 points of data and from all users, nice to use when you just want to 'check it out' but don't want to log in

Finally, we have a little initializer structure that tells GAE what pages link to what functions

```
application = webapp.WSGIApplication(
[('/', MainPage),]
('/report', PowerUpdate),
('/dump', DumpData)],
debug=True)

def main():
run_wsgi_app(application)

if name == "main ":
main()
```

Test!

OK lets try it out, first lets visit **http://wattcher.appspot.com/report**

Remember we made it a requirement to supply -some- data. Lets try again **http://wattcher.appspot.com/report?watt=19.22&sensornum=1**

Yay we got an OK! Lets check out the data stored by visiting **http://wattcher.appspot.com/dump**

There's two entries because I did a little testing beforehand but you can see that there are 2 entries. Nice!

We can also visit the GAE control panel and browse the data 'by hand'

Anyways, now that that's working, lets go back and add the reporting technology to our sensor-reader script

Getting the report out

Only a little more hacking on the computer script and we're done. We want to add support for sending data to GAE. Unfortunately right now our authentication is done through Google accounts so its not easy to run on an Arduino. To adapt it you'd have to send the username in the Report GET and hope nobody else uses the same one (unless you also add a basic password system)

Anyhow, I totally ripped off how to do this from some nice people on the Internet

Download **appengineauth.py** from the download page , and change the first few lines if necessary. We hardcode the URL we're going to and the account/password as well as the GAE app name

```
users_email_address = "myaccount@gmail.com"
users_password = "mypassword"
my_app_name = "wattcher"
target_authenticated_google_app_engine_uri = 'http://wattcher.appspot.com/report'
```

The real work happens at this function **sendreport** where it connects and sends the Watt data to the GAE site

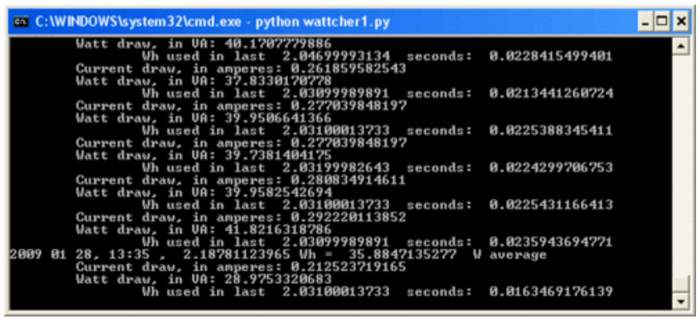
```
def sendreport(sensornum, watt):
# this is where I actually want to go to
serv_uri = target_authenticated_google_app_engine_uri + "?watt="+str(watt)+"&sensornum="+str(sensornum)

serv_args = {}
serv_args['continue'] = serv_uri
serv_args['auth'] = authtoken

full_serv_uri = "http://wattcher.appspot.com/_ah/login?%s" % (urllib.urlencode(serv_args))

serv_req = urllib2.Request(full_serv_uri)
serv_resp = urllib2.urlopen(serv_req)
serv_resp_body = serv_resp.read()

# serv_resp_body should contain the contents of the
# target_authenticated_google_app_engine_uri page - as we will have been
# redirected to that page automatically
#
# to prove this, I'm just gonna print it out
print serv_resp_body
```

Finally, we wrap up by adding the following lines to our computer script, which will send the data nicely over to GAE!

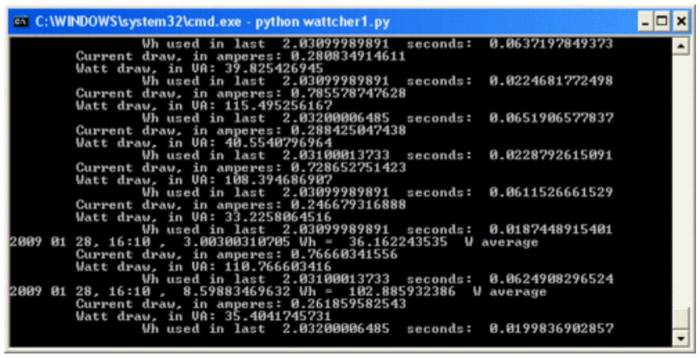
```
# Also, send it to the app engine
appengineauth.sendreport(xb.address_16, avgwattsused)
```

You can download the final script **wattcher.py - final** from the download page !

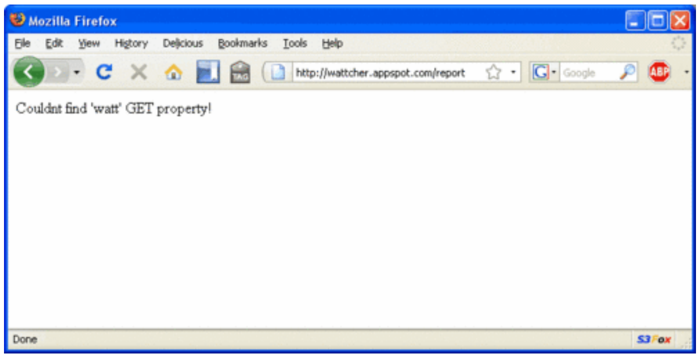
Don't forget to visit **wattcher.appspot.com** to check out the lastest readings



```
        Watt draw, in VA: 40.1707779886
                Wh used in last  2.04699993134  seconds:  0.0228415499401
        Current draw, in amperes: 0.261859582543
        Watt draw, in VA: 37.8330170778
                Wh used in last  2.03099989891  seconds:  0.0213441260724
        Current draw, in amperes: 0.277039848197
        Watt draw, in VA: 39.9506641366
                Wh used in last  2.03100013733  seconds:  0.0225388345411
        Current draw, in amperes: 0.277039848197
        Watt draw, in VA: 39.7381404175
                Wh used in last  2.03199982643  seconds:  0.0224299706753
        Current draw, in amperes: 0.280834914611
        Watt draw, in VA: 39.9582542694
                Wh used in last  2.03100013733  seconds:  0.0225431166413
        Current draw, in amperes: 0.292220113852
        Watt draw, in VA: 41.8216318786
                Wh used in last  2.03099989891  seconds:  0.0235943694771
2009 01 28, 13:35 ,  2.18781123965 Wh =  35.8847135277  W average
        Current draw, in amperes: 0.212523719165
        Watt draw, in VA: 28.9753320683
                Wh used in last  2.03100013733  seconds:  0.0163469176139
```

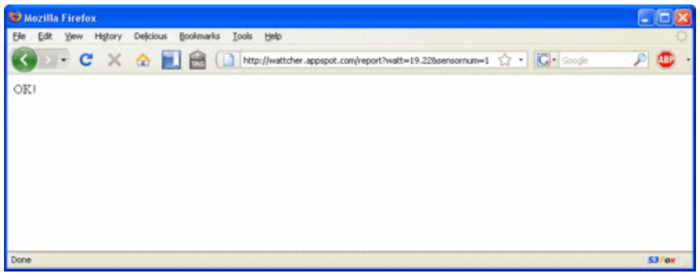


```
                Wh used in last  2.03099989891  seconds:  0.0637197849373
        Current draw, in amperes: 0.280834914611
        Watt draw, in VA: 39.825426945
                Wh used in last  2.03099989891  seconds:  0.0224681772498
        Current draw, in amperes: 0.785578747628
        Watt draw, in VA: 115.495256167
                Wh used in last  2.03200006485  seconds:  0.0651906577837
        Current draw, in amperes: 0.288425047438
        Watt draw, in VA: 40.5540796964
                Wh used in last  2.03100013733  seconds:  0.0228792615091
        Current draw, in amperes: 0.728652751423
        Watt draw, in VA: 108.394686907
                Wh used in last  2.03099989891  seconds:  0.0611526661529
        Current draw, in amperes: 0.246679316888
        Watt draw, in VA: 33.2258064516
                Wh used in last  2.03099989891  seconds:  0.0187448915401
2009 01 28, 16:10 ,  3.00300310705 Wh =  36.162243535  W average
        Current draw, in amperes: 0.76660341556
        Watt draw, in VA: 110.766603416
                Wh used in last  2.03100013733  seconds:  0.0624908296524
2009 01 28, 16:10 ,  8.59883469632 Wh =  102.885932386  W average
        Current draw, in amperes: 0.261859582543
        Watt draw, in VA: 35.4041745731
                Wh used in last  2.03200006485  seconds:  0.0199836902857
```



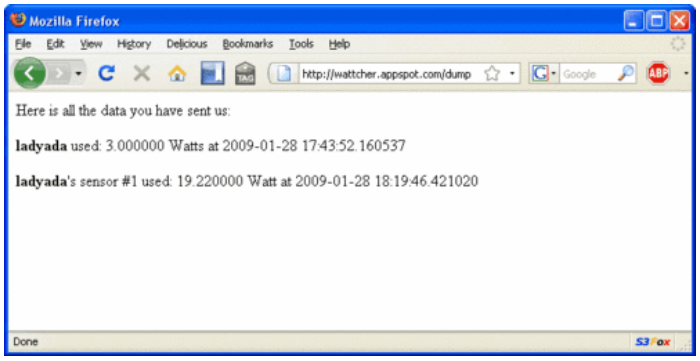
http://wattcher.appspot.com/report

Couldnt find 'watt' GET property!



http://wattcher.appspot.com/report?watt=19.22&sensornum=1

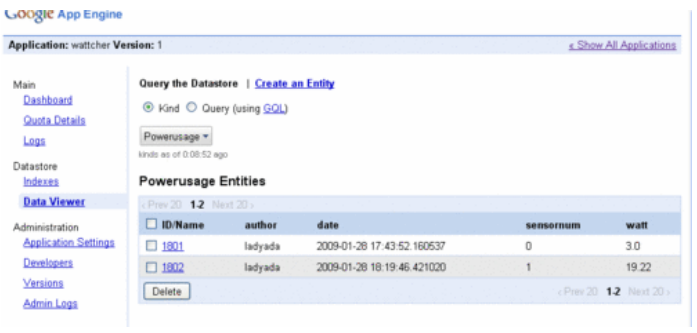
OK!


Mozilla Firefox
File Edit View History Delicious Bookmarks Tools Help
http://wattcher.appspot.com/dump
Done

Here is all the data you have sent us:

ladyada used: 3.000000 Watts at 2009-01-28 17:43:52.160537

ladyada's sensor #1 used: 19.220000 Watt at 2009-01-28 18:19:46.421020



Application: wattcher **Version:** 1

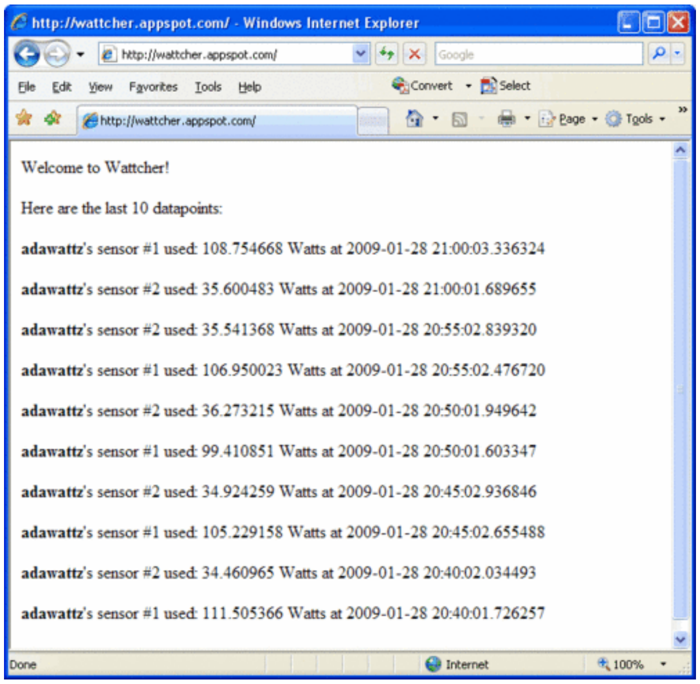
Query the Datastore

Kind [✓] Query (using GQL)

kinds as of 0:08:52 ago

Powerusage Entities

ID/Name	author	date	sensornum	watt
1801	ladyada	2009-01-28 17:43:52.160537	0	3.0
1802	ladyada	2009-01-28 18:19:46.421020	1	19.22


http://wattcher.appspot.com/ - Windows Internet Explorer
http://wattcher.appspot.com/
File Edit View Favorites Tools Help
Convert Select
Page Tools
Done
Internet
100%

Welcome to Wattcher!

Here are the last 10 datapoints:

adawattz's sensor #1 used: 108.754668 Watts at 2009-01-28 21:00:03.336324

adawattz's sensor #2 used: 35.600483 Watts at 2009-01-28 21:00:01.689655

adawattz's sensor #2 used: 35.541368 Watts at 2009-01-28 20:55:02.839320

adawattz's sensor #1 used: 106.950023 Watts at 2009-01-28 20:55:02.476720

adawattz's sensor #2 used: 36.273215 Watts at 2009-01-28 20:50:01.949642

adawattz's sensor #1 used: 99.410851 Watts at 2009-01-28 20:50:01.603347

adawattz's sensor #2 used: 34.924259 Watts at 2009-01-28 20:45:02.936846

adawattz's sensor #1 used: 105.229158 Watts at 2009-01-28 20:45:02.655488

adawattz's sensor #2 used: 34.460965 Watts at 2009-01-28 20:40:02.034493

adawattz's sensor #1 used: 111.505366 Watts at 2009-01-28 20:40:01.726257

Step 17: Design - graph

Making pretty pictures

Data is great, but visualizations are better. In this step we'll manipulate our stored history so that we can make really nice graphs!

First we'll start by making our sensors named, so that its easier for us to keep track of which is what. Then we'll look at our graph options and data formats. Finally we'll reformat our data to make it ready for graphing

Configuring the sensor names

Its no fun to have data marked as "sensor #1" so I added a 'config' page where the app engine code looks at what sensor numbers have sent data to the database and then allows you to name them. Of course, you need to have the sensor on and sending data -first- before this will work

The configure screen looks something like the image below.

This code uses GET when it should really use POST. I'm kinda old and dont like debugging with POST so...yeah.

```
class Configure(webapp.RequestHandler):
def get(self):
# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

self.response.out.write('<html><body>Set up your sensornode names here:<p>')

# find all the sensors up to #10
sensorset = []
for i in range(10):
c = db.GqlQuery("SELECT * FROM Powerusage WHERE author = :1 and sensornum = :2", users.get_current_user(), i)
if c.get():
sensorset.append(i)
```

```
self.response.out.write('<form action="/config" method="get">')
for sensor in sensorset:
name = ""
currnamequery = db.GqlQuery("SELECT * FROM Sensorname WHERE author = :1 AND sensornum = :2", users.get_current_user(), sensor)
currname = currnamequery.get()

# first see if we're setting it!
if self.request.get('sensornum'+str(sensor)):
name = self.request.get('sensornum'+str(sensor))
if not currname:
currname = Sensorname() # create a new entry
currname.sensornum = sensor
currname.author = users.get_current_user()
currname.sensorname = name
currname.put()
else:
# we're not setting it so fetch current entry
if currname:
name = currname.sensorname

self.response.out.write('Sensor #'+str(sensor)+': <input type="text" name="sensornum'+str(sensor)+'" value="'+name+'"></text><p>')

self.response.out.write("""<div><input type="submit" value="Change names"></div>
</form>
</body>
</html>""")
```

Now we can have more useful data in the history dump

Now we can see that Phil is mostly to blame for our power bill!

Google Visualizer

So we have data and we'd like to see our power usage history. Graphing data is a lot of work, and I'm lazy. So I look online and find that Google -also- has a visualization API! This means I don't have to write a bunch of graphical code, and can just plug into their system. Sweet!

OK checking out the gallery of available visualizations , I'm fond of this one, the Annotated Time Line

Note how you can easily see the graphs, scroll around, zoom in and out and each plot is labeled. Perfect for plotting power data!

Data formatting

Theres a few restrictions to how we get the data to the visualization api and our best option is JSon data. As far as I can tell, JSON is what happened when everyone said "wow, XML is really bulky and wasteful". Anyhow, theres like 4 layers of framework and interpretive data structions and in the end there was a pretty easy to use library written by the Google Visualizations team that let me 'just do it' with a single call by putting the data into a python 'dictionary' in a certain format.

Lets go through the code in sections, since the function is quite long

```
class JSONout(webapp.RequestHandler):
def get(self):

# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

# assume we want 24 hours of data
historytimebegin = 24
if self.request.get('bhours'):
historytimebegin = int(self.request.get('bhours'))

# assume we want data starting from 0 hours ago
historytimeend = 0
if self.request.get('ehours'):
historytimeend = int(self.request.get('ehours'))

# data format for JSON happiness
datastore = []
columnnames = ["date"]
columnset = set(columnnames)
description ={"date": ("datetime", "Date")}

# the names of each sensor, if configured
sensornames = [None] * 10
```

First up we get the user we're going to be looking up the data for. Then we have two variables for defining the amount of data to grab. One is "ehours" (end hours) and the other is "bhours". So if you wanted the last 5 hours, bhours would be 5 and ehours would be 0. If you wanted 5 hours from one day ago, bhours would be 29 and ehours would be 24. datastore is where we will corall all the data. columnnames and description are the 'names' of each column. We always have a date column, then another column for each sensor stream. We also have a seperate array to cache the special sensor names.

onto the next section! Here is where we actually grab data from the database. Now app engine has this annoying restriction, you can only get 1000 points of data at once so what I do is go through it 12 hours at a time. The final datastore has all the points but its easier on the database, I guess. One thing that's confusing perhaps is each column has a name and a description. The name is short, say "watts3" for sensor #3, but the description might be "Limor's workbench". I don't even remember writing this code so maybe you can figure it out on your own?

```
# we cant grab more than 1000 datapoints, thanks to free-app-engine restriction
# thats about 3 sensors's worth in one day
# so we will restrict to only grabbing 12 hours of data at a time, about 7 sensors worth

while (historytimebegin > historytimeend):
if (historytimebegin - historytimeend) > 12:
timebegin = datetime.timedelta(hours = -historytimebegin)
timeend = datetime.timedelta(hours = -(historytimebegin-12))
historytimebegin -= 12
else:
timebegin = datetime.timedelta(hours = -historytimebegin)
historytimebegin = 0
timeend = datetime.timedelta(hours = -historytimeend)

# grab all the sensor data for that time chunk
powerusages = db.GqlQuery("SELECT * FROM Powerusage WHERE date > :1 AND date < :2 AND author = :3 ORDER BY date", datetime.datetime.now()+timebegin,
datetime.datetime.now()+timeend, account)

# sort them into the proper format and add sensor names from that DB if not done yet
for powerused in powerusages:
coln = "watts" + str(powerused.sensornum)
entry = {"date": powerused.date.replace(tzinfo=utc).astimezone(est), coln: powerused.watt}
if not (coln in columnset):
columnnames.append(coln)
columnset = set(columnnames)
# find the sensor name, if we can
if (len(sensornames) < powerused.sensornum) or (not sensornames[powerused.sensornum]):
currnamequery = db.GqlQuery("SELECT * FROM Sensorname WHERE author = :1 AND sensornum = :2", account, powerused.sensornum)
name = currnamequery.get()

if not name:
sensornames[powerused.sensornum] = "sensor #"+str(powerused.sensornum)
else:
sensornames[powerused.sensornum] = name.sensorname

description[coln] = ("number", sensornames[powerused.sensornum])
#self.response.out.write(sensornames)

# add one entry at a time
datastore.append(entry)
```

Finally at the end of all the looping, we call the magic function that turns the dictionary into JSON, wrap it in the proper Google Visualization package, then spit it out!

```
# OK all the data is ready to go, print it out in JSON format!
data_table = gviz_api.DataTable(description)
data_table.LoadData(datastore)
self.response.headers['Content-Type'] = 'text/plain'
self.response.out.write(data_table.ToJSonResponse(columns_order=(columnnames),
order_by="date"))
```

If you were to visit **http://wattcher.appspot.com/visquery.json?user=adawattz@gmail.com&bhours=1** it would output something like this:

```
google.visualization.Query.setResponse({'version':'0.5', 'reqId':'0', 'status':'OK', 'table': {cols:
[{id:'date',label:'Date',type:'datetime'},{id:'watts1',label:'Limor',type:'number'},{id:'watts5',label:'Workbench',type:'number'},{id:'watts2',label:'Adafruit',type:'number'},{id:'watts
[{c:[{v:new Date(2009,1,25,21,20,2)},{v:64.8332291619},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,20,3)},,{v:230.122099757},,{v:null}]},{c:[{v:new
Date(2009,1,25,21,20,3)},,,{v:65.4923925044},{v:null}]},{c:[{v:new Date(2009,1,25,21,20,4)},,,,{v:48.6947643311}]},{c:[{v:new
Date(2009,1,25,21,25,3)},,{v:228.409810208},,{v:null}]},{c:[{v:new Date(2009,1,25,21,25,3)},{v:67.3574917331},,,{v:null}]},{c:[{v:new
Date(2009,1,25,21,25,3)},,,{v:66.0046383897},{v:null}]},{c:[{v:new Date(2009,1,25,21,25,4)},,,,{v:47.3892235642}]},{c:[{v:new
Date(2009,1,25,21,30,2)},{v:84.9379517795},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,30,3)},,,,{v:99.7553490071}]},{c:[{v:new
Date(2009,1,25,21,30,5)},,{v:229.73642288},,{v:null}]},{c:[{v:new Date(2009,1,25,21,30,6)},,,{v:66.6556291818},{v:null}]},{c:[{v:new
Date(2009,1,25,21,35,2)},,,{v:67.3146052998},{v:null}]},{c:[{v:new Date(2009,1,25,21,35,3)},{v:96.2322216676},,,{v:null}]},{c:[{v:new
Date(2009,1,25,21,35,3)},,{v:226.678267688},,{v:null}]},{c:[{v:new Date(2009,1,25,21,35,4)},,,,{v:158.428422765}]},{c:[{v:new
Date(2009,1,25,21,40,3)},,{v:232.644574879},,{v:null}]},{c:[{v:new Date(2009,1,25,21,40,4)},,,,{v:153.666193493}]},{c:[{v:new
Date(2009,1,25,21,40,6)},,,{v:66.7874343225},{v:null}]},{c:[{v:new Date(2009,1,25,21,40,12)},{v:95.0019590395},,,{v:null}]},{c:[{v:new
Date(2009,1,25,21,40,21)},{v:95.0144043571},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,40,23)},,,{v:66.8060307611},{v:null}]},{c:[{v:new
Date(2009,1,25,21,45,2)},,,{v:66.9814723201},{v:null}]},{c:[{v:new Date(2009,1,25,21,45,3)},,{v:226.036818816},,{v:null}]},{c:[{v:new
Date(2009,1,25,21,45,3)},{v:99.2775581827},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,45,4)},,,,{v:154.261889366}]},{c:[{v:new
Date(2009,1,25,21,50,4)},{v:102.104642018},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,50,4)},,,,{v:155.441084531}]},{c:[{v:new
Date(2009,1,25,21,50,5)},,,{v:67.0087146687},{v:null}]},{c:[{v:new Date(2009,1,25,21,50,5)},,{v:230.678636915},,{v:null}]},{c:[{v:new
Date(2009,1,25,21,55,3)},{v:103.493297176},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,55,3)},,,,{v:151.309223916}]},{c:[{v:new
Date(2009,1,25,21,55,4)},,,{v:66.9174858741},{v:null}]},{c:[{v:new Date(2009,1,25,21,55,4)},,{v:227.765325835},,{v:null}]},{c:[{v:new
Date(2009,1,25,22,0,3)},,,{v:67.0004310254},{v:null}]},{c:[{v:new Date(2009,1,25,22,0,3)},,,,{v:150.389989112}]},{c:[{v:new
Date(2009,1,25,22,0,3)},,{v:230.892049553},,{v:null}]},{c:[{v:new Date(2009,1,25,22,0,4)},{v:92.2432771363},,,{v:null}]},{c:[{v:new
Date(2009,1,25,22,15,3)},{v:97.5910440774},,,{v:null}]},{c:[{v:new Date(2009,1,25,22,15,3)},,,,{v:143.722595861}]},{c:[{v:new
Date(2009,1,25,22,15,4)},,,{v:64.4898008851},{v:null}]},{c:[{v:new Date(2009,1,25,22,15,4)},,{v:222.357617868},,{v:null}]}]}});
```

Anyways, you can kinda see the data, also note its actually a function call, this stuff is really kinky!

Now go to the Google Visualizations Playground and enter in that URL into the sandbox

And you can see the visualization itself pop out! (this is just a screen shot so go do it yerself if you want to mess around)

OK go mess around, adding and changing **bhours** and **ehours**

Wrapping up the visualization

OK we're nearly done. Now we just need to basically grab the code from the sandbox and make it a subpage in our app engine...like so:

```
class Visualize(webapp.RequestHandler):
def get(self):

# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

historytimebegin = 24 # assume 24 hours
if self.request.get('bhours'):
historytimebegin = int(self.request.get('bhours'))

historytimeend = 0 # assume 0 hours ago
if self.request.get('ehours'):
historytimeend = int(self.request.get('ehours'))

# get the first part, headers, out
self.response.out.write(
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>Google Visualization API Sample</title>
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load("visualization", "1", {packages: ["annotatedtimeline"]});

function drawVisualizations() {
)

# create our visualization
self.response.out.write(new google.visualization.Query("http://wattcher.appspot.com/visquery.json?user= +
account.email()+&bhours= +str(historytimebegin)+").send(
function(response) {
new google.visualization.AnnotatedTimeLine(
document.getElementById("visualization")).
draw(response.getDataTable(), {"displayAnnotations": true});
});
)

self.response.out.write(}

google.setOnLoadCallback(drawVisualizations);
</script>
</head>
<body style="font-family: Arial;border: 0 none;">
<div id="visualization" style="width: 800px; height: 250px;"></div>
</body>
</html>)
```

The first part is pretty straight forward, get the user name or login. Then we will assume the user wants 1 last day of data, so set **bhours** and **ehours** . Then we literally just print out the code we copied from Google's Visualization sandbox, done!

Viz Viz Viz

The only thing I couldn't figure out is how to get 3 visualizations going on at once (last hour, day and week) with the above code. It just kinda broke. So for the triple view I had to use iframes :(

```
class VisualizeAll(webapp.RequestHandler):
def get(self):

# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

self.response.out.write(
<h2>Power usage over the last hour:</h2>
<iframe src ="graph?user=adawattz@gmail.com&bhours=1" frameborder="0" width="100%" height="300px">
<p>Your browser does not support iframes.</p>
</iframe>

<h2>Power usage over the last day:</h2>
<iframe src ="graph?user=adawattz@gmail.com&bhours=24" frameborder="0" width="100%" height="300px">
<p>Your browser does not support iframes.</p>
</iframe>
```

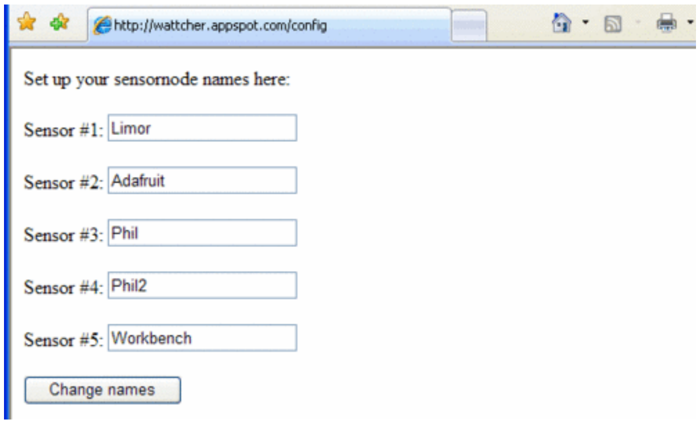
```
<h2>Power usage over the last week:</h2>
<iframe src ="graph?user=adawattz@gmail.com&bhours=168" frameborder="0" width="300%" height="500px">
<p>Your browser does not support iframes.</p>
</iframe>
```

)

Anyhow, it works just fine.

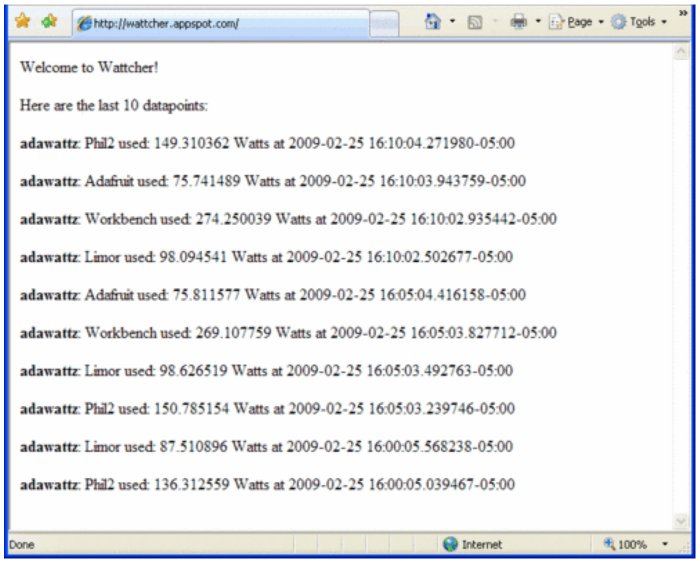
Timecodes!

The final thing that wont be reviewed here is how I got the date and times to be EST instead of UTC. As far as I can tell, its kind of broken and mysterious. Check the code if you want to figure it out.



Set up your sensornode names here:

Sensor #1:	Limor
Sensor #2:	Adafruit
Sensor #3:	Phil
Sensor #4:	Phil2
Sensor #5:	Workbench



Welcome to Wattcher!

Here are the last 10 datapoints:

adawattz: Phil2 used: 149.310362 Watts at 2009-02-25 16:10:04.271980-05:00

adawattz: Adafruit used: 75.741489 Watts at 2009-02-25 16:10:03.943759-05:00

adawattz: Workbench used: 274.250039 Watts at 2009-02-25 16:10:02.935442-05:00

adawattz: Limor used: 98.094541 Watts at 2009-02-25 16:10:02.502677-05:00

adawattz: Adafruit used: 75.811577 Watts at 2009-02-25 16:05:04.416158-05:00

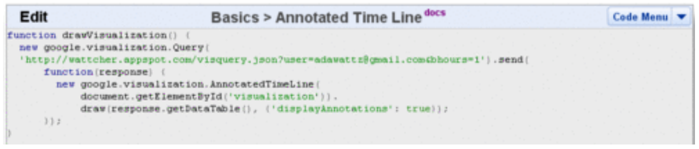
adawattz: Workbench used: 269.107759 Watts at 2009-02-25 16:05:03.827712-05:00

adawattz: Limor used: 98.626519 Watts at 2009-02-25 16:05:03.492763-05:00

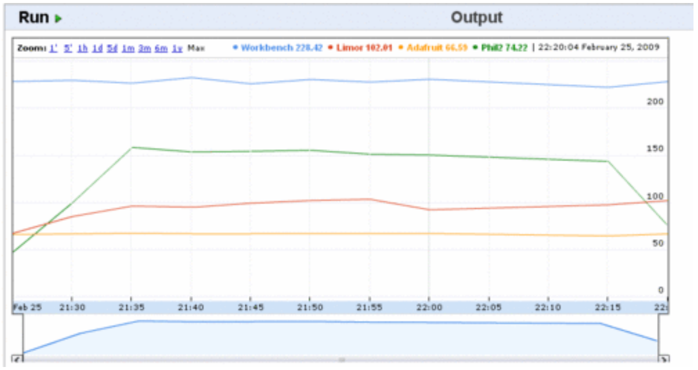
adawattz: Phil2 used: 150.785154 Watts at 2009-02-25 16:05:03.239746-05:00

adawattz: Limor used: 87.510896 Watts at 2009-02-25 16:00:05.568238-05:00

adawattz: Phil2 used: 136.312559 Watts at 2009-02-25 16:00:05.039467-05:00



```
function drawVisualization() {
  new google.visualization.Query(
  'http://wattcher.appspot.com/visquery.json?user=adawattz@gmail.com&bhours=1').send(
      function(response) {
        new google.visualization.AnnotatedTimeLine(
            document.getElementById('visualization')).
            draw(response.getDataTable(), {'displayAnnotations': true});
      });
}
```



Step 18: Resources

Other power monitoring projects!

Get some good ideas here!

- "Carbon Penance" a power monitor by Annina Rust that punishes the user
- Jason Winter's Real-Time power monitor
- Mazzini's project pushes data onto Pachube
- Pachube has lots of other projects!
- Furnace monitoring, using a DAQ board and phototransistor

Power monitoring products
Wanna just buy it?

- Black & Decker home electric meter watcher
- DIY KYOTO's power-clip Wattson is pretty
- CurrentCost

Websites & Software

- Myenergyusage.org (one fellow upgrading his Wattson's software by hand)
- Wattzon.org (social energy information)

Step 19: Download

Software

- Zip file of the most recent python scripts this is what you want if you've built a tweet-a-watt and you want to get your project running

X-CTU profiles

- Receiver, connected to computer
- Transmitter, embedded in the Kill-a-Watt. Change the unique ID if you have more than one!

Design files

All this stuff (other than the XBee library and the AppEngineAuth library, which are not written by me) is for you in the Public Domain! These are for debugging and design purpose and show how the project is put together. If you just want to "run the code" see the "software" above

- XBee library
- AppEngineAuth library for python
- Wattcher.py - graph just the mains data
- Wattcher.py - graph mains and wattage data
- Wattcher.py - Reports averages every 5 minutes
- Wattcher.py - Sends data to Wattcher Google App

For the latest code, please check the google code repository where you'll find any new code . And hey, are you good at this sort of code? I could really use some help. It works OK but it could look and run much better so please commit patches and hang out on the forum!

Related Instructables

Getting started with the Maxbotix sonar sensor - quick start guide by adafruit

Proto Shield by adafruit

Tilt Sensor Tutorial by adafruit

Musical MIDI Shoes by thobson

Self Destructing Message by foxxtrotalpha

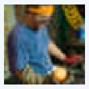
Hacking an Arduino ISP Shield for AtTiny45 / AtTiny85 by rleyland

Make your pet dishes tweet! by thoughtfix

The Best Arduino by msuzuki777

Bubblesteen Bubble Machine

by **belliedroot** on April 27, 2010



Author:belliedroot Bernard Katz Glass

I am a glass sculptor with a shop and gallery located in the Manayunk section of Philadelphia. Besides being a dad and running my business, I have a strong interest in electronics, and physical computing.

Intro: Bubblesteen Bubble Machine

Is it a 3D Spherical Atmosphere Encapsulated Phosphorous Printer? **YES!**

Is it a CNC Anti Gravity transparent Orb Machine! **YES!**

Its **The Bubblesteen Bubble Machine!** The spherical miracle that kids and cats have been waiting for. It comes complete with robotic edge detection(when a bubble hits an edge it pops, thus the edge has been detected).

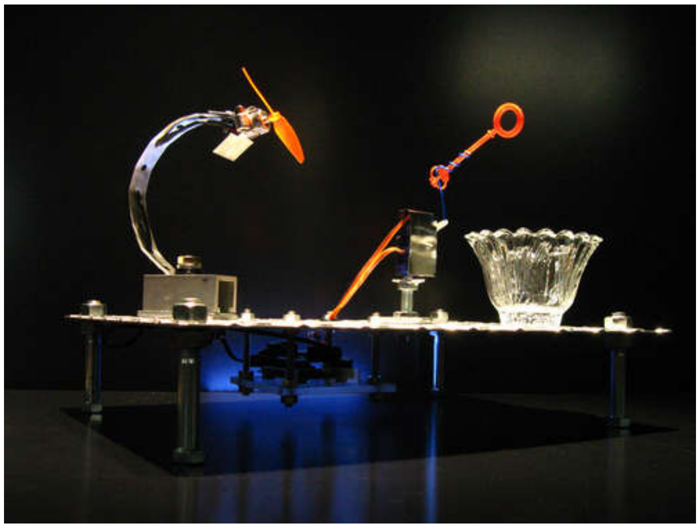
Turn up the sound and watch Lester the cat battle it out with the Bubblesteen!

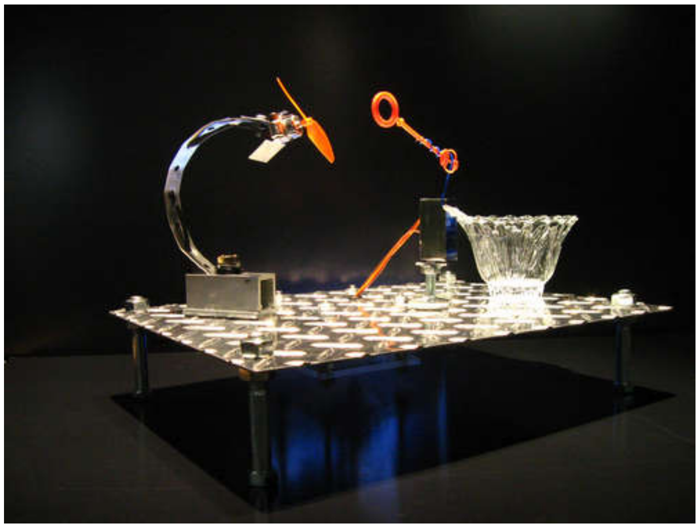
* No animals were harmed during testing.

This project came about after playing around with pan and tilt using servos. Most of the pan and tilt senereos I saw involved using webcams or some type of camera, which pan and tilt is perfect for. There are some good instructables and how-to's on the web for this very thing.

I may not have an available camera, but I did have some bubble mix :)

http://bernardkatz.com/





Step 1: Things you will need

This list is mostly for the electronic and mechanical stuff. How you create your own Bubblesteen will depend on your creativity and what you have laying around.

I will also include small tips on the materials I used and things to be careful of

1. 1. Arduino Duemilanove $30.00
2. 1 motor shield $19.50 (www.adafruit.com) * It is made to fit the Duemilanove
3. 2 micro servos- I used Hextronik HXT 500 $3.50 each (www.hobbyking.com)
4. 1 DC toy motor- something between 3v and 12v - easy to find, motor shield docs will help you decide if what you may already have will work.
5. 1 thing of bubble mix. - find at CVS or a dollar store. Some work better than others

These things I used, but are not critical. This is where your own creativity will need to come to play.

1. 1 roll of perferated metal tape- any hardware store
2. nuts and bolts of various sizes - thread count not critical :)
3. diamond plate- local scrap yard
4. aluminum channel- local scrap yard
5. 1 threaded rod hanger/ plate
6. earthquake putty or museum wax

7. 5 minute epoxy
8. 1/4" acrylic sheet- about 6" x 6" worth
9. acrylic adhesive

The tools you need will once again depend on what you build yours out of.

Image Notes
1. This stuff is almost as good as duct tape when it comes to versitility

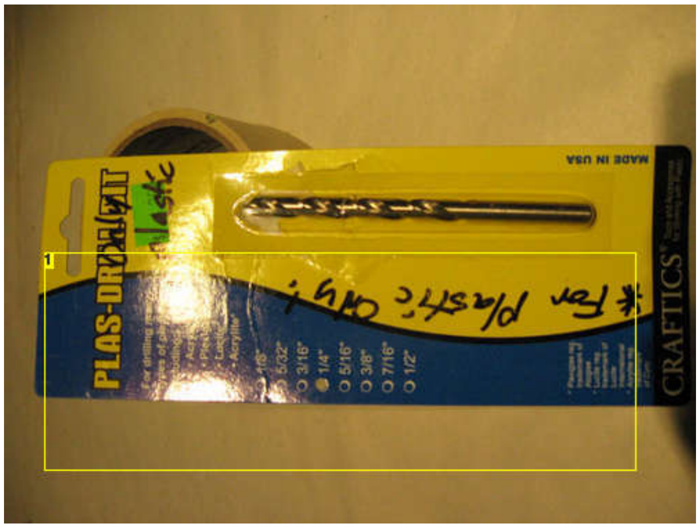


Image Notes
1. I highly recommend using special drill bits that are made for drilling plastic. Less chance of cracking the plastic. Can find on the web or local plastics supplier

Step 2: Dealing with the micro controller

I am going to assume that you have some experience with the Arduino. If not, you can check out either the web or this site for starting with arduino.

I will say that Ladyada's site www.adafruit.com is great for tutorials and buying arduino related stuff. In fact, you should refer to her site about using the Adafruit motor shield anyway.

The instructions on using the motor shield will tell you where to hook in the servo and DC motor, so I will not go into those details.

The code I used is posted below.

It is not the most elegant and for the most part hobbled together, but it works. Make sure you have the library for the motor shield

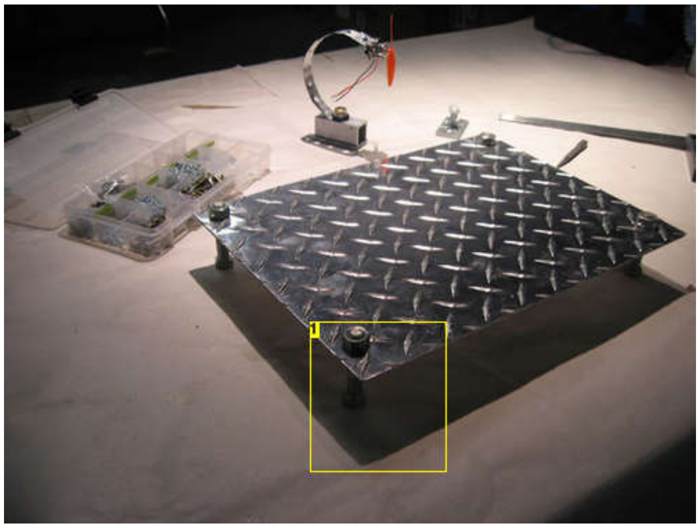
File Downloads

bubble_servo_DCMotor_april24b.pde (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'bubble_servo_DCMotor_april24b.pde']

Step 3: Putting it together

I had some diamond plate at the shop, so this became the platform.

Tip # 1 - The reason the arduino will be mounted below the platform is so it won't get **wet.** I am pretty positive a wet microcontroller doesn't work very well. Stuff will spash around!



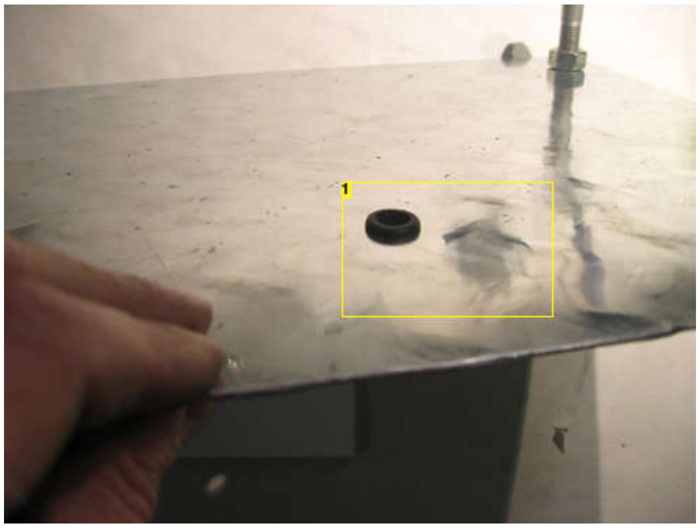
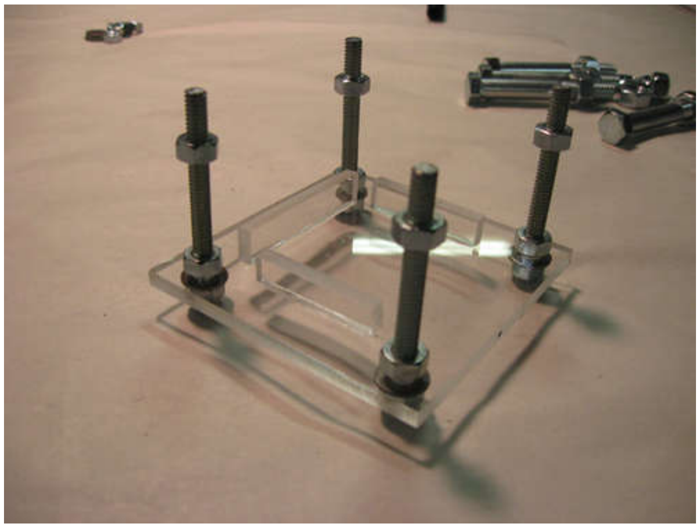


Image Notes
1. Bolts to raise the platform

Image Notes
1. gromet to protect servo and DC motor wires and to keep liquid from leaking to micro.

Step 4: Arduino & motor shield platform

These are just photos showing the construction of the microcontroller platform. I decided to have an on-off switch and a power jack.



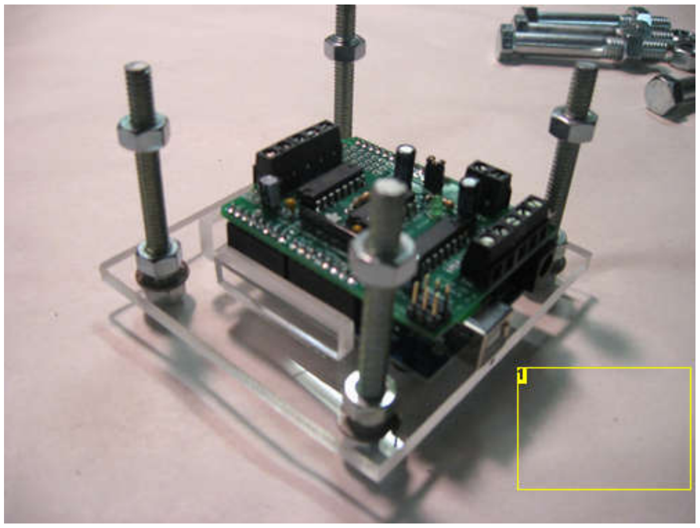


Image Notes
1. Arduino with motor shield

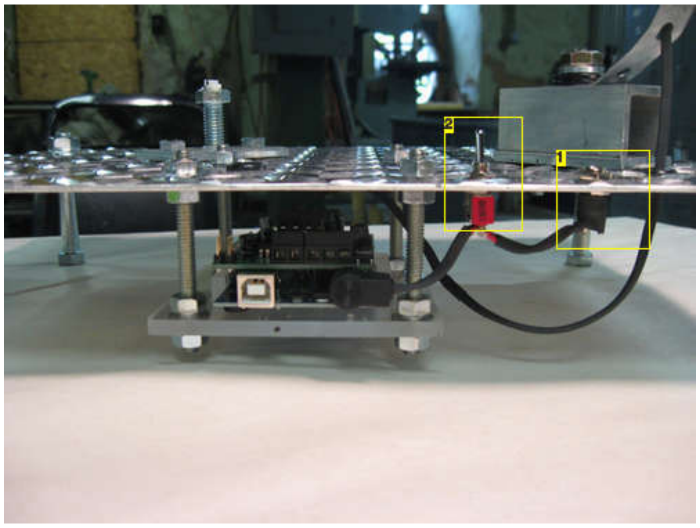


Image Notes
1. power jack
2. switch

Step 5:

Tip # 2 - I used museum wax to hold the servos together. It is fairly strong, but removable in case you need the servos for something else.

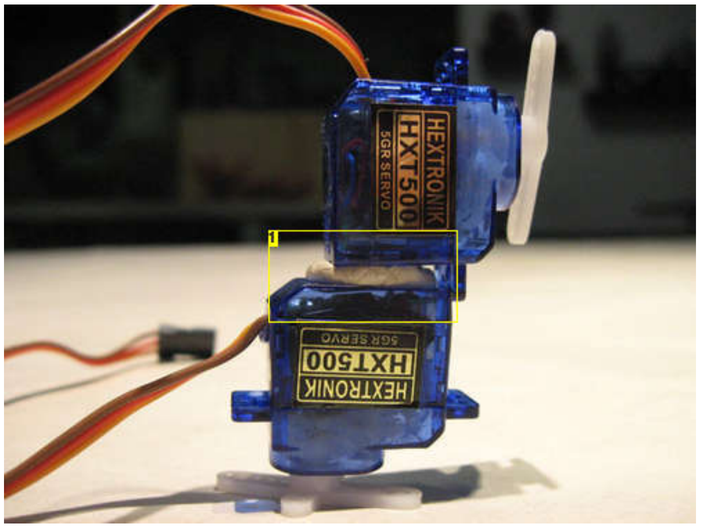


Image Notes
1. museum wax or poster putty

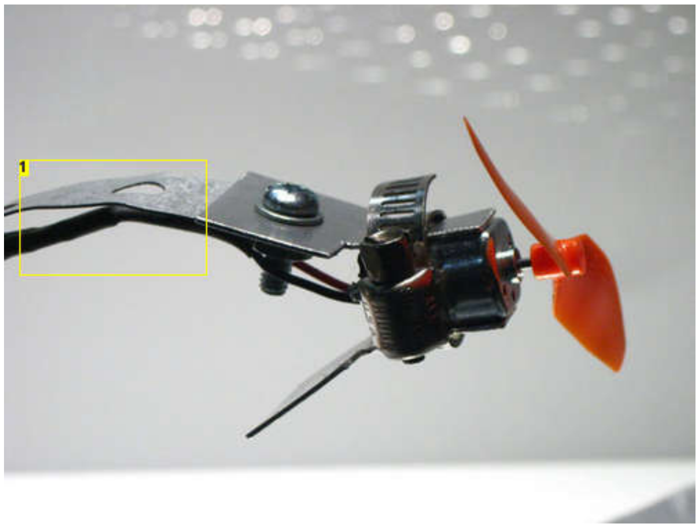
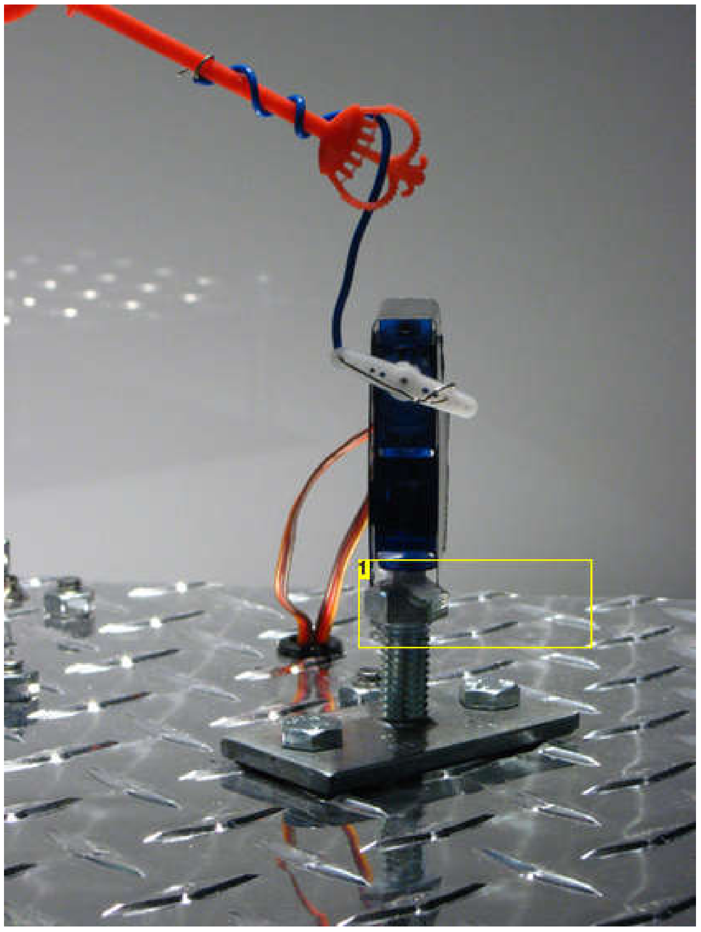


Image Notes
1. metal tape. easy to adjust height



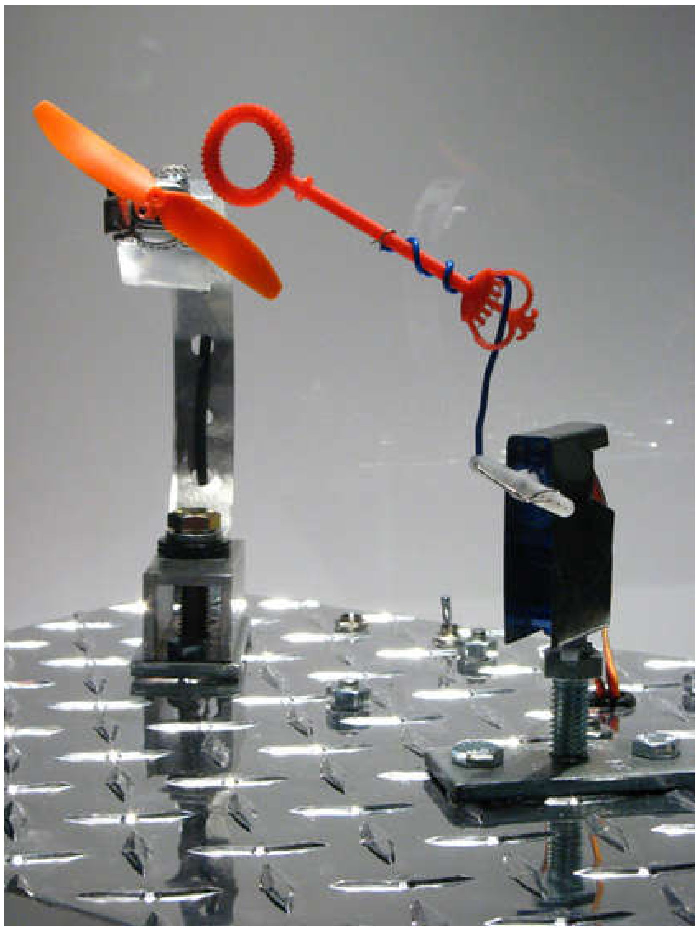


Image Notes
1. epoxy the servo horn to bolt

Step 6: Additional photos

That is about it. Here are some additional photos.

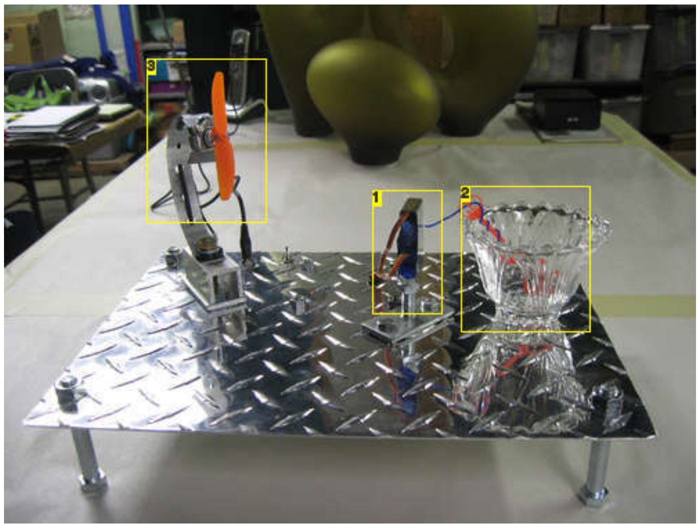
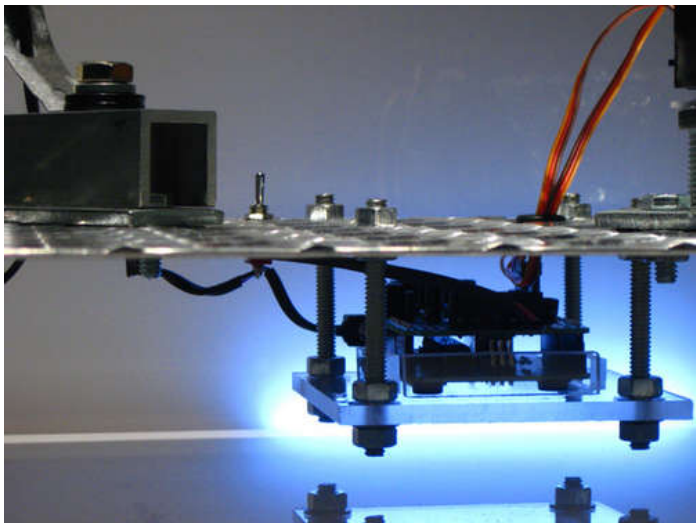


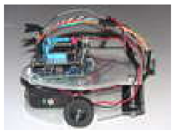
Image Notes
1. servos
2. bubble dipper and glass cup for bubble mix
3. small DC motor with propeller



Related Instructables

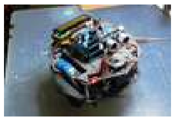
Arduino 4wd robot with ping sensor "J-Bot" by JamecoElectronics

Robotic Arm with Servo Motors by dustynrobots



Arduino-based line follower robot using Pololu QTR-8RC line sensor by techbitar

JabberBot! The Arduino robot with an ATMega brain and bluetooth braun! :-) (Photos) by declanshanaghy



My Arduino Ping Display Robot by Cello62

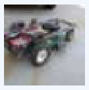
Bubblebot: Gigantic Bubble Generator by zvizvi

Robotic Talking Turret by RazorConcepts

Control an OWI Robotic Arm with Arduino by zlite

Arduino R/C Lawnmower (painted)

by johndavid400 on May 19, 2009



Author:johndavid400 author's website
I have always been one to take things apart to figure out how they work, so most of what I own has been dismantled. If it can't be taken apart or hacked, i'd rather not have it. And I like to do things the cheapest way possible, because I like to do a lot of things and I don't have a lot of money.

Intro: Arduino R/C Lawnmower (painted)

What this is:

This instructable will show you how to make your Arduino into an R/C interface that you can use for just about anything requiring remote control. I will also show you how I built an R/C lawnmower using my Arduino, a cheap R/C transmitter and receiver pair, and a couple of electric-wheelchair motors from Ebay. I have used this interface to control anything from basic LED's to Bipolar stepper motors, mini-robots, lifeless R/C cars from the thrift store, and even a 100lb lawnmower (all with appropriate motor controllers). It is very flexible and easy to change and very simple to set up.

See a slightly different version of the Lawnbot400 in my new book "Arduino Robotics" , as well as a DIY Segway and several other bots.

Check it out in MAKE magazine in the April 2010 issue (#22) or here:

UPDATE 3-24-10

New wheel-barrow bucket mounted on top with hinges so it can dump its contents.

UPDATE 3-10-10: NEW CODE

And new video of the Lawnbot400 moving a bunch of dirt from my truck to the flower beds across the yard, also I updated the code again.

.

I added some new code to the project that is safer, including a manual kill-switch and a Failsafe switch.

To implement the Failsafe, I used another Atmega168 (or an Arduino), to control a normally-open 60amp power relay. The relay disconnects the power to the motor-controller unless receiving a "good" signal from the 2nd microcontroller. This signal is updated 2 times every second and is either ON or OFF. If the bot gets out of range, it loses power to the motors. If I flip the kill-switch on the Transmitter, it loses power to the motors. This is also a handy way to disable it remotely if anything were to go near it that wasn't supposed to. The updated code for both microcontrollers is on the CODE page.

In addition to the failsafe, I changed the way the code reads the PPM signals to make it more reliable. Also, I realized that I was only able to run the bot at 80% speed with the old code, so now it is quite a bit faster and has more power (it can carry me across the yard @ 155lb).

Check out this new video of me riding the Lawnbot400, my wife driving it over a bunch of branches, then me making do some wheelies. Don't worry, the mower was turned off this time since the grass didn't need cutting, we were just having fun.

Disclaimer:
DANGER!!! This is a VERY dangerous piece of equipment if not handled appropriately. Since all the electronics have been home-built and the Arduino code is new, you MUST be very careful while operating anything heavy with this code. I have had 1 or 2 times during testing - and before adding a secondary failsafe - that the main Arduino jammed up and I temporarily lost control of the mower for a few seconds!!!! Though I have added several filters to discard unwanted signals and I rarely have any issues, an un-manned lawnmower IS STILL A POTENTIAL DEATH TRAP and I assume no responsibility for anything that happens as a result of your use of this code or this tutorial. This is meant as a guide for people who not only have the ability to build such a contraption, but the responsibiltity to operate it safely as well. Any suggestions or ideas on how to make this a safer project is always gladly accepted. Having said that, it's also awesome.

Background:

Most R/C equipment comes packaged for a single specific use, which makes it easy to use but is very limited in what you can do with it. So using the Arduino as an interpreter between the R/C system and the motor driver, I can use any motor controller that I want (depending on the size of the motor and power required), reprogramming the Arduino to supply the required signals.

What I ended up with:

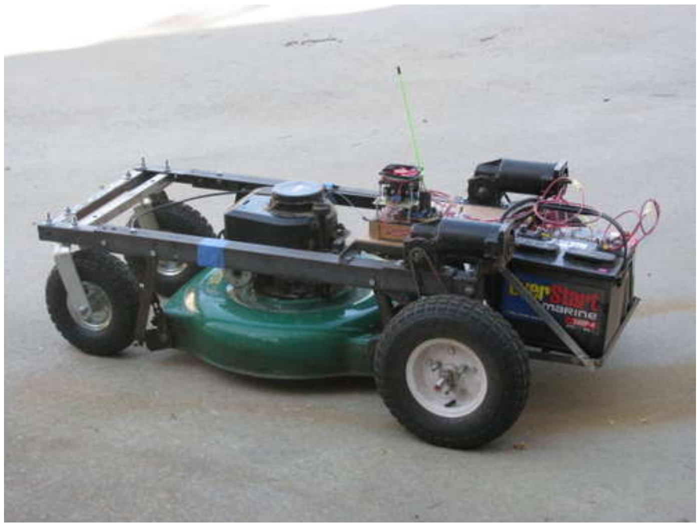
After successfully hacking a few R/C cars from the thrift store, I got bored driving them around the driveway and I was having a hard time convincing my wife that there was any usefulness in the revived toy car. So I decided it was time to make my biggest chore at home, a whole lot easier and actually put my Arduino to work, and thats how I ended up building an R/C lawnmower.

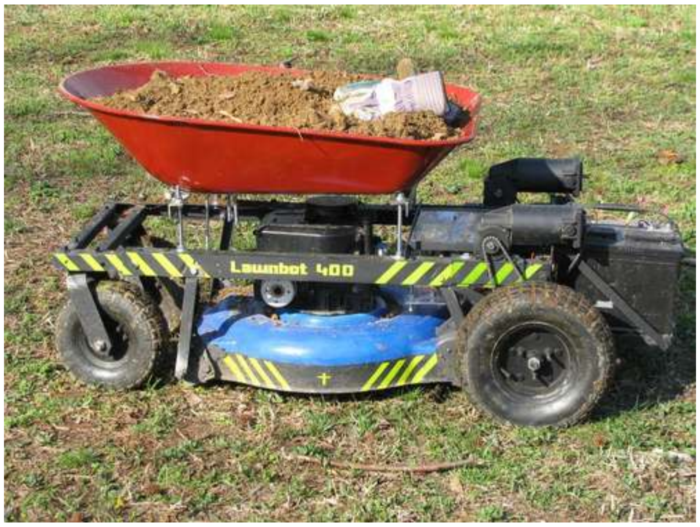
While designing the lawnmower, I thought it would be cool to learn about the electronics that made it move, so I designed and built my own motor speed controller (or H-bridge) to power the lawnmower. I looked around at every H-bridge design I could find before deciding to go with a Mosfet h-bridge that uses both N-channel and P-channel Mosfets.

I built several different motor driver boards for this project, the first two were on Radio-Shack perf-board and the next 4 were designed using EagleCad and etched to a piece of copper-clad PCB, using the toner-transfer method. The most recent board is the one I use to mow the lawn as it has the ability to stay cool even while operating for long periods of time (30-40 mins straight) at 10-20amps and 24vdc. FWIW, I had to burn up a lot of Mosfets to find this out. If you want to see any of my other motor controllers, go to www.rediculouslygoodlooking.com and check out the Mosfet shield.

Here is what I bought already assembled:
FM R/C transmitter and receiver pair from ebay = $40
Arduino = $30
I already had a used push-mower = $60

Here is what I bought and assembled into the Lawnbot400 (as I call it):
(2) electric-wheelchair motors from ebay = $40 ea
(2) 12v marine deep cycle batteries - Walmart - $60 ea new (used batteries might work)
36" pieces of 2" angle-iron (2) and 1" square-tubing (2) from Home Depot = $8 ea
36" pieces of 1" angle-iron (2) and 1" flat steel bar (2) from Home Depot = $5 ea
(a lot) of nuts, bolts, washers, lock washers 3/8" or 1/2" with drill bit = $20
(2) caster wheels from Harbor Freight Tools = $14 ea
(2) drive wheels from Harbor Freight Tools = $8 ea
(36") 5/8" threaded rod with several 5/8" nuts and washers from Home Depot = $8
(2) sprockets from Allelectronics = $5 ea
#25 roller chain and a few universal links from Allelectronics = $10 for 3'
sprockets from Electronics Goldmine = $1.50 ea
(24) mosfets from Digikey = $1 ea
(there were quite a few small parts for building the H-bridge, they are listed later on)

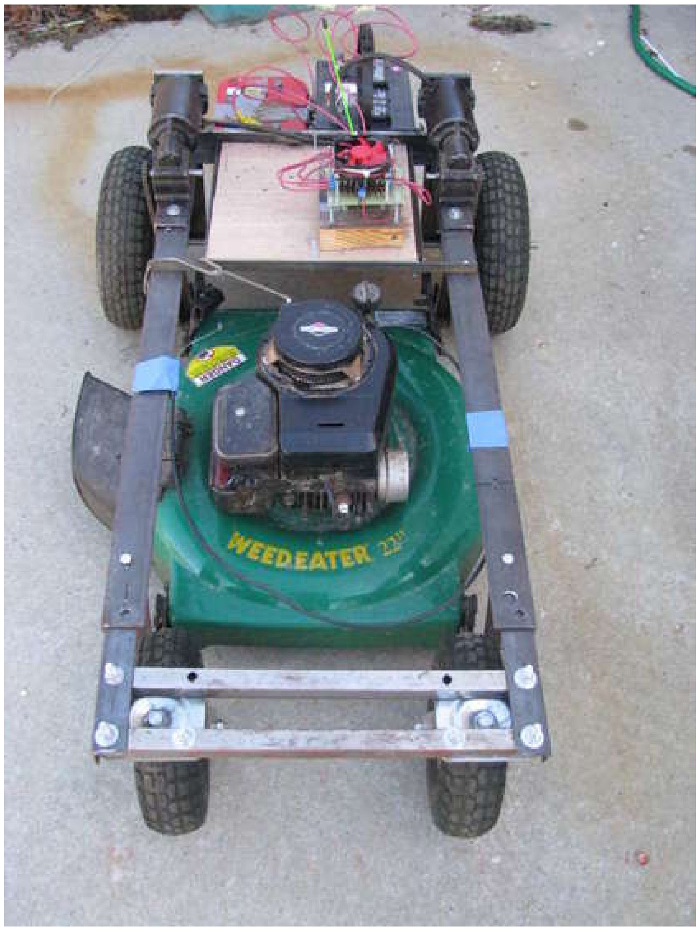




Lawnbot 400

Lawnbot 400

Lawnbot 400



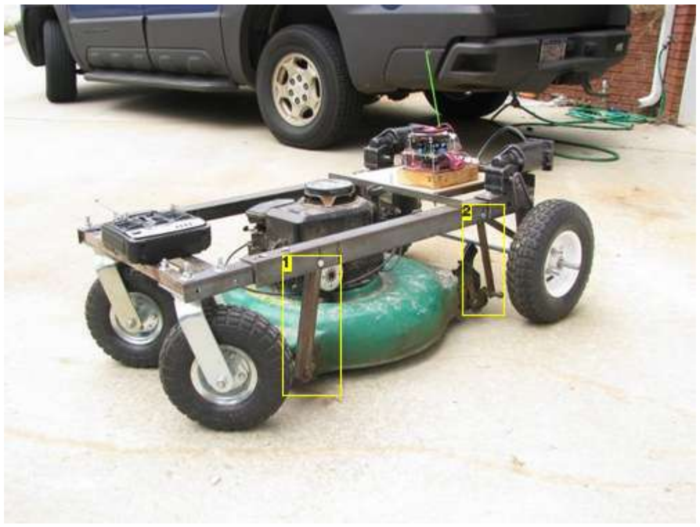


Image Notes
1. the front left mower deck hanger
2. the rear left mower deck hanger

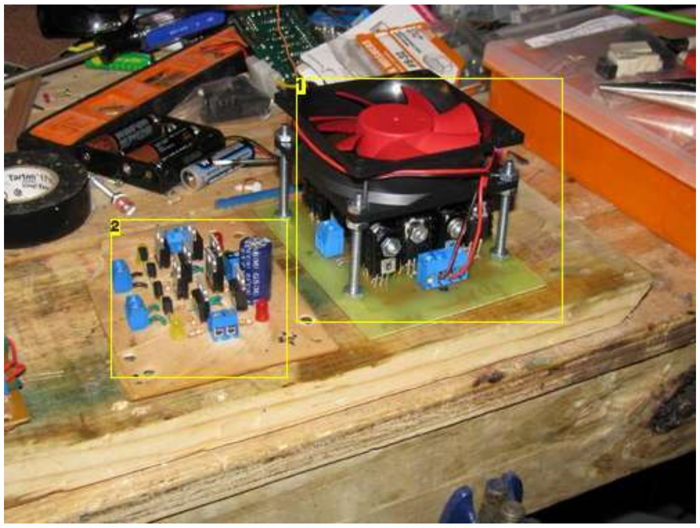


Image Notes
1. the Triple8 motor controller with 24 mosfets, each set of 3 is bolted together and each mosfet is heatsinked. It has 3x as many Mosfets as it's little brother, but essentially the same circuit.
2. the predecessor to the Triple8, only 8 mosfets total (just enough to complete a dual h-bridge). Though it would run the Lawnbot400 around for about 10 minutes, it would end up getting hot after some use.

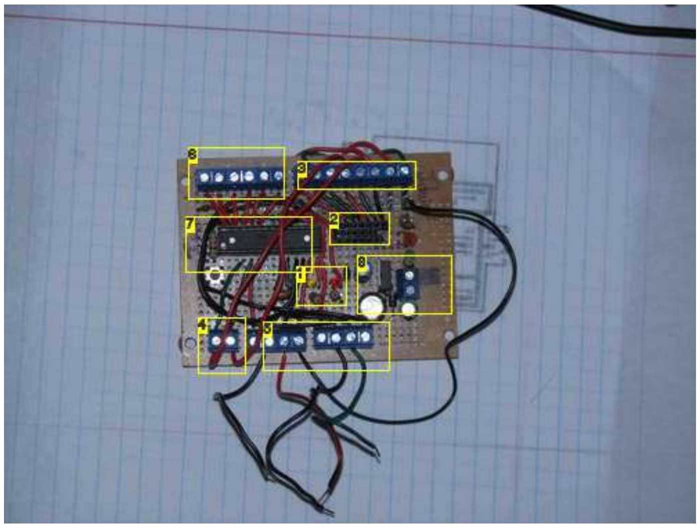


Image Notes
1. the 2 neutral indicator LED's (1 red and 1 yellow) hard-wired to digital pins 12 and 13. Anytime I center one of the control sticks on the lawnbot400, one of these lights turns on.
2. the female headers used to plug my R/C receiver directly onto (they use standard .1" spacing like perfboard you can buy at Radio Shack)
3. these are the breakout screw-terminals used to route the R/C receiver signals to the Atmega168. I am only using 2 of the 6 R/C channels right now, so the other 4 can be used for extra servo's or whatever else.
4. digital pins 2 and 3 of the Atmega168, used for the External Interrupts on those pins to capture the R/C signals from the receiver.
5. These are the screw-terminals for the signal wires leading to the H-bridge motor controller. I only need 4 wires to run my motor controller, but there are 3 extra digital pins that are unused by the current code.... Any ideas for their use?
6. all 6 analog pins are unused! I might add some sensors to automate the Lawnbot400 one day.
7. the Atmega168, it's reset button, and a kind-of hidden 16mHz crystal oscillator (together make a bare-bones Arduino).
8. 5-35v power terminal and onboard 5v regulator for powering the Atmega and R/C receiver. Plus a bunch of capacitors and a reverse polarity protection diode.

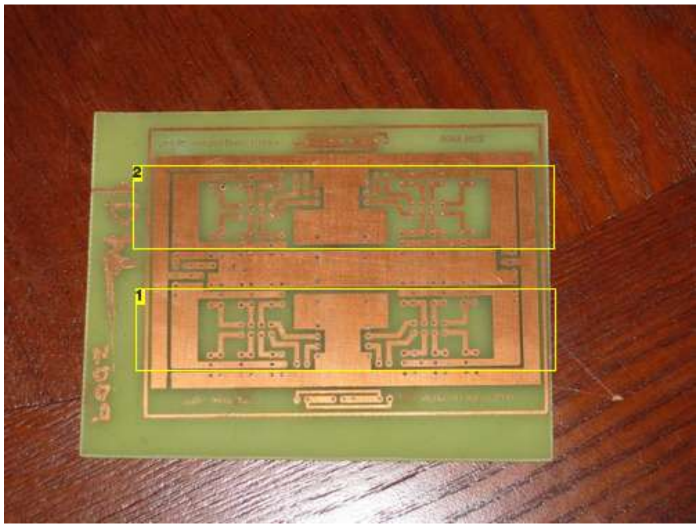


Image Notes
1. this is the 2nd H-bridge, notice that the motor screw-terminals for each motor will be on opposite sides of the board.
2. This is the 1st H-bridge

Step 1: Setting up

1. Get R/C transmitter and receiver (I have tested FM and AM systems and they both work)
2. Upload code to Arduino (it is on the last page)
3. Make sure you are getting a good signal

You will need an R/C radio transmitter(Tx) and receiver(Rx) pair, which is the most expensive part of the project, but can be used for every future project you might have involving R/C. I went with a 6-channel FM system, but I have tested a 27mHz AM transmitter/receiver and it works just as well. The beauty of the Arduino is that if you want to adjust the deadband or the motor-speed at turn-on, (unlike commercial ESC's) it is all easy changed in the Arduino IDE.

Once you have your radio, all you need to do is upload the code to your Arduino, plug in the 2 channels that you want to use from your radio receiver into Digital pins 2 and 3 of the Arduino (these are the 2 external interrupt pins on the Arduino) and you are ready to control whatever you want. If you don't have a batter pack for the receiver, you can run jumper wires from the Arduino +5v and GND to the R/C receiever for power, you only need to supply a single channel with GND and +5v (it is not necessary to power every channel).

Upload the code using the Aruino IDE (I am using version 0016 on Ubuntu).

I started by controlling 3 LED's with 1 channel on a breadboard. I wired a red LED to be Forward (digital pin 9), a yellow LED for Reverse(digital pin 5), and a green LED for Neutral (digital pin 12). This allows you to adjust the code to fit the needs of your radio system. You will have smooth 0-100% PWM control of both LED's and the neutral light will turn on when the control stick is centered. If needed, you can widen the deadband for Neutral, but doing so will increase the speed at turn-on (which starts at 0%, so that would likely be desirable). See pictures.

The code has 4 PWM outputs for motor control:

channel 1 Forward = Arduino digital pin 9
channel 1 Reverse = Arduino digital pin 5
channel 2 Forward = Arduino digital pin 10
channel 2 Reverse = Arduino digital pin 6

2 outputs for Neutral indicator lights:

channel 1 = digital pin 12
channel 2 = digital pin 13

The 2 INPUTS from the R/C receiver should go to:

channel 1 = digital pin 2
channel 2 = digital pin 3

If you are interested to see your readings, turn on your Serial Monitor in the Arduino IDE (set to 9600bps) and you can see the actual real-time pulse readings for each channel, they should read:

full forward = 2000 (2 milliseconds)
center = 1500 (1.5 ms)
full reverse = 1000 (1 ms)

These readings reflect the number of microseconds that the pulse signal from the R/C receiver stays HIGH (or at 5v). The typical Servo signal that comes from an R/C receiver is a pulse whose length varies from approximately 1 ms to 2 ms with 1.5 ms being Neutral (which should also be the position that the control stick returns to when you let it go). The transmitter reads the position of the control stick and sends that pulse length about once every 20milliseconds. So it is constantly updating for precise control (for more info, look up PPM on wikipedia). If you push the transmitter control stick forward, the reading should go up to 2000, if you push it backward it should go down to 1000. You can also use a voltage meter at this point to see that Digital Pins 5, 6, 9, & 10 will be changing from 0-5v depending on the position of the control sticks on the R/C transmitter.

If you care to know, the code uses the Arduino's 2 external interrupts to capture when the Rx signal pin changes states (goes from HIGH to LOW or vice versa), when it does at the beginning of each signal, it calls the interrupt function which reads the digital state of the pin and if HIGH, it records the microseconds value on the Arduino system timer0. It then returns to the loop until the pin goes LOW, at which point it subtracts the previously recorded microsecond value from the new current microsecond value to determine how long the pulse stayed HIGH (which tells us the position of the Transmitter control stick). It then does that over and over really fast.

I have the values constrained from 600-2400 in the Arduino code to keep things simple. Once it receives the signal and constrains it, it maps that value to be proportionally between 0 and 511, where 255 will be Neutral. The code then determines when the value changes and uses a function to determine the appropriate 0-255 PWM value in the appropriate direction and each direction has it's own PWM output pin to control the H-bridge.

On a side note:

To make things easier, I built an Arduino-based breakout board using Radio-Shack perf-board, a 28pin DIP socket, a 16mhz oscillator, and a bit of wire. I also added a set of female-headers in such a way that I can plug my R/C receiver directly onto the breakout board. For secure connections while mowing grass, I added screw-terminals on each Output pin and each of the 6 channels from the receiver. It also has a built in 5v regulator to power both the Atmega168 from the Arduino and the R/C receiver (which gets power when you plug it onto the breakout board). So you just route jumper wires from the channels you want to use on the receiver, to the Atmega digital pins 2 and 3. I also added 2 LED lights that are hard wired to the digital pins 12 and 13 for the Neutral lights for each channel so I can easily see when I am in neutral.

Since this bot is a Tank steer setup with 1 drive motor on each wheel, the coding is very straightforward where the left stick controls the left motor and the right stick controls the right motor. Both sticks forward means lawnmower goes straight forward, both backward and it goes in reverse. If you push the left forward and the right backward, it does a zero-turn circle. As you can imagine, mowing the grass is really fun now.

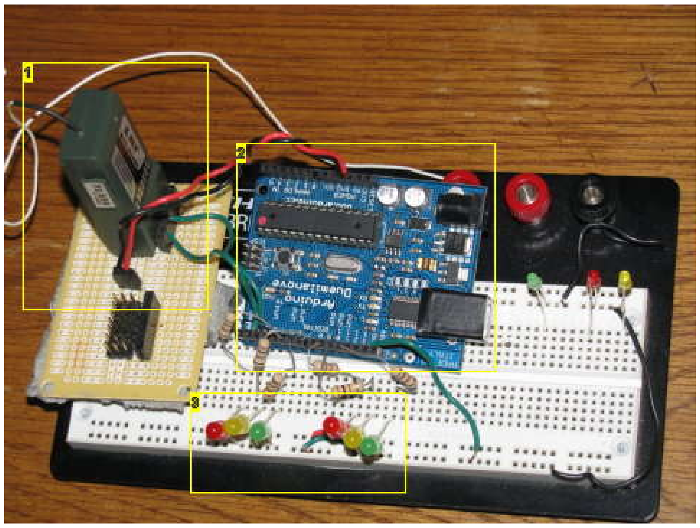


Image Notes
1. this is my receiver plugged into a breakout board I made for it using perfboard.
2. the Arduino receiving R/C servo signals and translating them into forward/reverse PWM values.
3. each set of LED's is controlled by it's own channel from the R/C receiver. Forward will turn on the green light, reverse the Red light, and neutral will light up the Yellow light. This is the easiest way to test the setup.

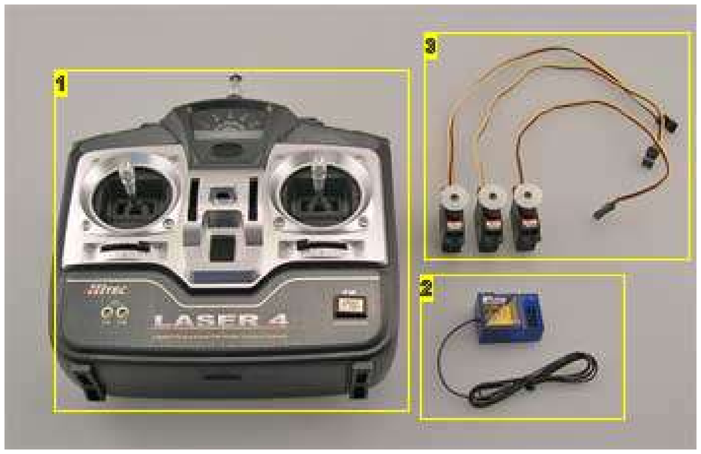


Image Notes
1. this is a typical R/C transmitter with 4 channels, the one I got is a knockoff of this one, but looks very similar.
2. this is a typical R/C receiver. Mine has it's connector pins on the end of the unit instead of the top, enabling me to plug my receiver directly onto the control board.
3. these are typical servo motors. They can be controlled directly by the R/C receiver and are useful for many things.

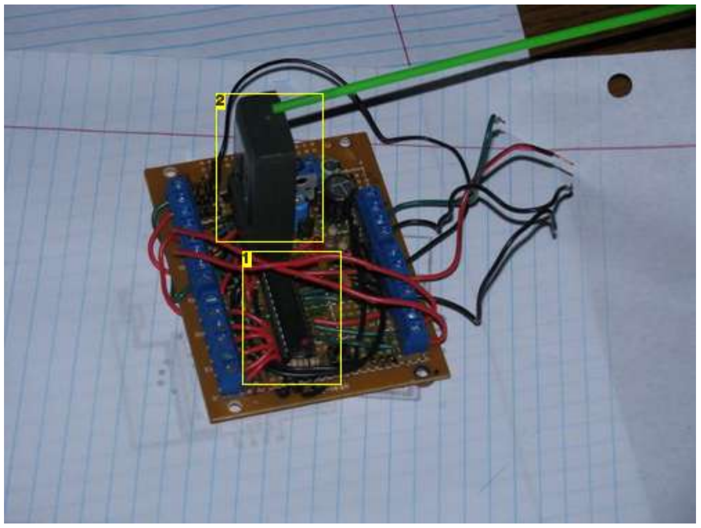


Image Notes
1. the Atmega168 from my Arduino (I bought a few extras to use for projects like this). I remove it when I need to re-program it in the Arduino.
2. my R/C receiver plugged into the control board. Notice the green antenna coming out.

Step 2: The Motor Driver

I built several motor drivers before finding a design that worked for my needs. For what it's worth, there are several nice products already out there that are fully assembled and require a lot less work if you are not interested in building your own electronics. The Open Source Motor Controller is an open source design that has been under constant community improvement for several years now and can handle up to 160amps at 36vdc! But they are over $100 and only control 1 motor. The Sabertooth 2x25amp motor controller is nice and controls 2 motors, but it is $125.

So I thought I would just make an extremely simple dual h-bridge that could handle at least 25 amps at 24vdc continuous and handle surges of up to 100amps for a few seconds. Once I found out that you can parallel Mosfets and multiply their current carrying capacity accordingly, I thought I would come up with a simple design and slightly complicate it by adding more mosfets until I had enough to handle the current that I needed. Digikey has a good selection of Mosfets to choose from and good filters to narrow it down by what you need, so I spent a lot of time looking for Mosfets that were rated for around 50amp and could handle over 30 volts. Also, they have to be cheap because my plan is to use a bunch of them. I decided on the FQP47P06 p-channel and the FQP50N06L n-channel Mosfets from Fairchild Semiconductor, which I bought from Digikey.

If you are wondering what an H-bridge is, find out here: en.wikipedia.org/wiki/H-bridge and this will all make more sense to you.

The design is simple: 2 P-channel mosfets control the high-side switches and 2 N-channel mosfets for the low-side switches. But instead of using 1 mosfet for each switch, lets use 3. Now we have 12 mosfets per H-bridge (3 mosfets x 4 switches) and theoretically the ability to carry 150 amps (that is not accurate though). The board is as small as I could make it with nothing touching. Each set of 3 mosfets have heatsinks and are bolted together to help dissipate heat. Also, there is an 80mm cooling fan mounted directly above mosfets to further keep them cool. The mosfets are very good at handling sudden changes in direction and speed changes.

Since there are 24 mosfets in total (8 groups of 3) I dubbed it the Triple-8. It is running at the Arduino default PWM frequency of 1kHz (I plan on playing with that to get the frequency higher). The board has 4 inputs, 2 for each bridge. If you bring an input HIGH, that side of the bridge goes HIGH.

Ideally, you would control the board by holding 1 input LOW and applying a PWM signal to the other input. This allows for easy speed control. I have written into the code that if you bring digital pin 7 HIGH, the code switches to Relay mode and either turns the mosfets all the way ON or all the way OFF. This is far more difficult to control, but is useful sometimes.

If you are interested in building your own H-bridge you can download the eagle file to etch a pcb and the schematic to show where everything goes. You can get everything to make this dual h-bridge at Radio-shack (including the copper clad), except the Mosfets and a special resistor network I used to save space. I bought most of the parts from Digikey though because it was cheaper and arrives to my house in 2 days.

Here are the parts needed for this motor driver:

(12) FQP47P06 - P-channel mosfet 47a 60v - Digikey - $1.73 ea
(12) FQP50N06L - Logic level N-channel mosfet 52a 60v - Digikey - $1.04 ea
(4) 2n7000 - Logic level N-channel mosfet 200ma 60v - Digikey - $0.26 ea
(8) 4606X-1-470LF-ND - 47ohm bussed resistor network - Digikey - $0.25 ea
(6) ED1609-ND - 2 position screw terminal - Digikey or Radio Shack- $0.46 ea
(24) CF1/84.7KJRCT-ND - 4.7k 1/8w resistor - Digikey or Radio Shack - $1.78 (for 50pk)
(1) PC9-ND - 3"x4.5" 1-sided copper-clad .064" 2oz copper - Digikey or Radio Shack- $4.66
(4) P5575-ND - 1000uf Capacitor or similar - Digikey - $1.19 ea
(1) 330ohm - 1kohm resistor 1/4w - for power LED, doesn't have to be exact
(1) power LED any color you like, I use the 3mm size to save space

Maybe something smaller?

If you are going to use this for something smaller than a 100lb lawnmower, you can look up one of the many H-bridge circuits and build your own smaller motor controller with as few as 4 mosfets (or BJT transistors) or even use a packaged IC H-bridge like the l293d (dual 1 amp) or the l298n (dual 2 amp).

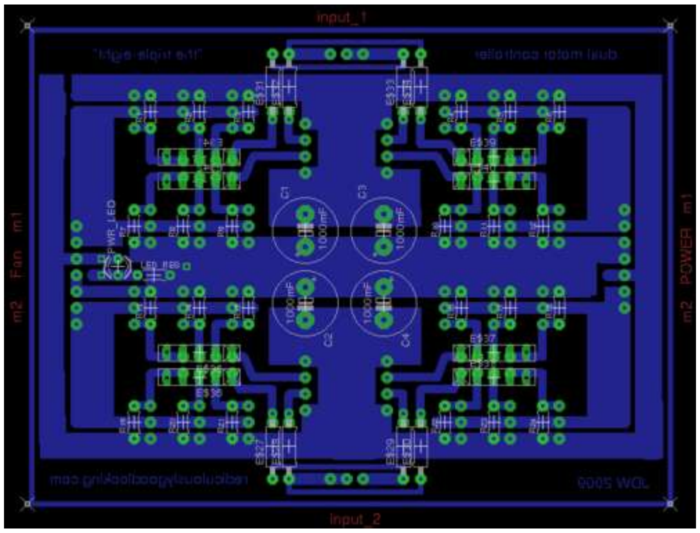
Or if anyone is interested, I will post a schematic and Eagle .brd file for a smaller version of this H-bridge that only requires 8 mosfets total (everything else is the same), and it can handle about 10amps at 24vdc.

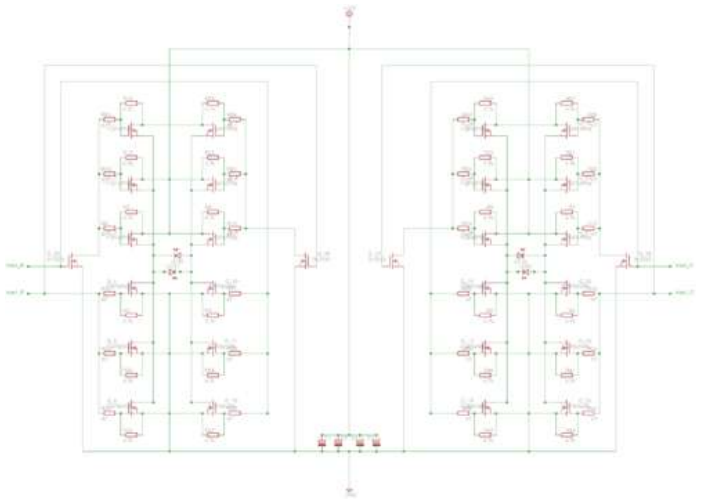
Etching:

I am not going to go into all the details of PCB etching, because there are already many excellent instructables on that topic. So once you download my .BRD file of my motor controller, all you need to do is print the .brd file onto some magazine paper using a laser printer, and iron that onto a piece of clean copper-clad. Then etch it with your favorite etchant solution (I use 2 parts Hydrogen Peroxide to 1 part Muriatic Acid and it works perfectly). And remove the toner with Acetone when done etching.

For ease of assembly I designed this board to be Single-sided and to use only through-hole components, no surface-mount stuff to mess with! Yay for you.

You can get the .brd files for the various h-bridges at www.rediculouslygoodlooking.com





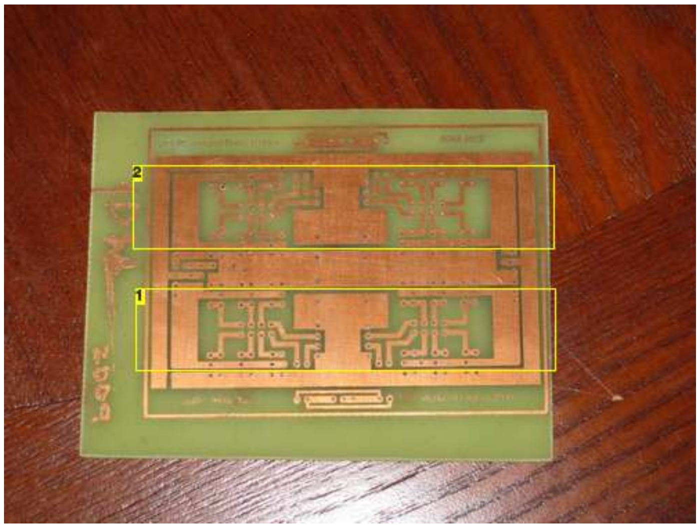


Image Notes
1. this is the 2nd H-bridge, notice that the motor screw-terminals for each motor will be on opposite sides of the board.
2. This is the 1st H-bridge

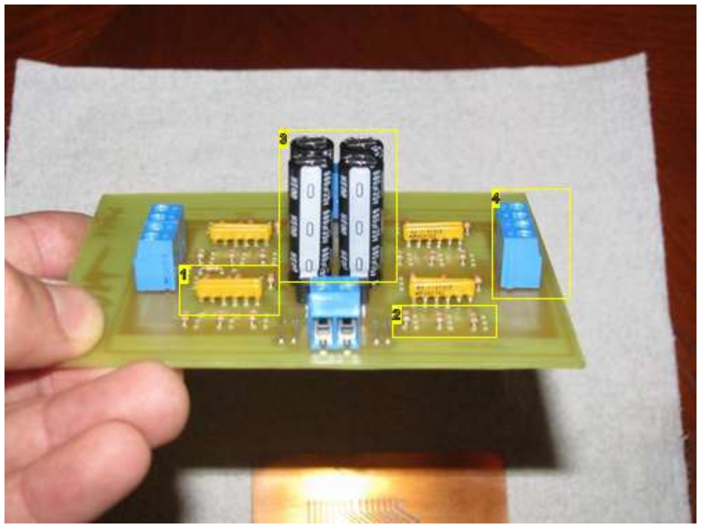


Image Notes
1. bussed resistor networks 47ohm. They have 1 input and 5 outputs, this board only uses 3 of the outputs.
2. pull up/down resistors 4.7k ohm, these keep the Mosfets turned off when not being used.
3. capacitors, I used (4) 680uF 50v, but you can substitute others that fit.
4. screw terminal connectors for motor terminals and power

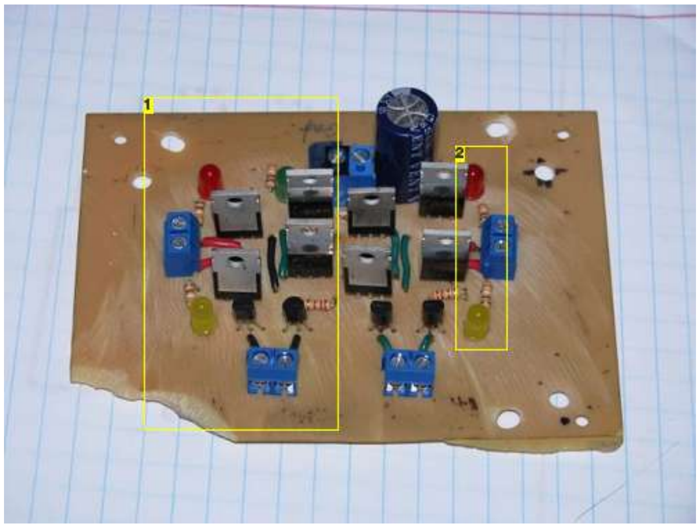


Image Notes
1. this is 1 complete h-bridge to control 1 DC motor. The 2 smaller mosfets toward the bottom are used as signal-inverters to control the High-side p-channel mosfets.
2. each h-bridge has it's own set of direction lights to determine the direction of the current.

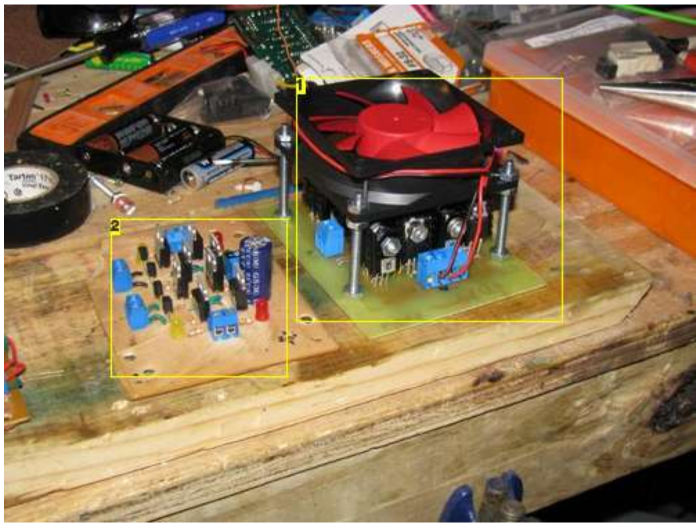


Image Notes
1. the Triple8 motor controller with 24 mosfets, each set of 3 is bolted together and each mosfet is heatsinked. It has 3x as many Mosfets as it's little brother, but essentially the same circuit.
2. the predecessor to the Triple8, only 8 mosfets total (just enough to complete a dual h-bridge). Though it would run the Lawnbot400 around for about 10 minutes, it would end up getting hot after some use.

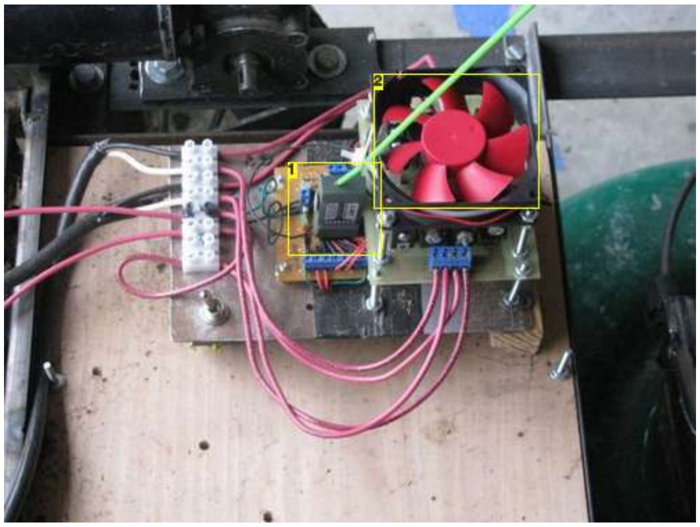
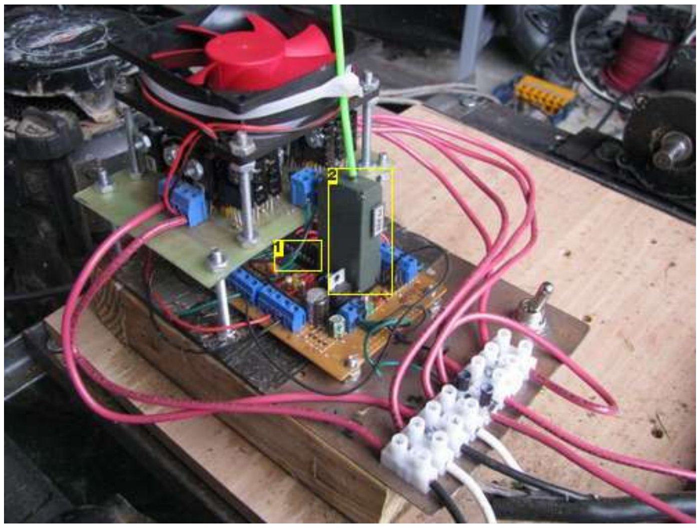


Image Notes
1. R/C receiver plugged into Arduino breakout board
2. cooling fan for motor controller (h-bridge)



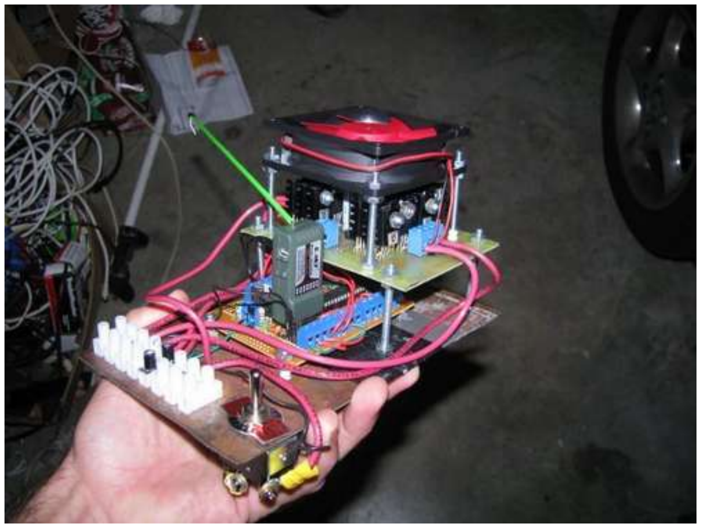
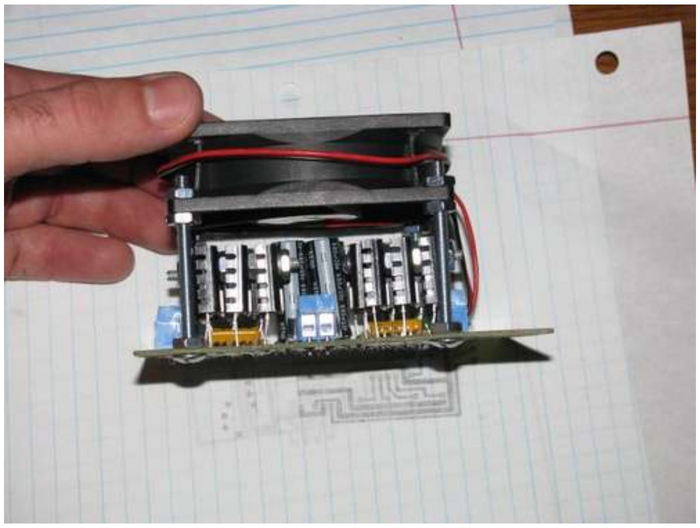
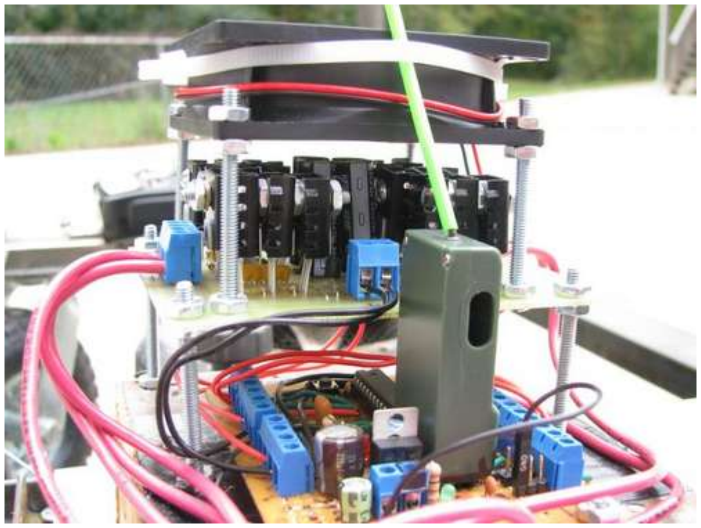


Image Notes
1. Atmega168 microcontroller programmed in the Arduino, then transferred to this home-made breakout board for permanent use.
2. The R/C receiver is plugged directly onto my home-made breakout board which supplies the +5v and GND needed for power as well as a breakout screw-terminal for each channel. This receives the signals from the remote-control (R/C transmitter) and sends them into the Atmega168 for processing.





Step 3: The Wheels

First you need to mount the drive sprockets to the wheels.

The EASY way:
If you are smart and have more money, you can find a set of wheelchair motors that have the wheels mounted to them.

The CHEAP way:
I could not find any in my price range, so I went with just the motors, then bought wheels, then sprockets. Believing it would not be strong enough to mount the wheels directly to the motors, I opted to mount the drive wheels on an axle, then the motors to the frame, and use chain to transmit the power. A picture is worth 1000 words, so look at them carefully.

Mount the sprockets to the wheels:

I had to place the sprocket on the center of the wheel and drill 3 holes through the sprocket and then through the wheel itself. Once the sprocket is lined up and properly centered, I placed the 3 bolts through the sprocket and wheel and tightened them up as much as possible. I then welded the sprocket to the wheel hub to keep it centered.

The wheels from Harbor Freight Tools have built in bearings for a 5/8" shaft, hence the 5/8" threaded-rod we are going to use as an axle.

Repeat this process for both wheels.

There is more detailed info tagged in the pictures.

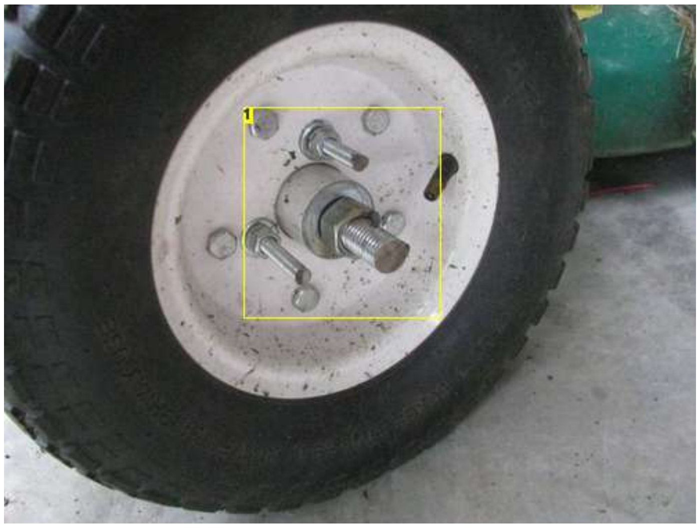


Image Notes
1. The bolts coming from around the axle are the 3 bolts that hold the sprocket onto the other side.

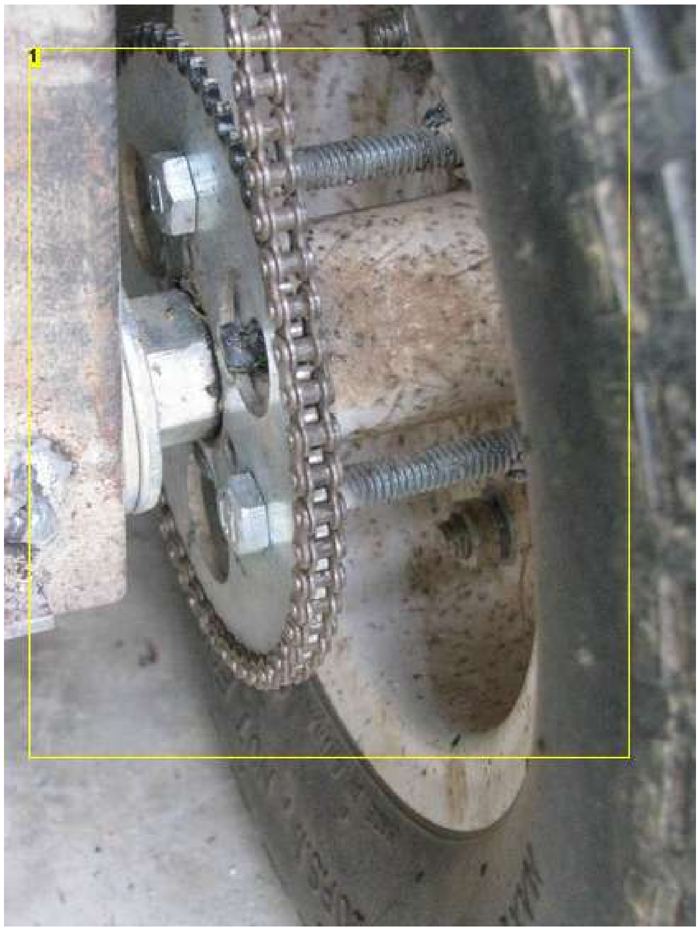


Image Notes
1. The drive sprockets are about 6.5" in diameter and had no holes to mount them. I had to drill 3 holes and mount bolts through the sprocket into the wheel. I then added a small bead of weld to keep it centered around the axle.

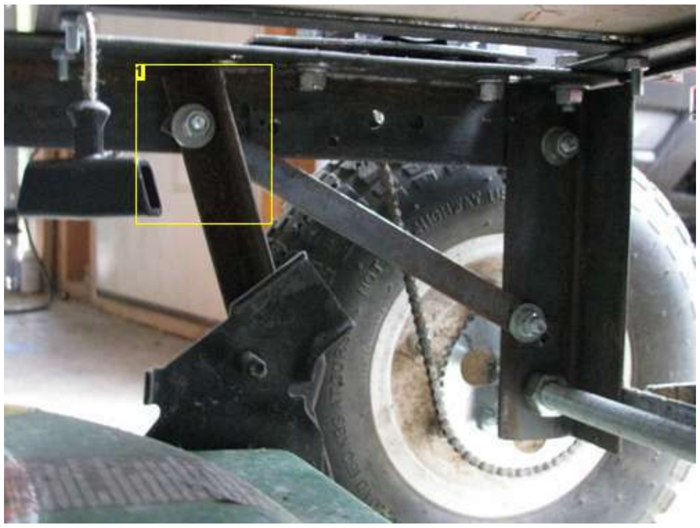


Image Notes
1. save a bolt on each side by using the same one that you used to bolt the frame riser brace into the frame.

Step 4: The Frame part A

This is the difficult part to explain. You will likely have to have some mechanical ability and a good set of tools to build a large metal frame from scratch. And since this was a prototype, the dimensions are not all perfect, but luckily they don't need to be.

The frame will be custom measured for your particular lawnmower, so I won't be giving you exact measurements.

Tools needed to build a frame:
measuring tape
angle-grinder
ratchet set
crescent-wrench
a level
electric drill
bolts, nuts, washers, and lock washers of either 3/8" or 1/2" diameter and 3/4"- 2" long
drill bits the size of the bolts you are using
1" and 2" angle-iron (36" long pieces) you'll need both
1" square tubing (36" pieces, steel)
1" flat steel bar (36" long pieces)
the 4 wheels you got from Harbor Freight Tools (2 drive wheels and 2 caster wheels)
5/8" threaded rod (36" long) and several 5/8" nuts/washers

First you need to plan out the frame of your bot. Since I was attaching a lawnmower, I started by measuring the height that the lawnmower stood off the ground and took some basic measurements to see how big the frame needed to be. My frame turned out to be about 24" wide (this distance must match the width from the center of the rear lawnmower wheels) and 48" long (long enough for the front caster wheels to swing 360 degrees without hitting the front of the mower deck) and about 18" tall. Since we want the height of the mower-deck to be adjustable, we are going to attach the mower to the frame by removing the lawnmower wheels and using angle-iron to suspend the mower-deck from the frame of the bot.

1. I started out by using 2 of the 36" pieces of angle-iron (2" wide) for the main part of the frame running long-ways.
2. Cut the rear-piece of angle-iron the width of the rear of the mower (this measurement will be from the center of the left-rear wheel to the center of the right-rear wheel).
3. Drill holes in the ends of the angle-iron and bolt the rear-piece to the adjacent pieces from step 1, making sure they are straight.
4. Cut two front-pieces using 1" square steel tubing, the same length as the rear. We need 2 in the front to bolt the caster wheels to.
5. Drill holes and bolt these 2 pieces to the front of the angle-iron from step 1. You have to measure the holes from the 2 front caster wheel's mounting plates and drill the pattern into the front square tubing bars. Then bolt the wheels through those holes onto the front of the frame.

I later added another set of 2" angle-iron bars to the front caster wheel assembly to make the length of the bot adjustable at the front (see pics)

Now we should have a rectangular frame with the front wheels attached.

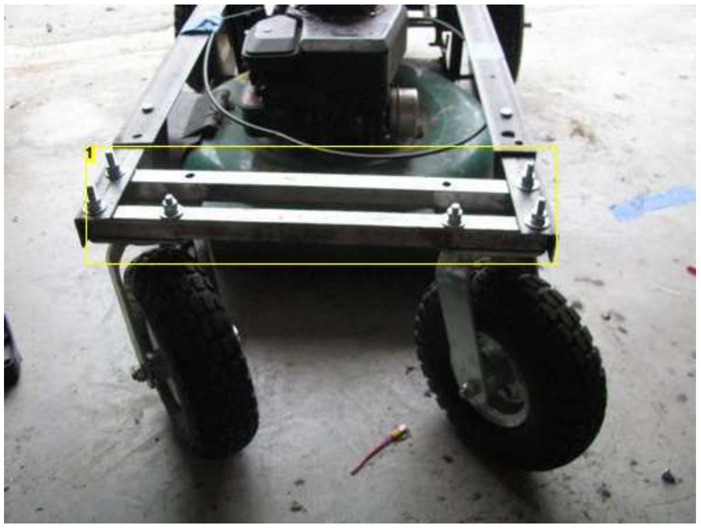


Image Notes
1. the front 1" steel square tubing that the front caster wheels attach to.

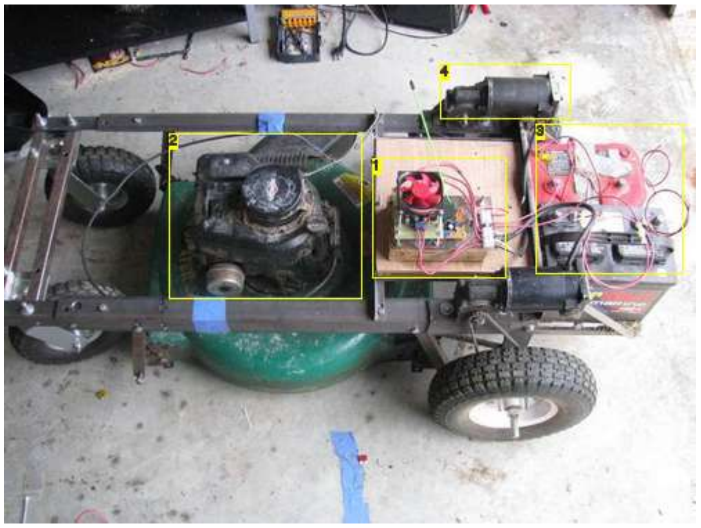


Image Notes
1. Motor controller and Arduino
2. push mower
3. (2) 12v batteries (deep cycle marine is the best)
4. electric wheel-chair motors

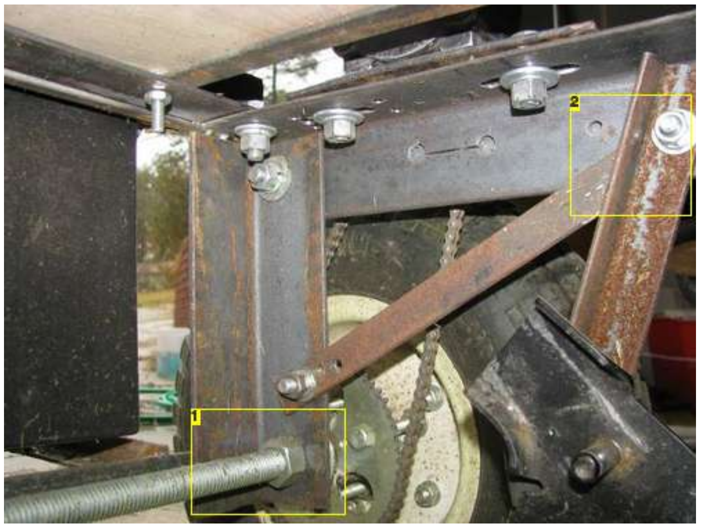


Image Notes
1. you need 1 nut on the inside of the frame riser bar to, and 1 on the outside to hold it securely to the axle.
2. I bolted the support bar in with the rear lawnmower-deck hangers to save a bolt on each side.

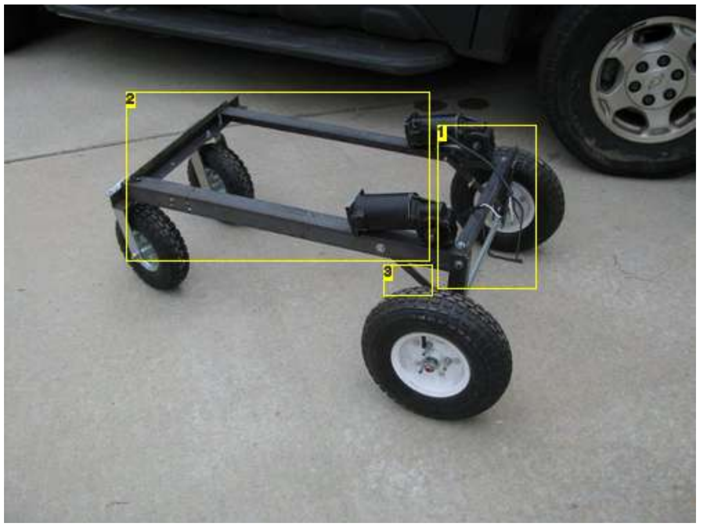


Image Notes
1. the rear bar should be the same width as the center of the rear wheels on your push-mower (must be measured before you remove the wheels).
2. the main frame bars.
3. the support brace

Image Notes
1. one of the main frame bars from step 1, which is 2" angle-iron.
2. the other main frame bar from step 1

Image Notes
1. the front left mower deck hanger
2. the rear left mower deck hanger

Step 5: The Frame part B

We now need to see how far down to mount the drive axle to make the frame level. So raise the rear of the frame up until the top of the frame is level with the ground (use your level). Now measure the distance from the top-rear of the frame to the ground, this is the frame height.

Now we need to take into account the height that the wheels will raise the axle off the ground. So measure the distance from the center of the rear drive wheel to the ground (the wheel's radius). Subtract the wheel radius from the frame height and we will have the correct distance from the top of the frame to the drive axle, which we will call the frame-riser height (we need to cut these pieces next). They are going to connect the rear of the frame down to the axle which the wheels will be mounted on.

6. We are going to add 2" to the frame-riser measurement (so we have a little to work with) and cut the 2 frame risers (mine were about 10-12" long).
7. Now drill (2) 5/8" holes, 1 at the bottom of each frame riser (about 1" from the bottom), this is where the drive axle will go through.
8. Drill 2 holes at the top and bolt the frame risers to the rear of the main-rectangular frame with the frame-risers pointed down.
9. Now feed the threaded-rod through the bottom holes of the frame risers and use 4 nuts to secure the frame risers to the drive axle (1 nut on each side of each frame riser, tightened down).
10. put the rear wheels on the axle and use 1 more nut on each wheel to secure them to the axle (these wheels have built in bearings). The sprockets should face inward toward the frame.

Now we should have a frame that stands on it's own with 4 wheels. However, the rear axle is not completely secure yet. We will need to add 2 braces from the bottom of the frame risers (near the axle) to the main part of the frame in order to keep the frame risers positioned properly. These braces can be flat steel and do not need to be very thick, they are just keeping the frame risers from moving.

Measure about 2" above each axle and drill a hole, then measure how far down that hole is from the top-rear of the frame and measure the same distance from the rear of the frame toward the front. Drill another hole on each side at this measurement. The support braces will need to be measured to be bolted in through these holes on each side (see pictures). The placement of the support braces is less important, meaning you can bolt them in wherever is convenient, as long as they are present.

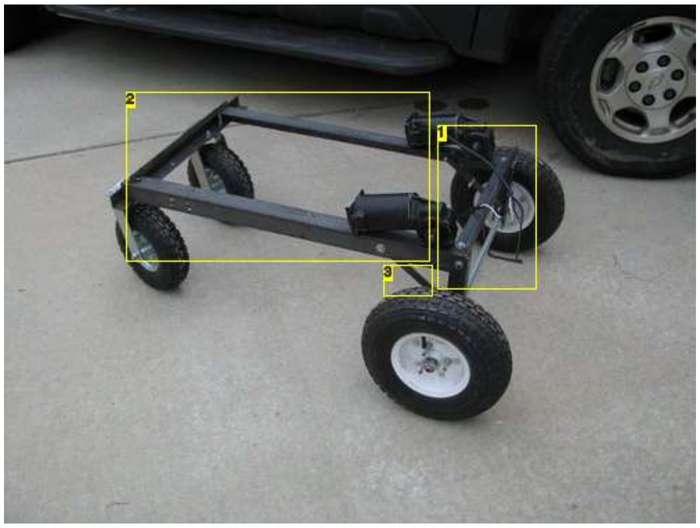


Image Notes
1. the rear bar should be the same width as the center of the rear wheels on your push-mower (must be measured before you remove the wheels).
2. the main frame bars.
3. the support brace

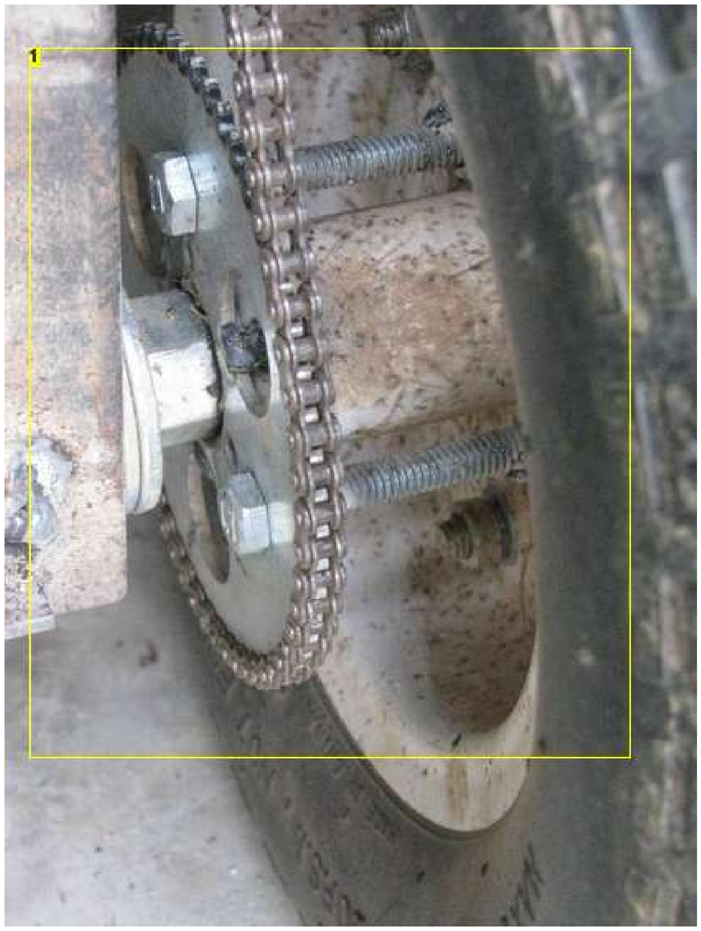


Image Notes
1. The drive sprockets are about 6.5" in diameter and had no holes to mount them. I had to drill 3 holes and mount bolts through the sprocket into the wheel. I then added a small bead of weld to keep it centered around the axle.

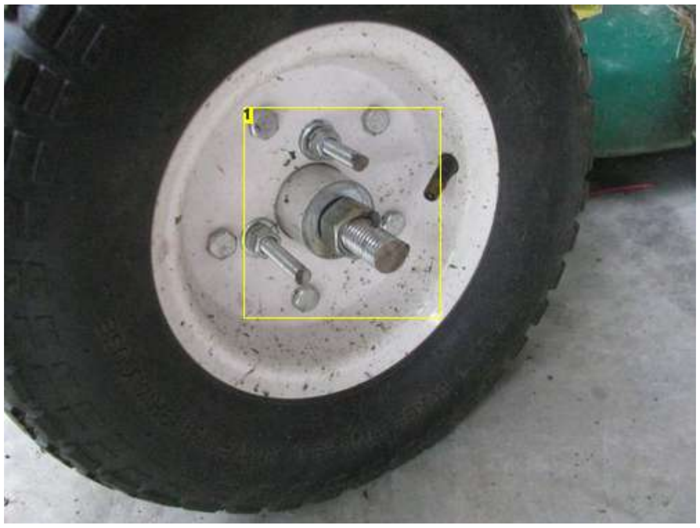


Image Notes
1. The bolts coming from around the axle are the 3 bolts that hold the sprocket onto the other side.

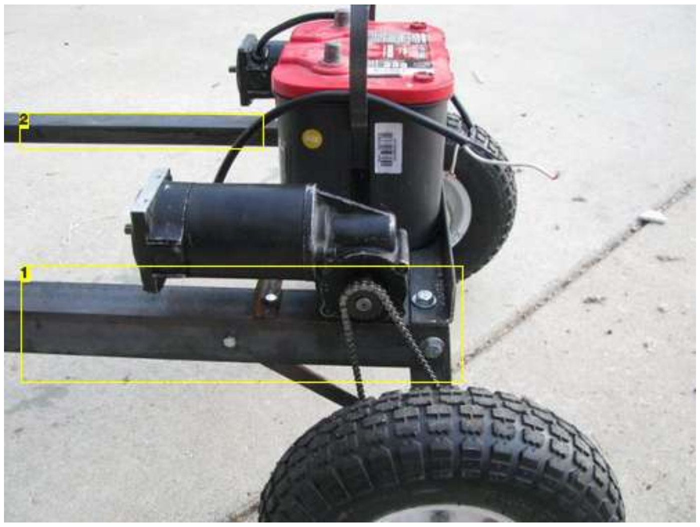


Image Notes
1. one of the main frame bars from step 1, which is 2" angle-iron.
2. the other main frame bar from step 1

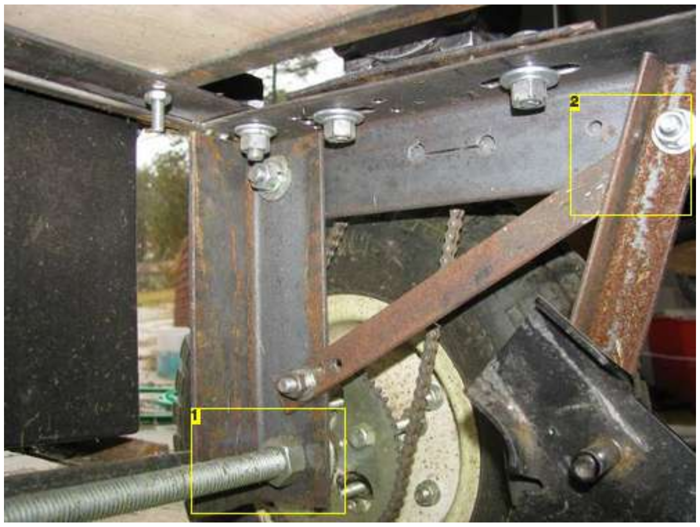


Image Notes
1. you need 1 nut on the inside of the frame riser bar to, and 1 on the outside to hold it securely to the axle.
2. I bolted the support bar in with the rear lawnmower-deck hangers to save a bolt on each side.

Step 6: Mounting the motors

This was the most difficult part to plan out on the frame. We need the motors to be adjustable so we can adjust the tension of the chain, however they just have 4 holes in the bottom of each motor and nobody makes a mounting plate that I could find.

The simplest way I could come up with was to mount the motors to an 8" long piece of 2" angle-iron, and then mount that piece of angle iron to the frame through some specially cut holes that allow the motor mount to travel forward and backward (but not side to side) along the frame.

Make the motor mount plate:

Cut an 8-10" section of 2" angle-iron, depending on how much room your motors need to mount. Mine only needed about 4", so I made it 8" to have plenty of room for the mounting bolts. Drill a hole about 1.5" from each end of the top of this bar, this is where the mounting bolts will go through the frame.

Mount the motor to the motor mounting plate:

Now you have to find the center of your motor mount plate (the 8" long piece of 2" angle iron) and measure the mounting holes on your DC motors. Use a sharpie marker to plot the hole pattern from the motor, centered onto the motor mount plate. My motors have (4) 1/4" diameter tapped holes in a rectangular pattern on the bottom of the gear box.

Drilling and cutting the adjustment holes on the frame:

Next you need to drill and cut the holes in the frame to let the motor mounting plate become adjustable. I cut these holes using a dremel tool and a cutoff wheel. You have to line up the motor mounting plate (with motor mounted preferrably) onto the frame rail and use a sharpie marker to mark where the holes will need to be on the frame rails. Start as far back as you can (without hitting any other bolts underneath the frame), and mark the center of each hole. Then move the motors forward 2" and mark the holes again. You want to cut the holes out of the frame so that the motor mount plate (with bolts going through the frame), can move forward or backward about 2". The holes in the frame are the width of the bolt and about 2" long. I drilled 1 hole at each end and used the dremel to cut out the rest.

The holes drilled in the motor mount plate are just single holes for the bolt to fit through, the holes through the frame were cut with a Dremel tool with a cutoff wheel to make channels for the motor mount bolts to travel forward/backward through. You want the 2" angle-iron motor mount bracket to set as much on top of the main frame rails as possible, the bolts (which you can't see with the motors mounted) that hold the motors to the motor mount plates will keep the motor mount plate from laying flat against the frame bars. Go ahead and mount the motors loosely to the frame using 2 bolts on each.

Cutting and connecting the chain:

Now get your 10' of #25 chain and wrap it around the main drive sprocket on the wheel. With the motors pushed all the way toward the back of the frame (closest to the drive wheel sprockets), wrap the chain around the motor drive sprocket and mark where they overlap. You need 2 of the universal chain links from to connect the 2 loose ends. Cut the 2 pieces of chain and connect them to each side with the universal links to connect them.

Tensioning the chain:

Push the motor mounts forward until there is good tension with the chain, and tighten up the bolts that hold the motor mount plates to the main frame.

Now you can generate electricity. Connect a voltage meter to 1 set of motor terminals and push the bot around.

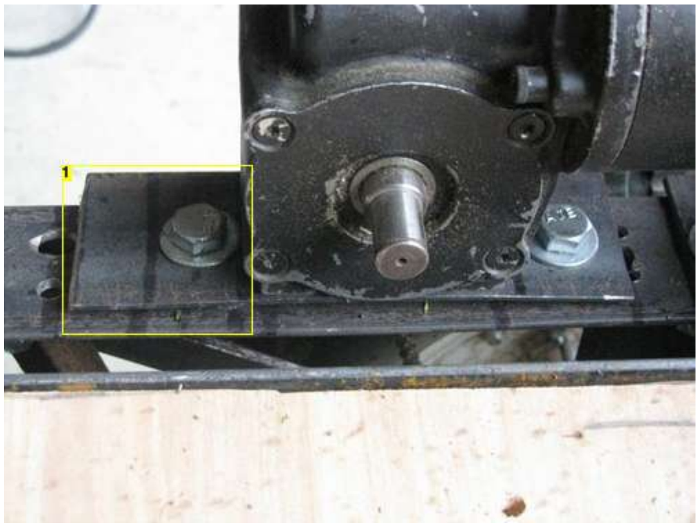
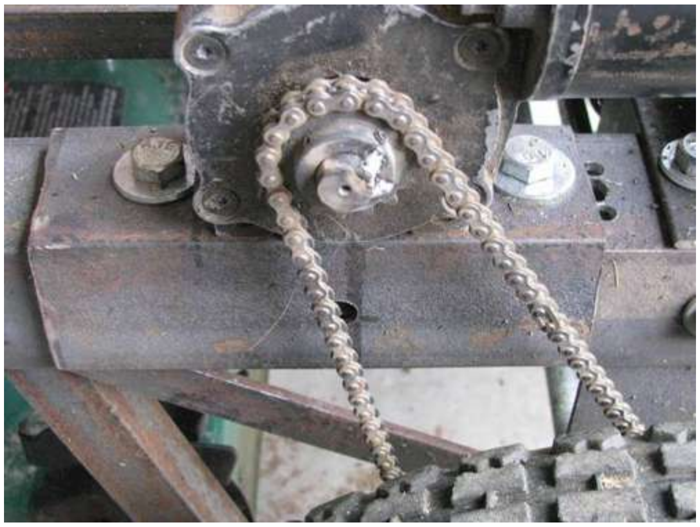


Image Notes
1. notice the motor is mounted to this piece of 2" angle-iron and that is mounted to the frame with these bolts. They allow the motor to slide forward/backwards on the frame when loosened.



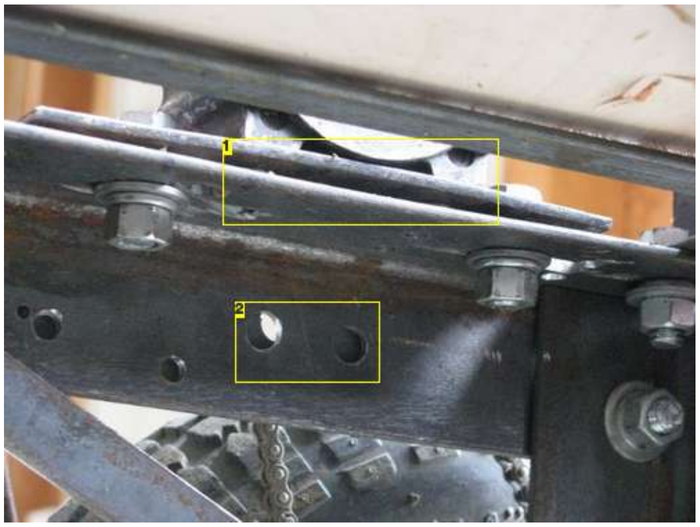


Image Notes
1. notice the gap between the motor mount plate and the main frame bar. This is caused by the bolts that hold the motor to the motor mount plate.
2. These are 2 unfinished holes for a 3rd mounting hole which I later deemed unnecessary.

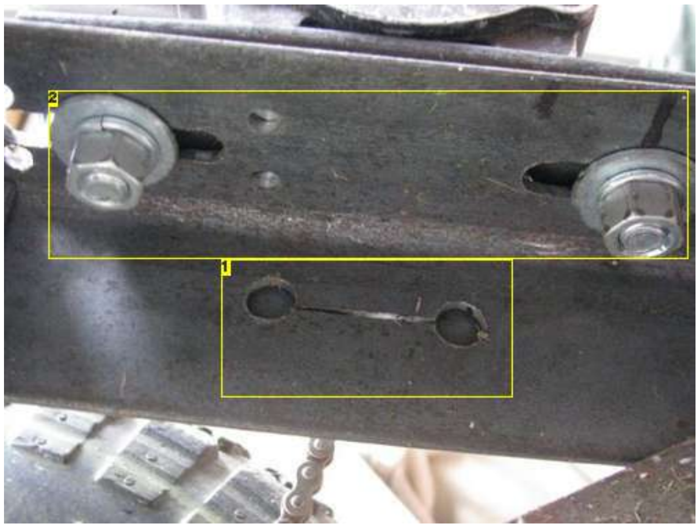


Image Notes
1. This is how to make the motor mount slide holes. Drill 2 holes where you want the ends of the track to be. Then use a Dremel with a cutoff wheel to cut a straight line between the tops and bottoms of each hole. They should end up looking like the ones above with bolts in them.
2. Tighten up these bolts when you get proper tension with the chain.

Step 7: Mounting the mower deck

Next we need to mount the mower deck to the frame. Remember we made the frame wide enough that the edges of the frame would be centered on the lawnmower wheel shafts.

All we have to do is cut 4 pieces of 1" angle-iron equal lengths so that the mower deck hangs evenly from the frame.

So measure the height of the frame from the top to the ground. Now measure how high the mower sits off the ground from the center of the wheel shafts (when the original wheels are on the lawnmower and all the height adjusters for each wheel are in the middle position). Now subtract the height the mower sits of the ground from the frame height, and cut 4 pieces of 1" angle iron to that length.

Now drill 1 hole in the end of each piece of angle-iron, about 1/2" from each end. The holes at the bottom will need to be the diameter of the lawnmower wheel shafts and the holes at the top will need to be bolted into the frame (hung at equal distances from the top of the frame).

Once you have all 4 hangers installed, you can install the mower deck and tighten up the bolts. Make sure you have at least 1/2" of clearance or more between the drive tires and the lawnmower wheel shafts.

You are almost ready to go.

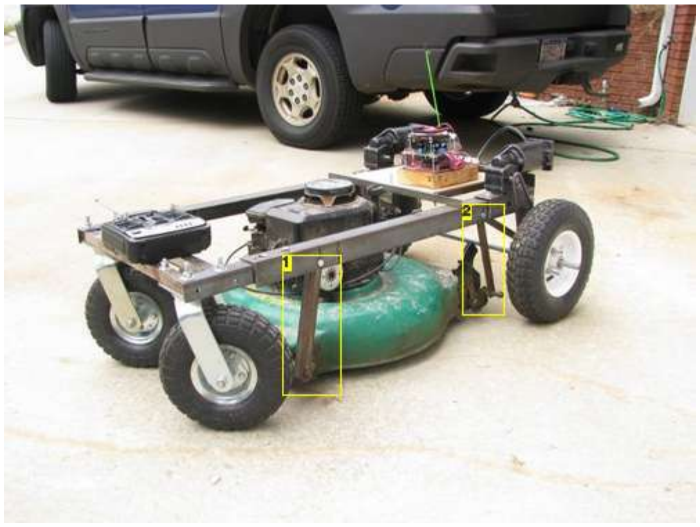


Image Notes
1. the front left mower deck hanger
2. the rear left mower deck hanger

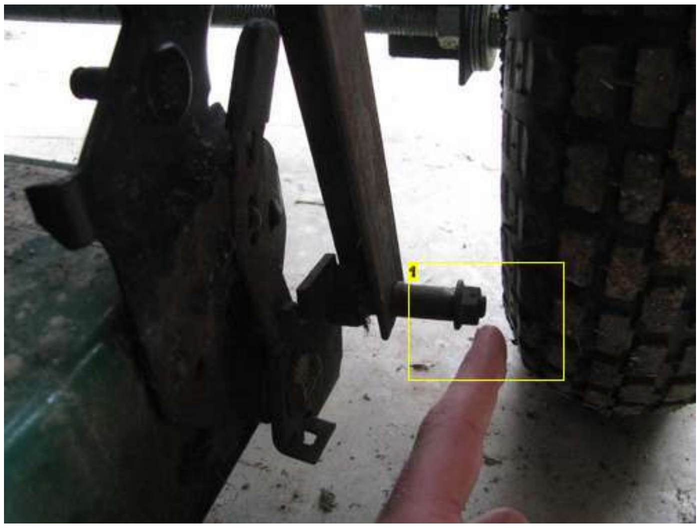


Image Notes
1. make sure to keep the old wheel shafts from touching the drive tires (leave 1/2" or so)

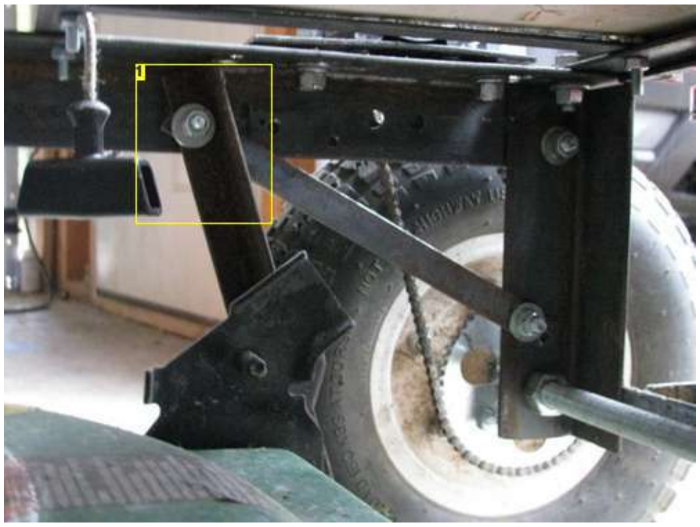


Image Notes
1. save a bolt on each side by using the same one that you used to bolt the frame riser brace into the frame.

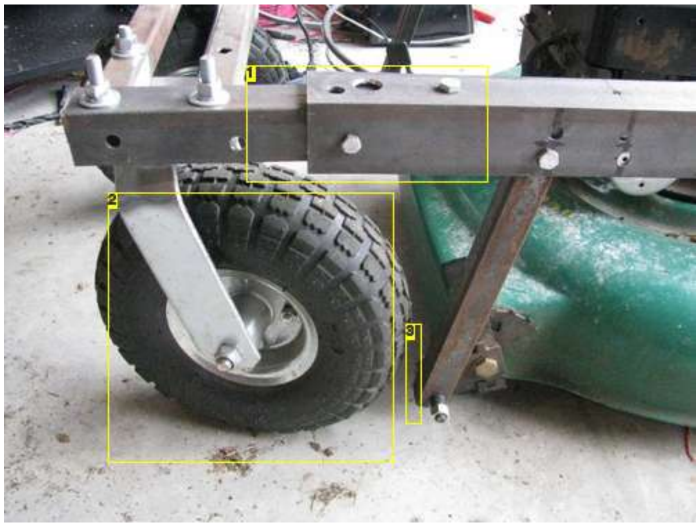


Image Notes
1. adjustable total length (for different model push mowers)
2. caster wheels with 360 degree turning
3. leave a gap or the front wheels will hit the mower deck!!!

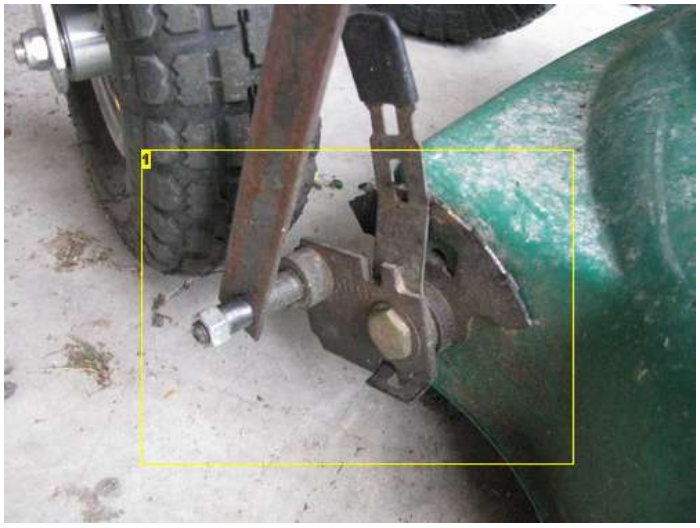


Image Notes
1. by mounting the lawn mower deck-hangers to the old wheel shafts, you can still

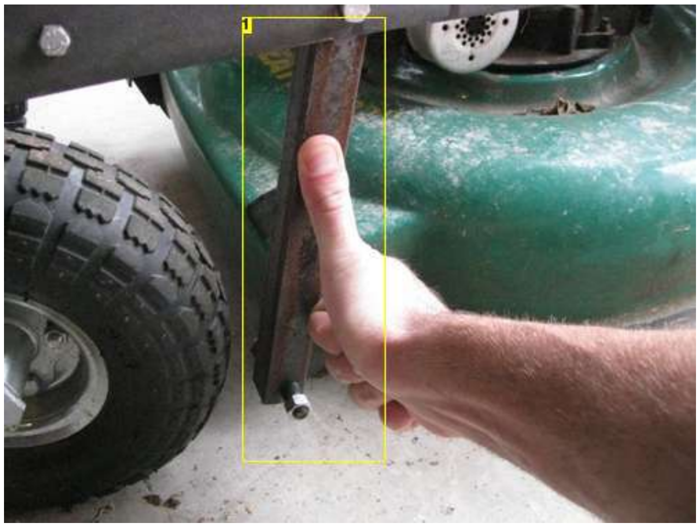


Image Notes
1. these are the 1" angle-iron lawnmower-deck hangers, they hold the mower-

adjust the mowing height of the mower deck without taking anything apart.

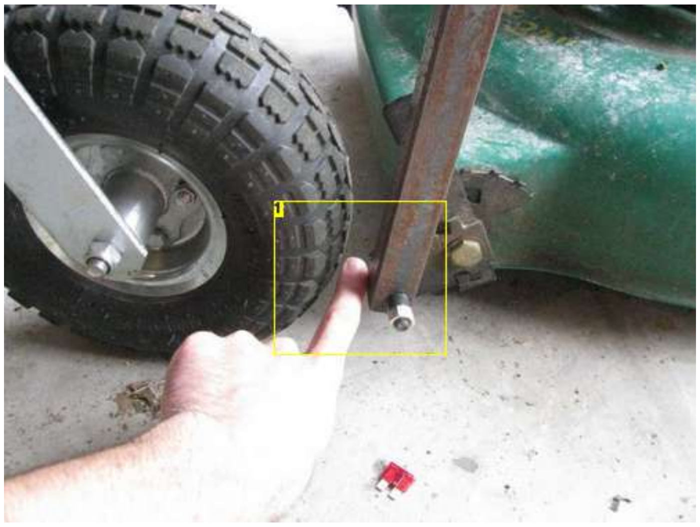


Image Notes
1. make sure the front caster wheels won't hit the mower deck when they swing around (leave at least 1/2" clearance)

deck to the main frame

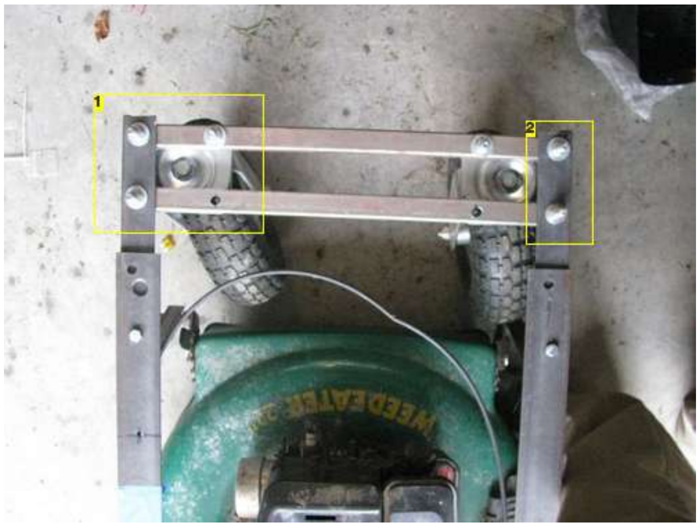


Image Notes
1. I only installed 3 of the 4 bolts on each front caster wheel.
2. these 2 bolts on each side go through the caster wheel mounting plate AND the frame

Step 8: Select and Install the batteries

This is the simple part. Go BIG. I only bought 1.. which I got at Walmart for $62.

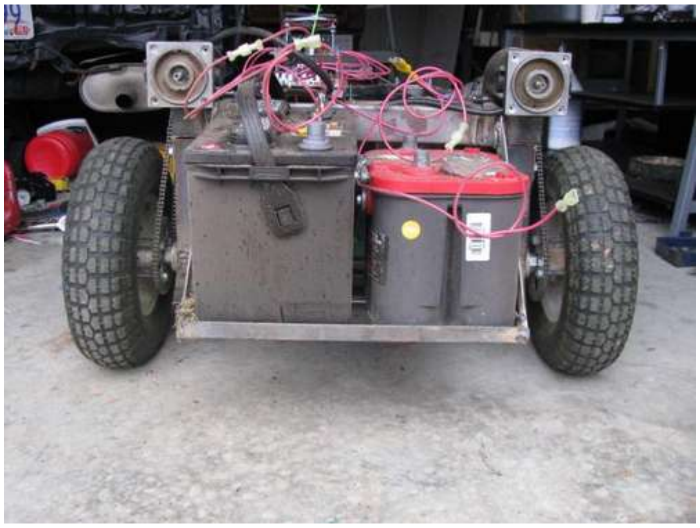
I got 2 car batteries (actually 1 marine deep cycle and 1 gel-cell car battery) both 12vdc. They together keep my lawnmower running strong for the duration of my front and back yard (I have about 1/2 acre of grass to cut and it is somewhat hilly). I slacked while trying to learn about batteries and just went with the biggest ones I could find for the price (the gel cell is actually used). I initially thought 12vdc would work, but the added weight of the mower deck made it travel so slowly at 12vdc, that it would not quite make it up some larger hills, so 24volts was necessary. The 2 batteries are connected in series with each other.

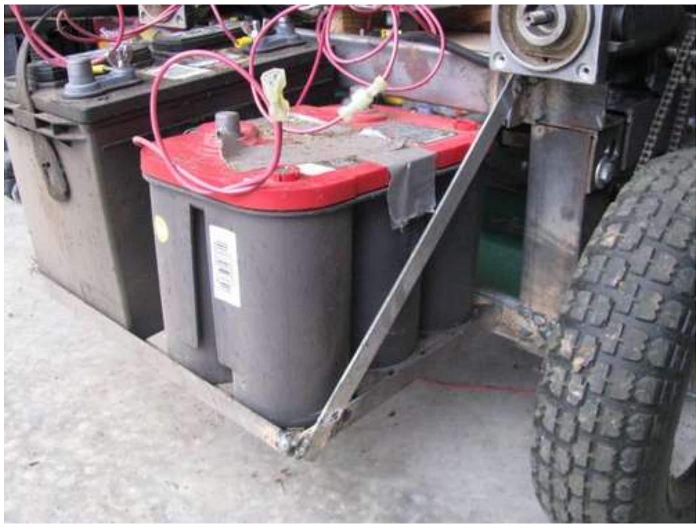
The microcontroller is also powered by these batteries. I have never had any problems with the electronics not getting enough power, so I didn't see the need to have a separate power supply.

The batteries (due to their weight) are mounted behind the rear wheels. This GREATLY improves control of the bot because it counters the weight of the mower deck in front. Zero-turns are very easy now.

I needed a place to hold the 2 big batteries that were going to power the lawnbot, so I measured the 2 batteries and welded a small 1" angle-iron frame to hold them. It is welded to the rear of the frame behind the drive axle to maintain even weight distribution.

You can use bolts and 1" angle-iron to make a battery holding cage that is bolted to the rear of the bot, or you can use smaller batteries and secure them to the top of the bot. 12v 20ah Sealed Lead Acid batteries can be found online for around $35-45 each. Any battery rack that you can whip up will likely be just fine, as long as it can support the weight of the batteries it is carrying. I used a welder to speed up the process.





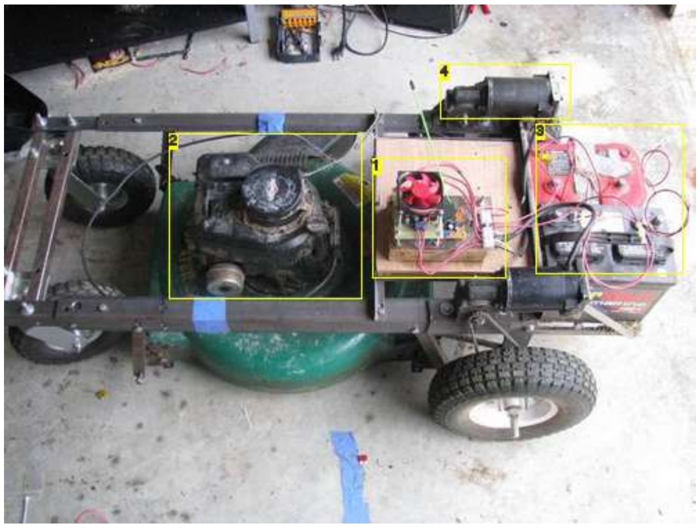
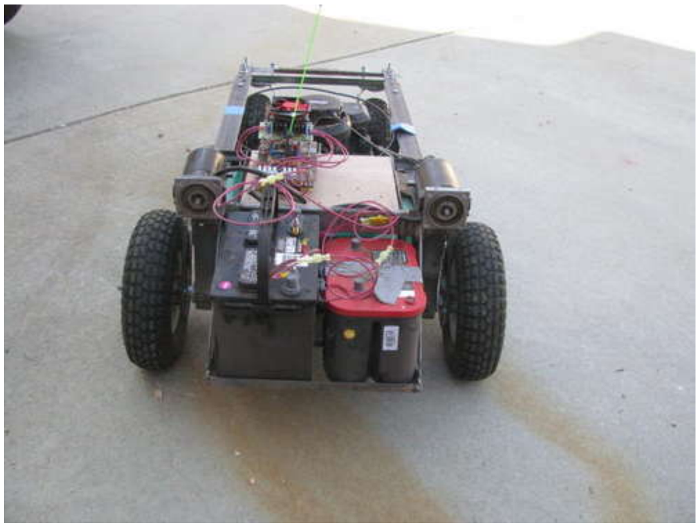


Image Notes
1. Motor controller and Arduino
2. push mower
3. (2) 12v batteries (deep cycle marine is the best)
4. electric wheel-chair motors



Step 9: Mount the electronics

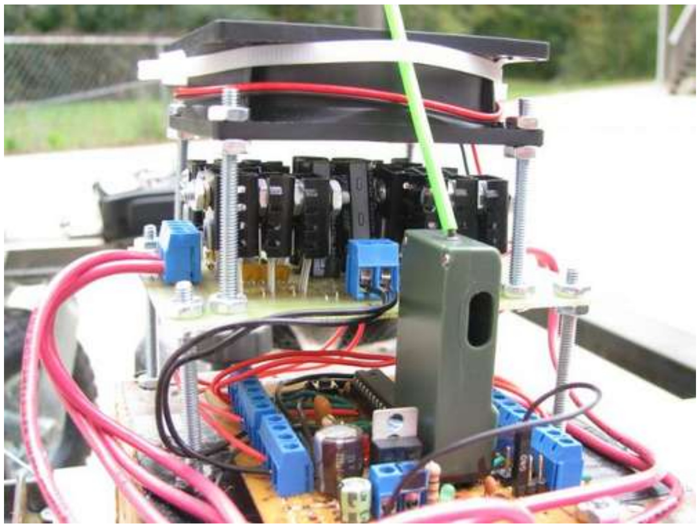
Connect the electronics to the motors and batteries. The motor drive board has 1 connector for the main battery power and 1 power connector for the 80mm cooling fan that I would highly recommend you install directly above the mosfets. There is spacing for some long skinny bolts to hold a cooling fan. I bolted the motor driver above the Arduino breakout board to save space.

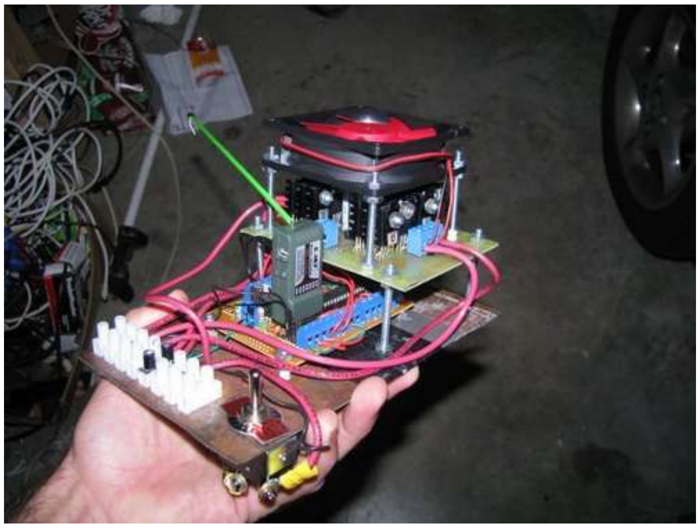
Also, you might want to use some smaller wire coming from the batteries to power the Arduino board, as the 10ga wire I used for main power and motors is a bit overkill for the microcontroller.

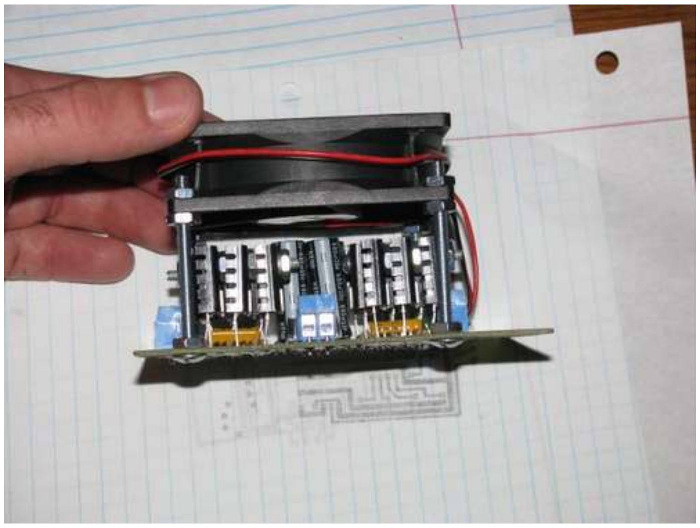
I installed a 30a 120v toggle switch from Radio Shack to switch the main power ON/OFF, this is my kill-switch. I also found a terminal-block for power distribution at Radio Shack for a few bucks. It is the white thing that all the wires go into in the pictures. This makes removing the electronics a whole lot easier.

It is very important that you wire everything up correctly. Otherwise you might blow up the motor controller.

So make sure that you check the code before connecting anything to verify that you haven't mixed any wires up.







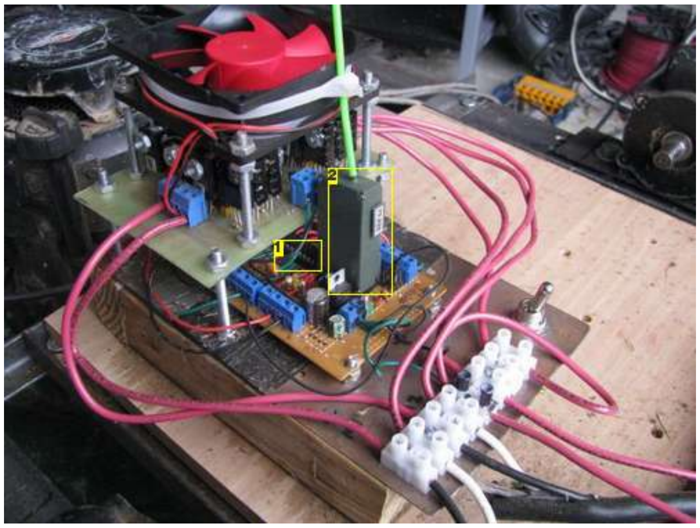


Image Notes
1. Atmega168 microcontroller programmed in the Arduino, then transferred to this home-made breakout board for permanent use.
2. The R/C receiver is plugged directly onto my home-made breakout board which supplies the +5v and GND needed for power as well as a breakout screw-terminal for each channel. This receives the signals from the remote-control (R/C transmitter) and sends them into the Atmega168 for processing.

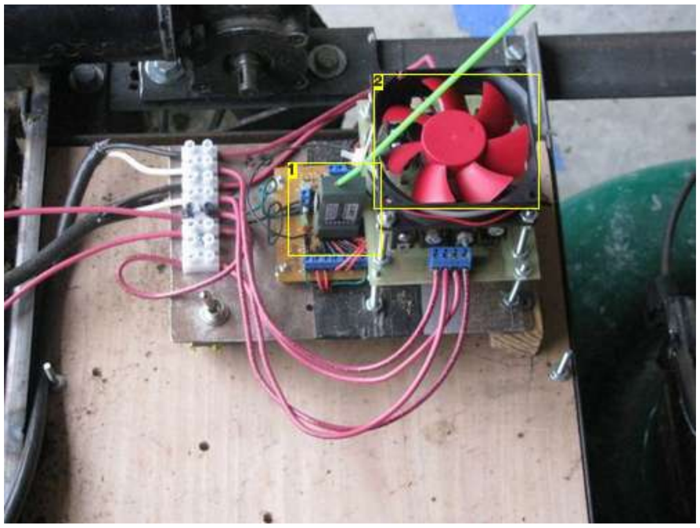


Image Notes
1. R/C receiver plugged into Arduino breakout board
2. cooling fan for motor controller (h-bridge)

Step 10: The Code

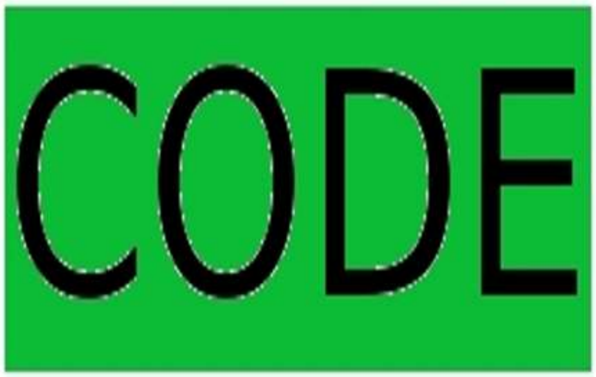
I changed the code so that the Interrupt Service Routines (ISR) would run more quickly and the sketch would spend less time in the ISR. This means less overhead which means more signals are processed and smoother operation of the bot.

I also added a 2nd sketch for the 2nd microcontroller to process 2 signals (you can add as many more as you want) using the pulseIn method instead of using interrupts. This only processes about 1/5th of the available signals from the R/C Receiver, but also severely decreases the chance of receiving a "BAD" signal. Also, since the power relay is setup to only be ON if the signal is "GOOD", when you go out of range, it automatically shuts off the power to the motors only.

The 2nd Atmega by default should have digital pin 4 used as the R/C servo signal input from the R/C receiver, digital pin 6 should control a 5v relay or N-channel mosfet that is used to switch the 60amp power relay ON/OFF. That is all that is needed, you can also use an LED on pins 12 and 13 to indicate whether the relay is ON or OFF.

You can also add 2 12v running lights from Walmart for a car... I use an N-channel mosfet directly tied to pin 9 of the 2nd Arduino to control the brightness of the lights using a hacked channel on my transmitter. This input from the receiver would go to digital pin 2. Check the code.

Download the .zip file on this page and upload the sketches. If you don't plan on adding the 2nd Atmega with the failsafe and killswitch, that is fine. You can still update the new code for just the main Atmega and it should run more smoothly.



File Downloads

Lawnbot400_code.zip (152 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Lawnbot400_code.zip']

Step 11: More Videos

here are a few more videos in case anyone wanted to see...

#2

#3

#4

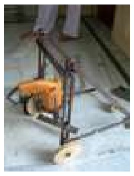
#5

Related Instructables

How To Change Lawn Mower Oil by toolrepair

How To Replace a Lawn Mower Blade by toolrepair

How To Build A Solar Charged Remote Control Electric Lawn Mower (video) by hastyhost


Electric Lawn Mower by susanta

Robocam - Homemade Video Robot (Photos) by talk2bruce

Remote Controlled Lawnmower (Photos) by Bob Bowie

remote start system for car truck and suv by lonemeno

Remote control lawnmower by pfoglietta

How to Build an Arduino Powered Chess Playing Robot

by **mJusticz** on March 4, 2011

Intro: How to Build an Arduino Powered Chess Playing Robot

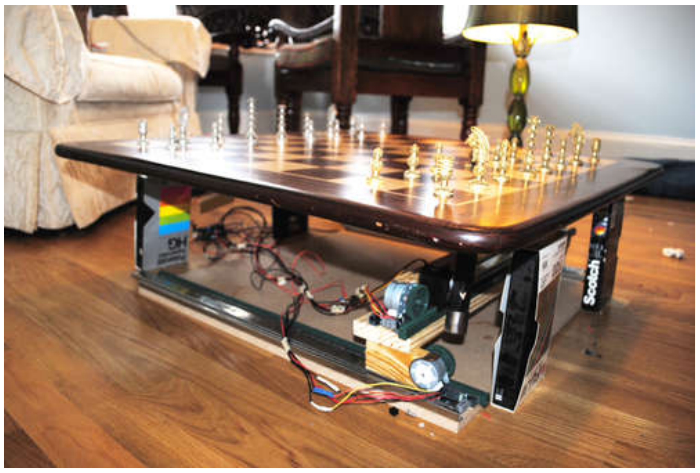
Judging by the sheer number of chess related Instructables, I think it's safe to say the community enjoys the game. It can be difficult, however, to find someone who plays on the same level you do. To solve this dilemma, and to increase my playing skills, I built this arduino powered chess playing robot.

****UPDATE** PCWorld just posted about this project on their blog! Thanks, guys!**
This project is entered in the 13-18 division of the robotics week contest.

The board works like any other xy table, with a few key differences. First, the x axis has an extra servo attached to it, which raises and lowers a magnet. The magnet is attracted to pieces on the chess board above, allowing them to move. Second, embedded in the board are 64 magnetically activated reed switches, allowing the arduino to know the location of each piece.

What I love about this project is its adaptability. If you decide you're done with it as a chess board, it can instantly convert into a CNC mill by modifying a few pieces. I'll talk more about this possibility at the end.

All in all, while there was definitely some great novelty in watching the computer move pieces around with the board, this project was not as successful as I had hoped. The magnets were way too powerful, so extra pieces would often get drawn in when they shouldn't have. I think with some changes this could have been an even better, more functional project. However, I think this Instructable still serves as a pretty good guide to make your own functioning chess playing robot.



Step 1: Parts and Materials

You may have many of the parts for this project already, but if you don't, the whole list costs ? $350, depending on where you get your parts from. Many, many of them can be salvaged, so look to recycle before you buy!

- 1 Arduino Uno or Diecimila

We'll be using this arduino to drive our stepper motors and servos. You can pick these up just about anywhere online. I got mine from Adafruit. $30

- 1 Arduino Mega

This is the most expensive item in the project. It'll be dealing with the inputs from each chess square to let the computer know where you've moved. We're using the mega here due to its speed and number of inputs. Adafruit $65

- 1 Mux Shield

The mux shield (short for multiplexer) gives us even more inputs for our arduino mega. We'll need 64 inputs in total, one for each square. Sparkfun $25

- Motor Shield

The motor shield will be controlling our stepper motors and servo. You'll need to solder it together. Adafruit $19.50

- 1 Large chess board with pieces

This one is a little more self explanatory. We want a large chess board here because the pieces need to be able to move in between each other with disrupting others. Make sure you measure the diameter of the bottoms of the pieces. We'll need that in a moment. I'm not sure where mine is from, but you can pick them up from a flea market for a bargain. The playable area of my board is 24".

- 64 NO Reed Switches

Reed switches are magnetically activated switches. They'll help us find the location of moved pieces. NO stands for normally open, that is, the circuit is disconnected Digikey ?$30

- 16 10K 1/4 Watt Resistors

These are the pull up resistors for the built in digital pins. The mux shield, luckily, has integrated pull downs, so we don't need to worry about those. Digikey ? $2

- Roughly 90 feet of 30AWG Wire

This is the hookup wire for all of our sensors. Radioshack ? $16

- Neodymium Magnets to fit your pieces

This is where the measurements from the bottoms of your chess pieces come in handy. You'll need disc magnets to fit underneath each piece. For proper strength, they should be about 1/8" think. A great source for these is K&J Magnetics . ? $55

- 1 Large Neodymium Magnet

This magnet will be attached to the XY table underneath the board, to move each piece around. K&J Magnetics $19 Note: This was Waaaay too powerful. It would draw in pieces it shouldn't have. You'd be better off going with some smaller ceramic magnets, like you'd find at Staples or another office supply store.

- 2 Pairs of 24" Drawer Bearings

The size of your bearings will depend on the playable area of your chess board. These allow for the stepper motors to move back and forth underneath the board. Amazon ? $30

- 2 Stepper Motors

Stepper motors can move in very precise increments. In the late 90s they were in just about every piece of tech you could find. The best place to get these are in old dot-matrix printers. You can them at the flea market for next to nothing!

- 2 Vex Rack and Gear Sets

The rack gears allow the stepper motors to travel on the drawer bearings. See the Step 4 for a more detailed explanation. Vex Store $40

- 1 Standard Hobby Servo

This servo will be raising and lowering the powerful magnet below the board. You can find them at a hobby shop for ? $10, or Amazon ? $12

- 1 2' x 2' Perf Board

The perf board is super thin and will be the mounting surface for all of our reed switches. The price will vary greatly on this one, but I got mine from Home Depot for ? $5

- 1 2' x 2' x 1/2" MDF Board

Similar to the perf board, I got this from Home Depot for ? $5

- Various lengths of scrap 1"x2" wood

This wood forms the bridge between the X-Axis drawer bearings. Go behind any hardware store and you'll see dumpsters full of this stuff for totally free!

- 5 Minute Epoxy

This stuff is a godsend. It's used for just about everything in this project, from mounting motors to attaching the rack gears. I'm in love -- and I picked mine up from Radio Shack for $3

- 1 Wood Saw

You probably already have this one, but if you don't, I picked mine up at Ace Hardware for $10 a couple of years ago.

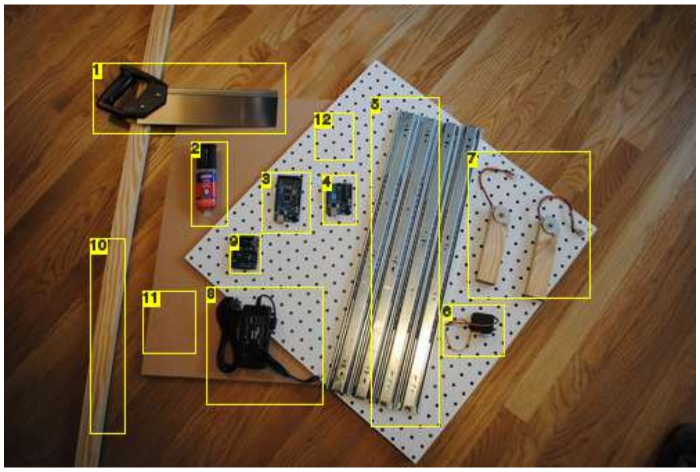


Image Notes
1. Standard wood saw
2. 5 minute epoxy
3. Arduino Mega
4. Arduino Uno. A Diecimila would also work.
5. 24" Drawer Bearing
6. Standard Servo
7. Two 24v 1A stepper motors. They are already mounted here. I describe the

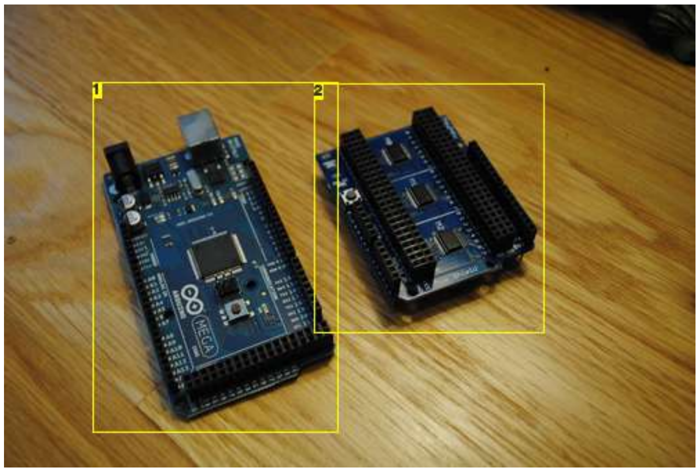


Image Notes
1. Arduino Mega... This thing is a beast!
2. Mux Shield. This gives us an additional 46 inputs on our mega.

process in the next step.
8. 24v 1A power supply
9. Adafruit Motor Shield
10. 1x2 lumber. Some of the better stuff I have found for free behind home depot.
11. 2' x 2' MDF
12. 2' x 2' pegboard

Image Notes
1. 24 gauge hookup wire. This stuff is pretty cheap -- a 90ft roll is about $5
2. Diagonal cutters -- these are great for clipping leads that are just a little too long.
3. Heat shrink tubing is great for cleaning up your connections. It is optional.

Image Notes
1. These are reed switches... metal encased in glass. They are a bit fragile. I would buy a few more than you need just in case.

Step 2: Design and Code Explanation

That parts list is a bit scary if you're not sure what everything is going to do, so here's how many of the pieces will be used.

You can see in the images below that each stepper motor can move freely about its axis thanks to the drawer bearings. On the Y Axis, each rail is connected with the wooden structure, so that the X Axis may sit atop it. Also on the X Axis is the servo that raises and lowers the powerful magnet, so that it may position itself before moving pieces.

Feel free to download the sketchup file and mess around if you're not sure of anything.

Another interesting element of this design is how to code talks with the arduino and motors. We need to address each square as a set of coordinates so that we may find slope and distance, however the traditional method of labeling squares A1, A2, etc. doesn't work particularly well in code. Standard (x,y) coordinates are much friendlier. Those coordinates, however, need to be in the form of a single number. What I ended up doing is assigning each square to a number, as you can see in image #3. Those numbers don't really work as coordinates on an 8x8 chess board, however, because we use a base 10 number system.

To solve that issue, we take the base 10 number of each square and convert it to base 8 using the modulus operator in C. 27, for example, is 33 in base 8, with the first digit being the x coordinate and the second the y. If you count over three squares and up three squares, voila! You end up on square 27. This converted coordinate system ends up looking like image #4.

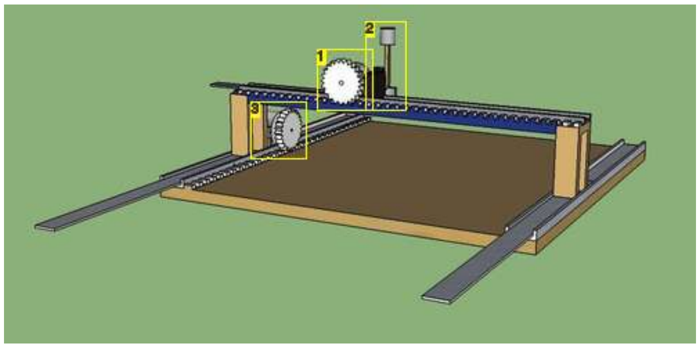


Image Notes
1. X Axis Stepper Motor
2. This is the servo that raises and lowers a powerful neodymium magnet.
3. Y Axis Stepper Motor

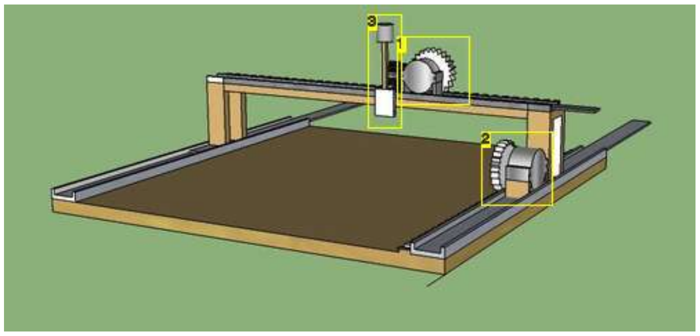
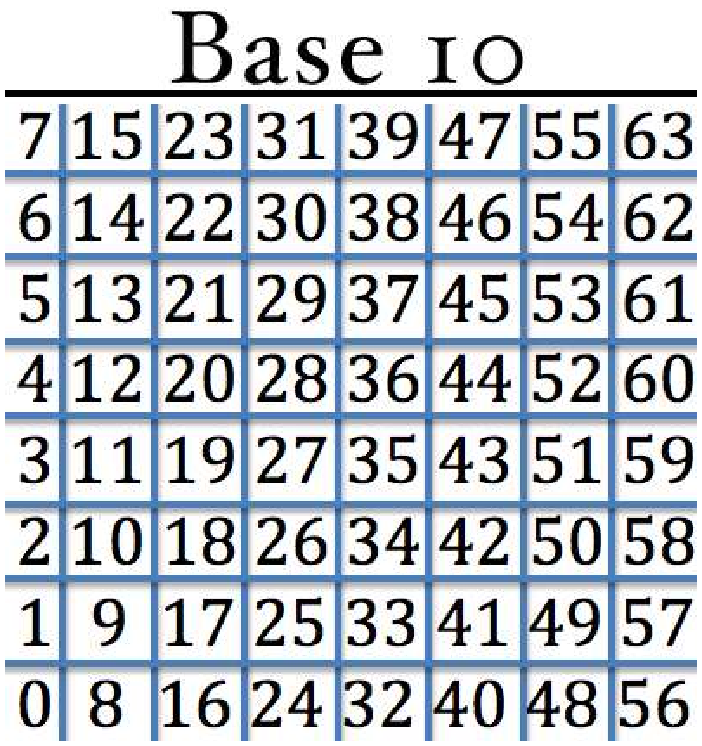
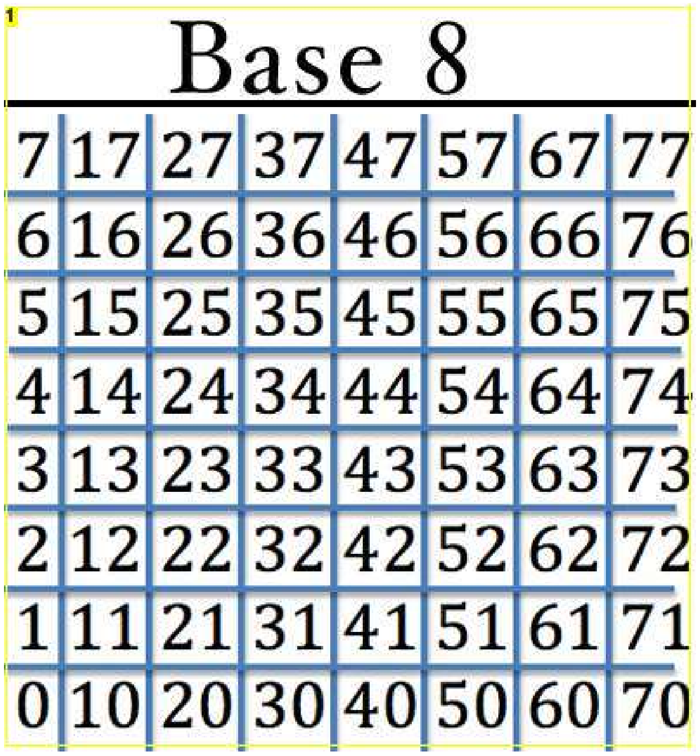


Image Notes
1. X Axis Stepper
2. Y Axis Stepper
3. Servo to raise and lower magnets (pseudo z-axis)



Base 10

7	15	23	31	39	47	55	63
6	14	22	30	38	46	54	62
5	13	21	29	37	45	53	61
4	12	20	28	36	44	52	60
3	11	19	27	35	43	51	59
2	10	18	26	34	42	50	58
1	9	17	25	33	41	49	57
0	8	16	24	32	40	48	56



Base 8

7	17	27	37	47	57	67	77
6	16	26	36	46	56	66	76
5	15	25	35	45	55	65	75
4	14	24	34	44	54	64	74
3	13	23	33	43	53	63	73
2	12	22	32	42	52	62	72
1	11	21	31	41	51	61	71
0	10	20	30	40	50	60	70

Image Notes
1. The first digit corresponds with the x axis and the second digit with the y.

File Downloads

XY.skp (347 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'XY.skp']

Step 3: Mounting the Drawer Bearings (Y Axis)

The drawer bearings are what allow the axes to move in their respective direction. The mounting instructions may vary slightly depending on the brand, but usually it's as simple as driving a couple of screws.

The only reason I've made this its own step is that **aligning the bearings perfectly is key.** Should you fail to do this, and they both point slightly outwards, they'll stop at some arbitrary point and refuse to move once you connect them. Save yourself a lot of trouble and use something you know is square as a reference for alignment. The corner of a book is perfect.

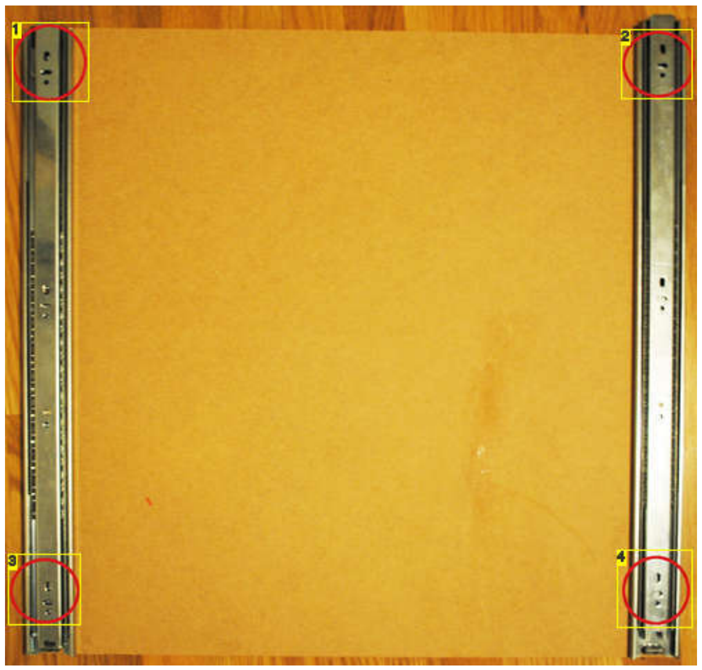


Image Notes
1. There is a screw behind this bearing. My screws came with the mounting

brackets.
2. Screw here
3. Screw here
4. Screw here

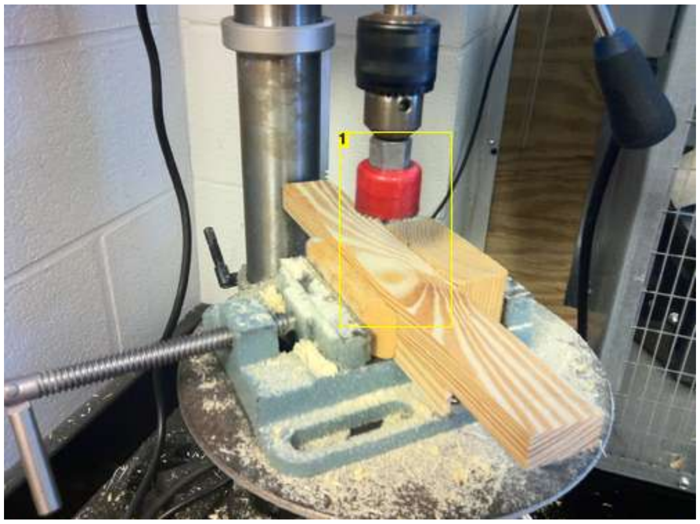
Image Notes
1. Nice and flush!

Step 4: Building the Motor Mount (Y Axis)

The stepper motors we'll be using have fantastic torque, but are circular. This means mounting them to our bearings later on will be nearly impossible, unless we build a square mount. To build one, find a hole saw with a similar diameter to your motor. You'll want to use a drill press rather than a portable drill for this, so I borrowed my school's.

Once you've cut the hole, slice the circle in half to get two mounts. This chipped the tips of my semicircle, so I used some 220 grit sandpaper to clean up the edge.

My steppers came with mounting screw holes, which line up well with the wooden frame. I used the smallest screws I could find. Mine fit so well that it wasn't necessary, but you might consider adding a bit of epoxy to strengthen the bond.



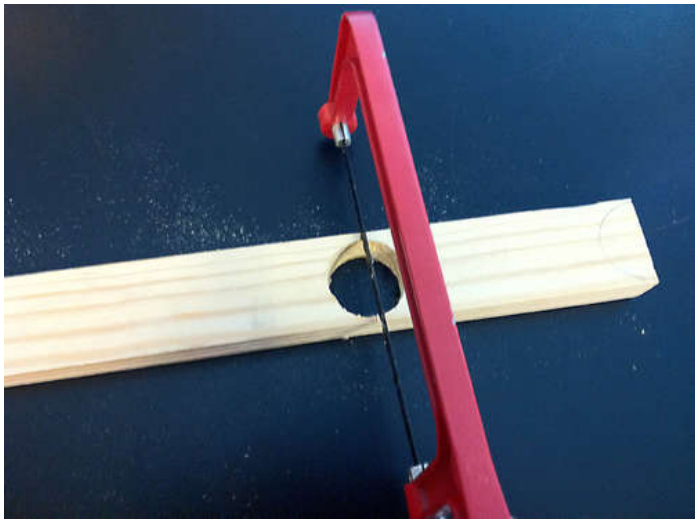
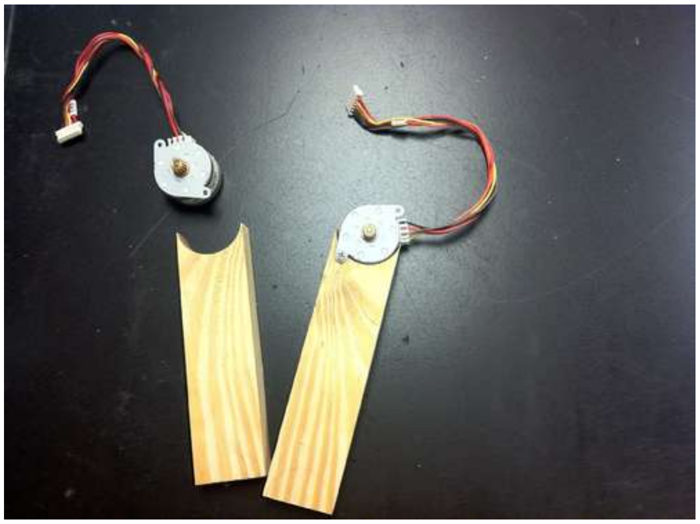


Image Notes
1. Go slowly or you risk chipping the wood on the other side.



Step 5: Installing the Rack Gears (Y Axis)

The rack gears are what allow the motor to latch onto a surface to pull itself along. Your physics teacher probably defined them as a way to convert rotational energy to linear.

Again, we use the epoxy to attach the gears onto the MDF. In addition to heavily applying epoxy to the board itself, make sure some is spread on the side of the drawer bearing, that way there is stability in two dimensions. Do your best to prevent epoxy from getting in places it shouldn't be -- you may gum up your motor.

It works out that the rack gears extend a little bit off of each end. This is a good thing -- it enables the gear to travel the full length of the board without running off. The stepper motor will be offset just enough that if the gears only covered the board's length the whole motor assembly would get stuck at one end.

Also install the circular gears onto your motor at this time. Mine had a set-screw, but you may wish to use some JB weld to hold your gear in place. If you go that route, the joint needs to fully cure before you try to use it, or you risk the gear popping off!

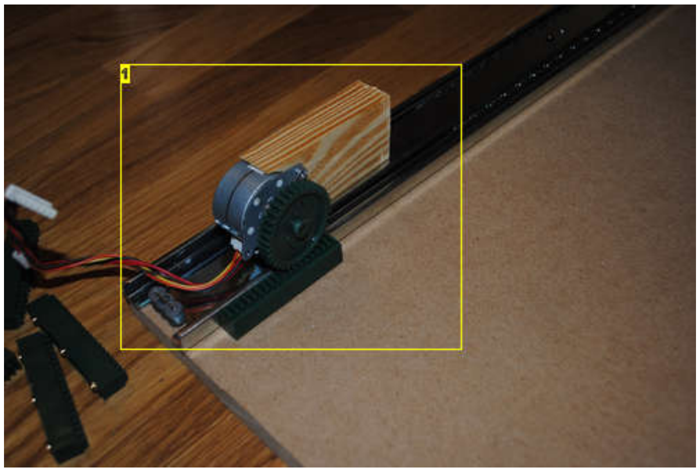


Image Notes
1. This is how the motor will fit into the rack gear once completed.

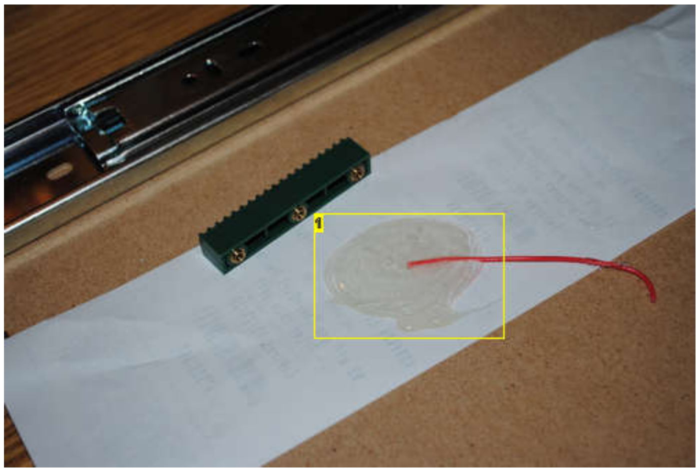
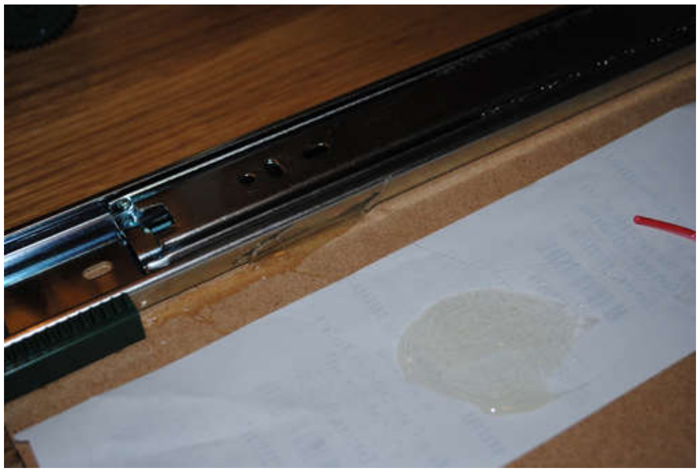
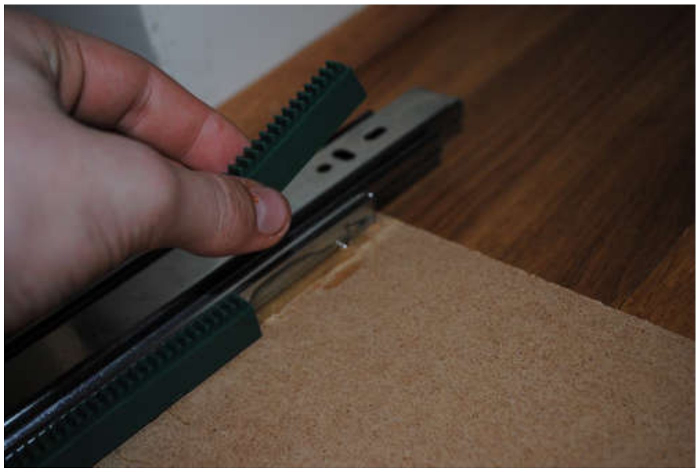
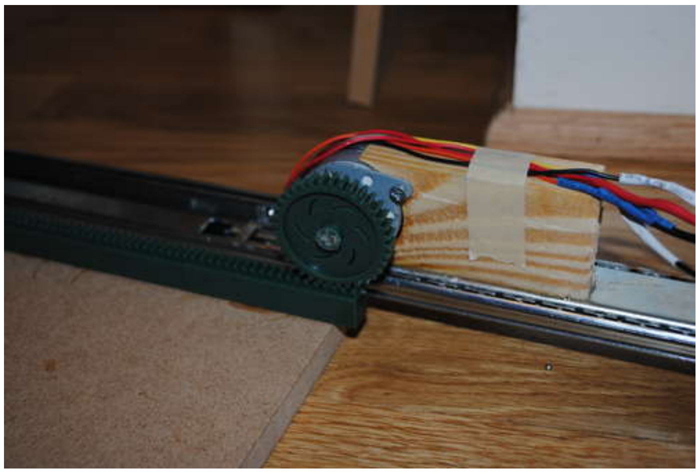


Image Notes
1. Make sure to mix this really thoroughly or it may not dry.







Step 6: Wiring and Mounting the Motor (Y Axis)

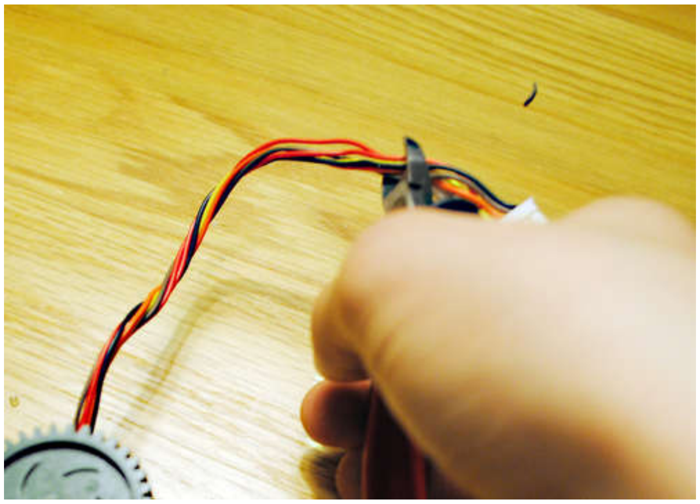
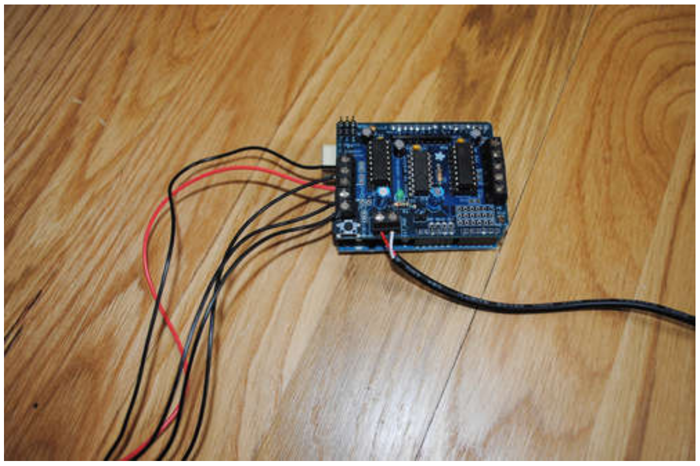
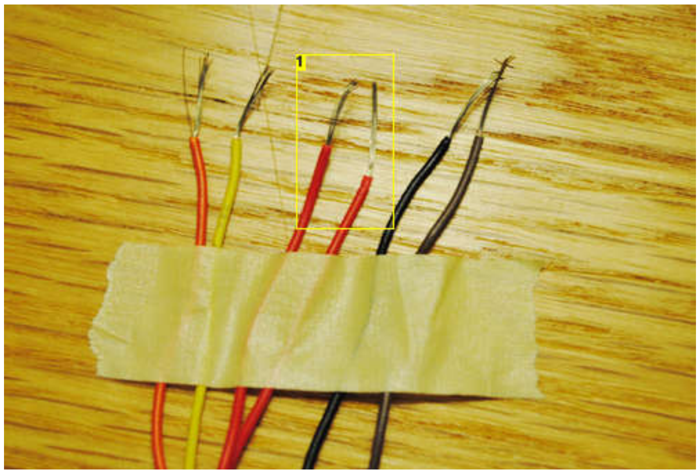
The leads that come attached to the stepper motors are very short. Because the Arduino is mounted off the board, the wires need to be at least the length of one side. That made mine about 2' 5" long. Heat shrink tubing is your friend here -- we're using enough power that it might arc if you're not careful.

If your stepper motor has 5 wires, you're all set. If there are 6, however, it means you have to connect your center taps. Jason Babcock has a great tutorial on reverse-engineering your motors. In my case, however, the wires were the same color.

After extending the wires, the center taps go into the center of one of your motor hubs. The wires from one coil go to one terminal on the motor shield, and from the other coil to the other terminal. At this time we also hook up our 24v 1A power supply to the motor shield. If you get the polarity wrong on this, your motor shield is toast.

After trimming the motor mounting block to about 4 inches, it's time to attach it to our bearings. Mix up the epoxy, and liberally apply it to the area of the bearing the block will touch.

Also, if you have any pets, be sure to animal-proof the room you're working in. Cats seem to have an affinity for knocking over things that are drying.

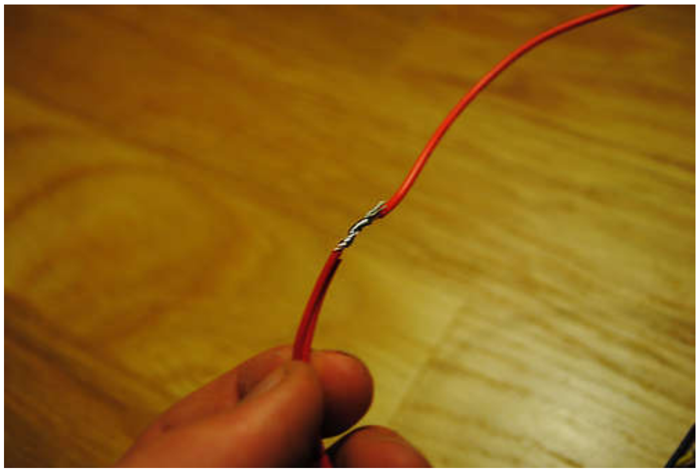
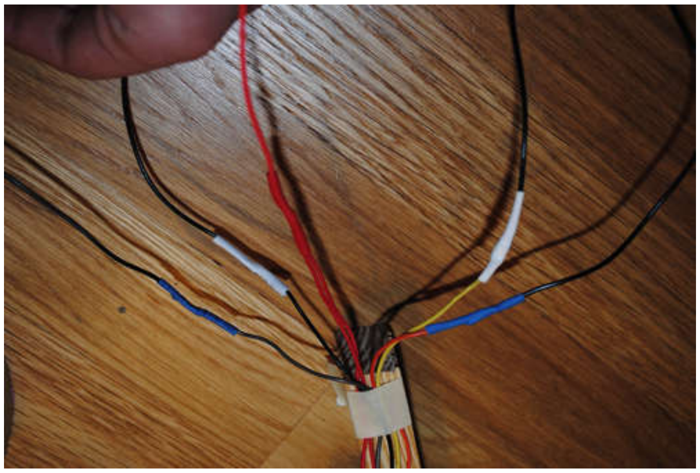
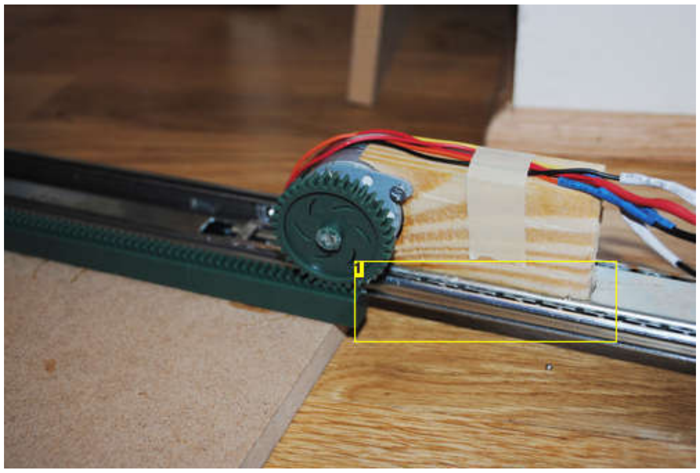


Image Notes
1. These two center taps get connected. Usually they will be the same color.





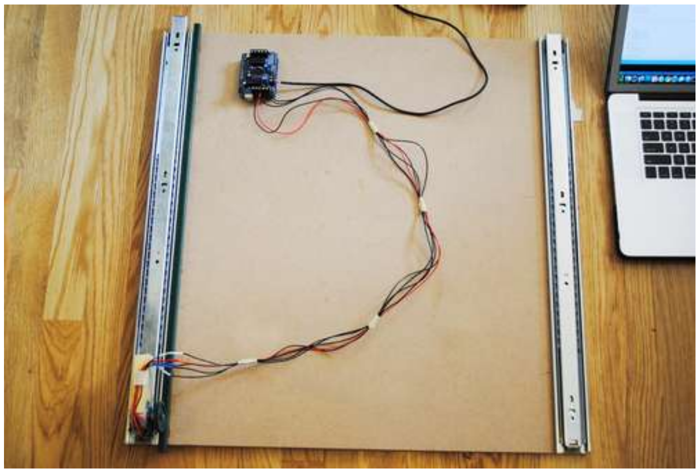


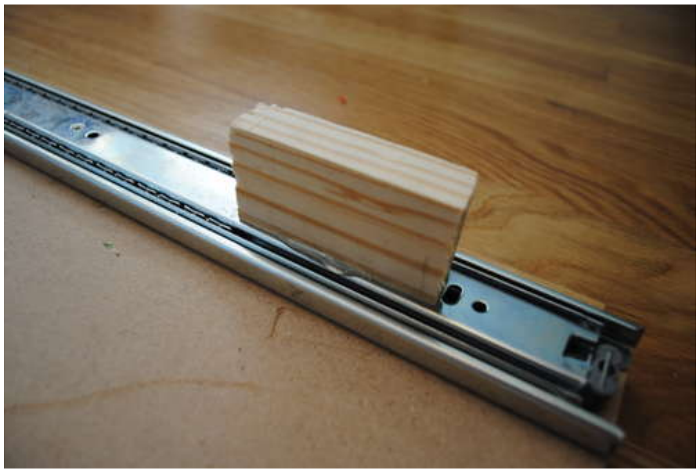
Image Notes
1. Make sure to keep the bearings epoxy free.

Step 7: Mounting the Crossbars (X Axis)

Atop the Y Axis sits a cross-bar which will hold the X Axis. To do this, we mount another block on the opposite Y Axis bearing. This will across from the motor mounting block.

Then we cut two 2' lengths of 1"x2" wood and mount them to the blocks with wood screws. Make sure these screws are in tight, or the bearings might not move at the same time. You might consider adding some wood-glue to lock them more tightly in place. While these cross bars need to be large enough to support our X Axis bearing, we want to use the least amount possible to avoid unnecessary weight. If the entire X assembly weighs too much, our Y Axis stepper motor won't have enough torque to move efficiently, or, if it's really heavy, at all.

Test that your bearings move evenly with each other. The crossbars should be as close to perpendicular with the bearings as possible.



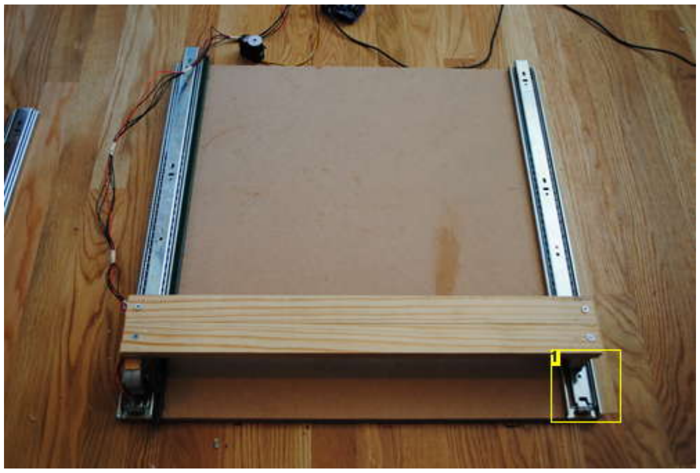


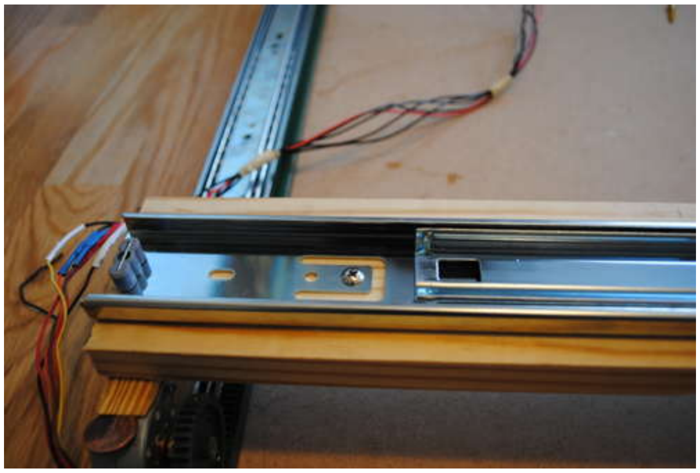
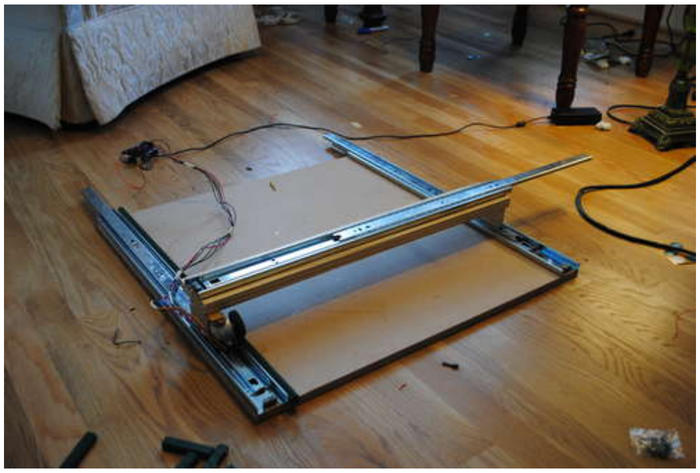
Image Notes
1. Make this as close to a 90 degree angle as possible.

Step 8: Mounting the Drawer Bearing and Rack Gears (X Axis)

We are going to mount this drawer bearing in the same way we did on the Y Axis. Using the included mounting screws, attach it to the crossbars, making sure to leave room on one side for the rack gears.

This time, however, we have some options for mounting the rack gears. We can choose which side to mount them on. This is going to vary depending on how your motors are placed on their blocks. Once you've decided where to place them, though, they are glued in exactly the same way as on the Y Axis. Extend the rack gears slightly off of each end to make sure we have full travel on our steppers.

Again, make sure to try and keep any glue out of the sliding mechanism. If you do allow some glue to enter the mechanism, all is not lost. Fortunately for us, drawer bearings come in pairs, so we'll have one left over anyway. Mount it to the same screw holes and you're set! Just don't do it again! :)

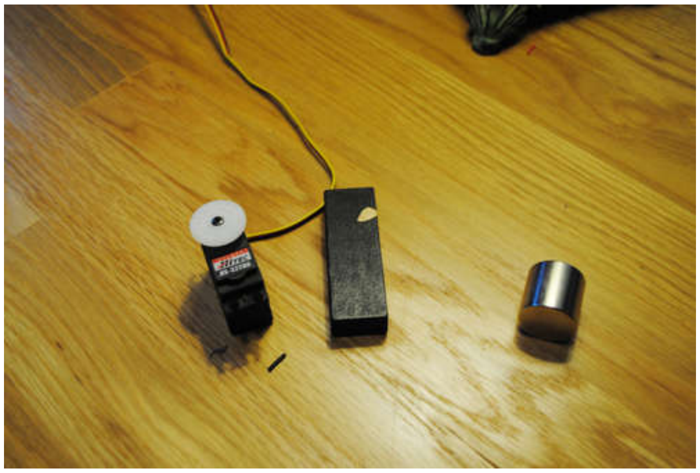
Step 9: Attaching the Magnet to the Servo (X Axis)

The chess board needs to be able to move into position and grab a piece. To accomplish this, we achieve a pseudo Z Axis by mounting a magnet to a servo on top of the X Axis. Use epoxy to attach your large magnet to a small piece of wood (Jenga blocks work wonderfully). Once that has dried, attach the wooden block to your servo.

If you're concerned about the torque of your servo, you might consider adding a counterweight to make lifting the magnet less difficult.

At this time you should also extend the leads on your servo motor. Do this the same way you did for your stepper, going one wire at a time and covering it with heat-shrink tubing or electrical tape.

Epoxy bonds best when there is a significant weight holding the two bonding surfaces together. Keep in mind that while 5 minute epoxy dries in 5 minutes, it may not cure for several hours.



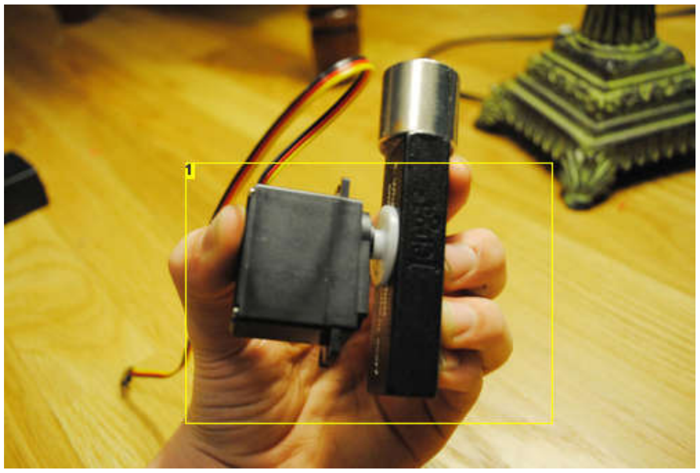


Image Notes
1. Clamp this as soon as you press it together for the strongest bond.

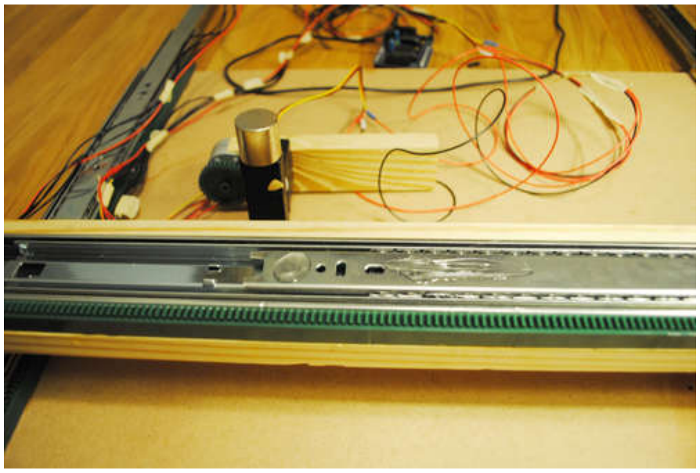
Step 10: Wiring and Mounting the Motor (X Axis)

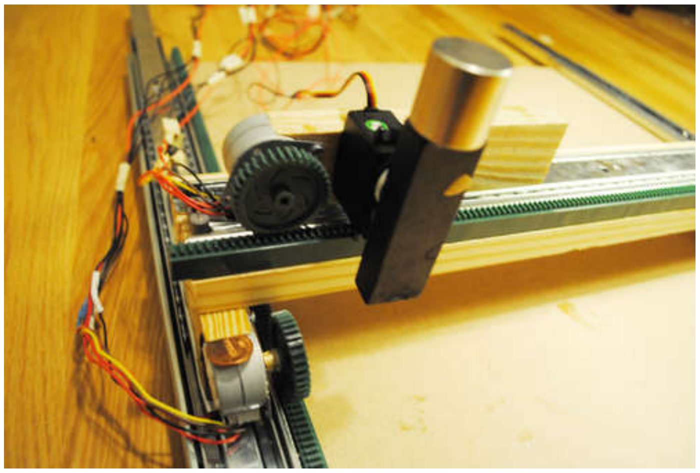
We mount and wire our X Axis motor as we did earlier, however we also need to attach our servo assembly. We need to mount the servo as closely to our motor as we can get without interfering with the gearing. This way we are guaranteed to have maximum coverage underneath the board.

Using a piece of wire or cotton swab, carefully apply epoxy to the block where you've decided to attach your servo. If you aren't careful, some epoxy may end up on your stepper motor, rendering it useless. It's a real pain to clear mixed epoxy out of motors, so it's worth the extra time it will take. Firmly hold your servo in place for several seconds, and leave it clamped to dry.

Once your servo is mounted, we can attach the motor block to our bearings. Being cautious as to not get glue in the sliding mechanism and, like earlier, apply the epoxy to the exposed metal the block will touch.

After that is dry, wire the motor like you did before, extending each lead. This time, however, we'll connect it to the other motor terminal. Congratulations, the XY Table is done!





Step 11: Wiring the Sensors

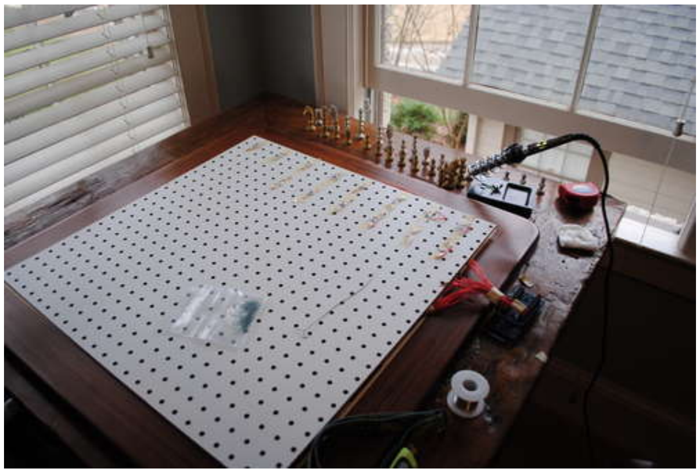
Each chess piece has a magnet embedded in its bottom, which makes detecting location really simple. Underneath each square is a magnetic reed switch hooked up to our Arduino mega. When a change is detected, the Arduino spits out the coordinates.

Pull up resistors make sure we don't get false readings from our sensors. 48 of the 64 switches won't need pull up resistors, because the multiplexer has them built in. Unfortunately, we still have to solder 64 sensors. To make this go a lot faster, tin your wires before you try to solder them to the switches. Basically, just add solder to the wire alone before soldering with it. Label each switch with tape as it is completed to avoid a wiring nightmare later!

Hook up each switch to the Arduino as you go along. The multiplexer has a built in ground next to each input, which is really convenient.

To wire the pullup resistors to the built in pins, connect one resistor end to 5V and one to the pin you're using. Then, skipping the resistor, connect one end of your switch to ground and the other directly to your pin.

Find a comfortable chair, because this is going to take a while!



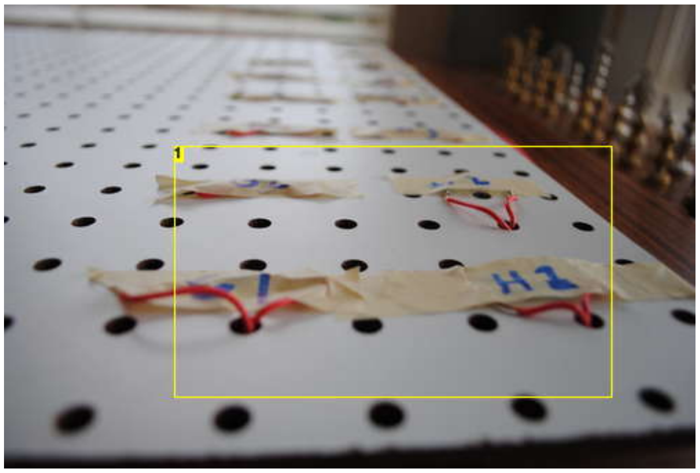


Image Notes
1. The holes in pegboard are at every square inch.

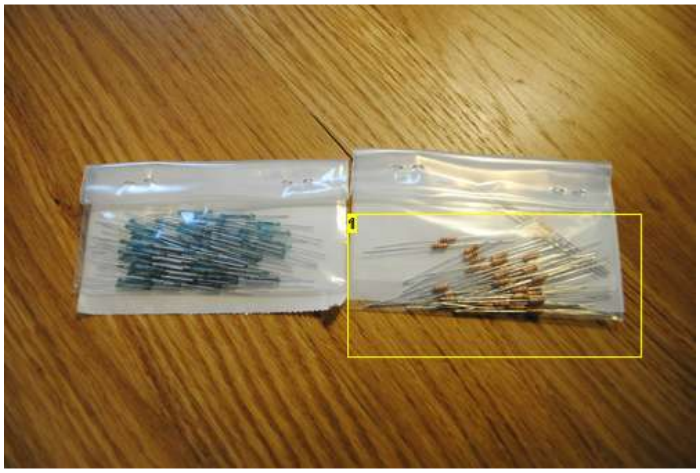


Image Notes
1. The mux shield has resistors built in. You only need 16.

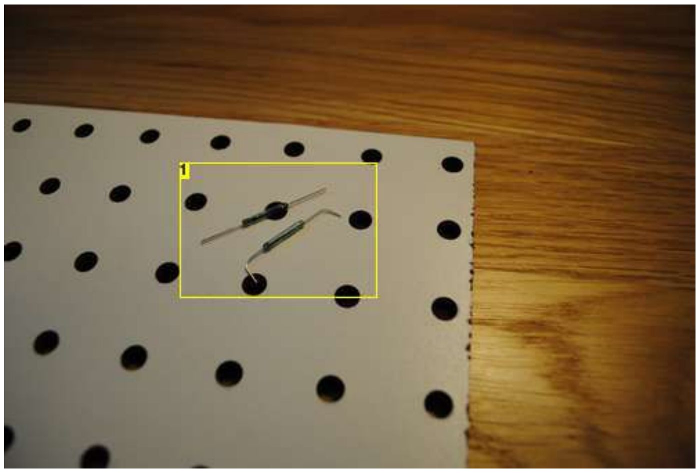
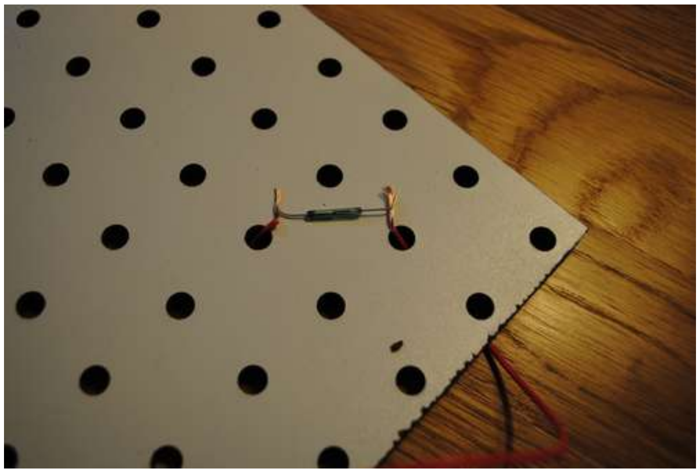
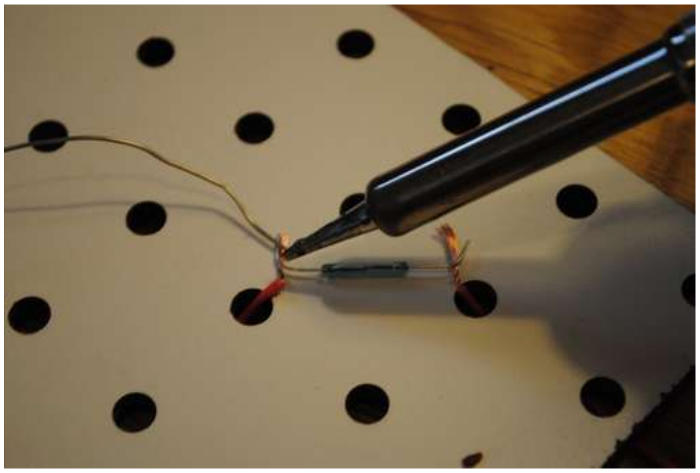
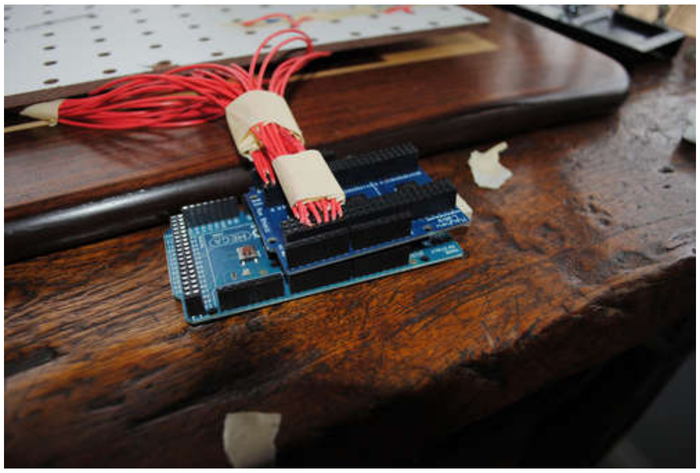
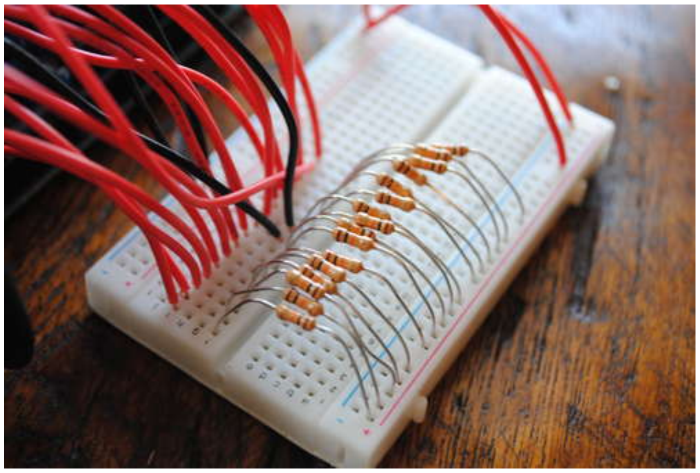


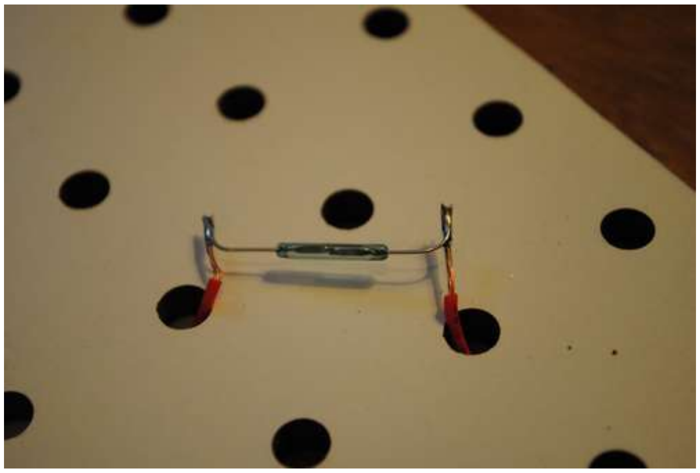
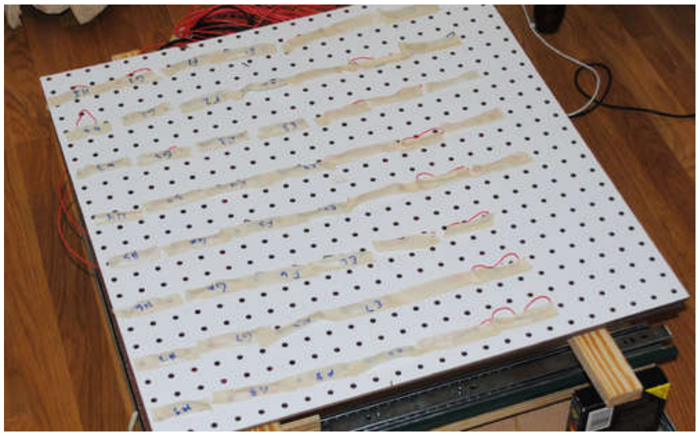
Image Notes
1. Be careful not to break the glass when bending the leads.









Step 12: Place the Magnets

Each chess piece has a magnet embedded in its bottom to trigger the reed switches. The only thing that's important here is that every piece on one side of the board has one polarity (like North), and every piece on the other side has the opposite (like south). If you'd only like the computer to play one color, make sure it is the opposite of the magnet attached to the servo, so they will attract each other.

Other than that, though, this only takes a few minutes.

Step 13: Code, Final Assembly + Reflection

The code for this project is surprisingly simple. It takes the reading from the chess board out of terminal, plugs it into a chess algorithm already built into Mac OS X (look inside the Chess.app bundle), and spits out the coordinates back into the Arduino window with some fancy applescript. The algorithm is a fantastic open source project called Sjeng , meaning this will work cross platform as well.

The final assembly for this project is fairly simple. Find something to hold up your sensor grid, and lay that down. Next, place your chess board and pieces on top. You're done! Run the software and give it a whirl.

If you ever get bored of chess, it's really easy to make this into a CNC project. Replace that magnet with an X-Acto knife, flip the entire assembly upside down, and you have a stencil cutter! Replace that X-Acto knife with a pencil and you have a draw-bot. Modify the servo to activate a dremel and you have a CNC machine! The possibilities are, please excuse the cliché, endless!

Reflection

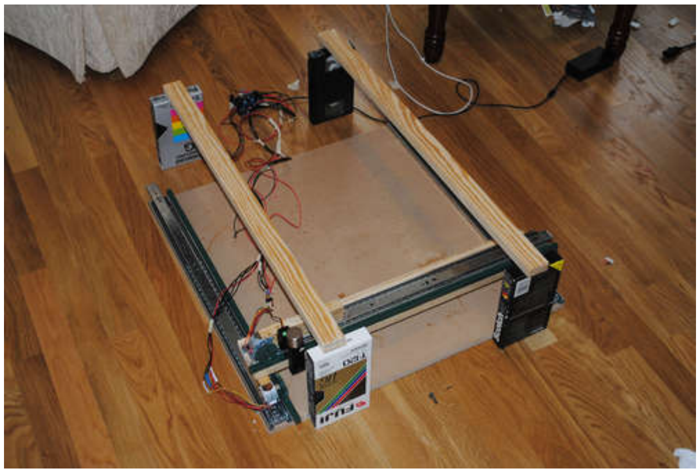
There are changes I would make to this project if I revisit it. First, the mounting of the gears should be improved; a few times the gears would pop off due to the heating of the stepper motors. Proper mounting with screws would fix this. Second, the board gets (fairly severely) out of calibration after a few moves, presumably due to skipped steps, and must be manually moved back to the origin to avoid misalignment. Some potentiometers allowing the board to know its absolute position would fix this. The rails, despite my best efforts, were misaligned, making the last few rows of squares very difficult for the stepper motors, causing them to skip steps and click loudly (bigger steppers would be better). Sometimes the motors would stop altogether and wouldn't be able to make it in those rear squares, rendering a game unplayable without human intervention. Aligning the rails better would fix this. The magnets are also waaaaaay too powerful, and often pull pieces in they shouldn't. This renders games unplayable if you're not actively helping the board out, so weaker magnets really are a must. Often what would happen was that when the large magnet flipped over, a whole bunch of pieces would get drawn towards it (and it's a real pain to place all the magnets back inside the pieces if you haven't attached them). If you don't switch to a smaller magnet, this will happen pretty much every time. The ones underneath the pieces are mostly fine - it's just the main magnet that is way too strong. Ceramic magnets like the ones people stick to whiteboards would work far better. Finally, the code could definitely be streamlined further; I completed this project when I was relatively new to C and Arduino. I'm sure there's a better way to do this than having an applescript copy and paste into the Arduino environment!

Thoughts on the Epilog Challenge

It's kind of a funny feeling when as a maker you have something that can, well, build other things. An epilog laser would help me in some way with literally everything I make. With this laser I would, on a regular basis, cut solder stencils for surface mount work, make more elegant housings for my projects, and engrave logos onto personal laptops and cell phones. As a student, it would provide me with a small extra source of income from engraving the gadgets of others. Rapid prototyping circuit boards would allow me to take my work to the next level, rather than having to spend several hours etching a PCB just to find out one trace is routed incorrectly. This laser, combined with help from the awesome community here on Instructables, would help me so much in my pursuit of a career in engineering, my ultimate goal.

I'll leave it up to the merit of my Instructable, though, to decide whether I deserve this laser, not the list of sappy thing's I'd do with it. :D Instructables and Epilog have done something great here, and the projects created by this fantastic contest will set a standard of quality for years to come.

I hope you've enjoyed reading about this project as much as I enjoyed making it. Make sure to leave a comment if this has inspired you to build anything, I'd love to see what you've made!



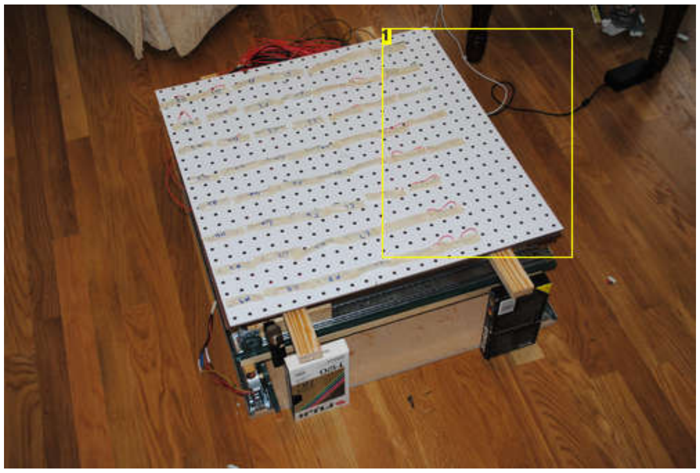


Image Notes
1. The sensors are not completed in this photo but need to be before putting on the chess board.

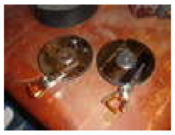
File Downloads

Chess.zip (10 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Chess.zip']

Related Instructables

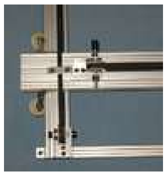
Sandbox Project is now Online (Photos) by CarlS

Steampunk-Inspired Hardware Chess Set (video) by sparkyrust


Hard Drive Fridge Magnet by Everfalling

Build Laser Cutter (Photos) by fishy8082

Homemade 2'x4' Wood CNC Router by Kyles Woodworking

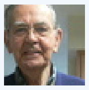

Internet Arduino Controlled T-Slot XY Table by CarlS

Full Contact Chess by KRA5H

Chess Board Safe With Hardware Chess Men (Photos) by Data643

SITWAY

by mickydee on February 7, 2012



Author:mickydee
My name is Roland MacDonald, my friends call me Mac
I am a retired but not bored engineer
My great joy is my workshop, a two car garage with heat and air.
I am a private pilot with 1800 hours flying time
I have built four Kit planes and restored two Cessna aircraft
My last contact with electronics was in the vacuum tube world.
I am really enjoying the new transistor world.

Intro: SITWAY

You are never to old to learn and try new things. I think one of the best days in my life was the day I discovered the Instructables web site. It opened up a whole new world to me. This is my third instructable . I really enjoy building anything that I can ride on or get in to.
I bought an Arduino Uno and was planning on building a balancing Robot. I was really impressed with the Balancing Skate Board that was published by Xenon John. It had most of the code that I would need to build a balancing something. That something evolved from a Robot to a Sit Down Segway clone, which I named the SITWAY. I want to at this time thank John for all the help and patience he showed me in building and testing this ible.

This is my second project involving a discarded electric wheel chair. The motors have great torque and are very reliable. They use 24 volts and have great range using two U1 type garden tractor batteries. You can't go any cheaper than that.

The build went pretty smoothly. Thankfully Xenon John pitched in and helped me modify his code to work with my wheel chair motors.
After running all the tests I felt were needed I elected to have a young neighbor take the first ride. It turned out to be a real blast. So far eight or ten people have ridden it, the youngest being 12, and the oldest 81 (me). The training wheels limit the speed by limiting the forward tilt. I plan to keep the rear training wheels on permanently because I don't need a lot o speed going backwards.

The SITWAY appears to be pretty safe, but it does not have all the built in backup systems that a real Segway has, I have only tested it on my smooth driveway at this time. I have driven over small objects, and it still stayed stable..Any one can learn to drive it in about 5 or10 minutes. With all the testing and driving we have done I have yet had to charge the batteries. The original wheel chair had a published range of 20 miles.. HAVE FUN!!!

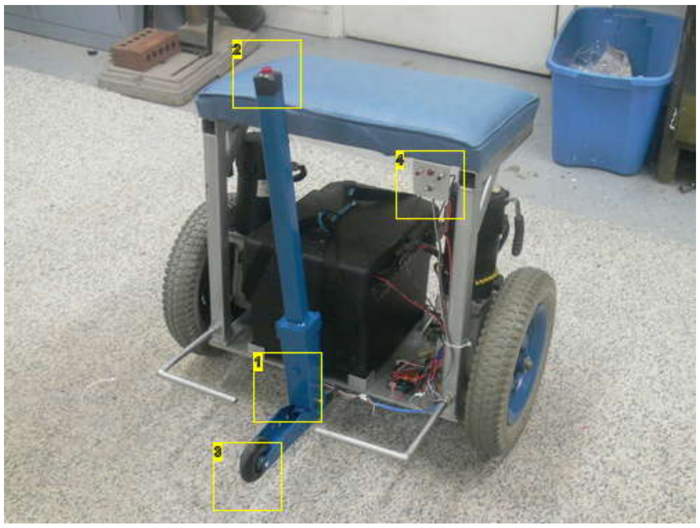


Image Notes
1. Pivot bolt. and tapped screws to adjust micro switch contact points
2. Dead man switch
3. Adjustable front wheel
4. Main power and balance trim switches

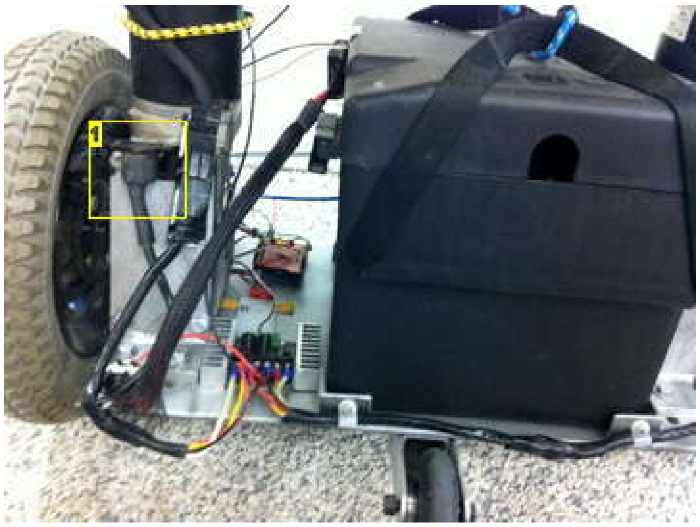


Image Notes
1. Optional 24 VDC charging plug

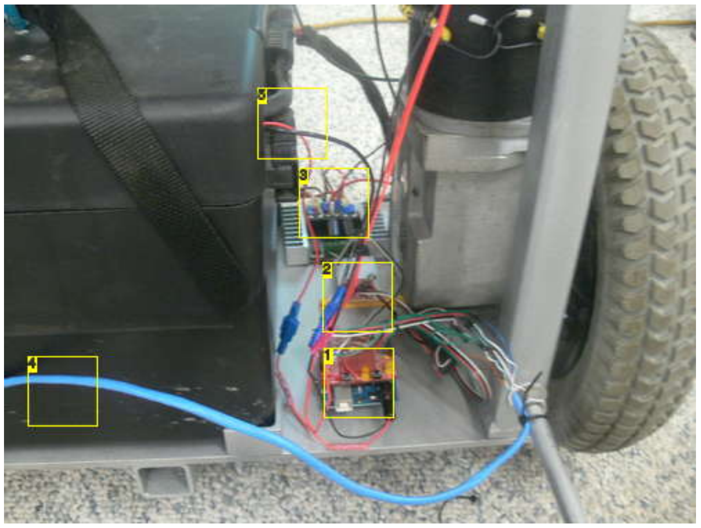


Image Notes
1. -Arduino Uno and shield
2. 5 degrees of freedom IMU
3. Sabertooth 2X25 Motor Controller
4. Cable to hand controller
5. 12 VDC tap from batteries

Step 1: MATERIALS AND COSTS

Item Source Cost

1. Donor electric wheelchair Various places $50 to $200 Depending on condition
2. Arduino Uno Maker Store $30
3. Arduino Uno Proto-shield Spark Fun $15
4. 5 degrees of freedom IMU Spark Fun $49
5. Sabertooth 2X25 Dimension Engineering $129
6. Two surface mount LED's Radio Shack $1.29
7. Two Momentary contact switch's (trim) Radio Shack $1.29
(normally open)
8. Two micro switches for Steering Radio Shack $3
(normally open)
9. Single pole single throw power switch Radio Shack $1
10 13X20X1/4" plate for base Local $10
11. 1/2" steel tubing for seat frame Local $10
12. Vinyl and foam for seat Local $4
13. Asst hook up wire Local $3
14. One can spray paint Local $3
Total $309 to $459

(note) I actually bought my used wheelchair for $35 at a yard sale
It was pretty beat up but the motors were good and even included
A 24 volt charger.My project cost less than $300.

Step 2: Salvaging parts from the donor wheelchair

Save the motors and drive wheels, they are usually one piece. Also salvage the connectors and wiring from the motors to the battery., leave the leads as long as possible. Keep the electronics if you want them for future use. Mine were trash. I never throw away wheels, they always come in handy. Most wheelchairs have two castoring wheels for steering, and two for stability when getting on the chair. Save the two stability wheels for training wheels on the project. Save the battery box and battery cover, you will use these. Dis-card the rest of the chair unless you think you will have use for it in the future. I threw most of it out to reduce clutter.

Step 3: Build the frame and mount the wheels and motors

I guess you could make this frame out of plywood, but I like to use steel. it's a lot stronger and welding is a lot of fun. The base is a 1/2" plate measuring 12X20". The uprights and seat frame is made of 1" steel tubing. Don't forget the 45 deg. braces in the corners. the four holes in the seat braces are for mounting the plywood seat support. The holes for mounting the motors are slotted about 6" long . At this point you do not exactly know where the C.G. will be. The motors can be adjusted fore and aft to adjust the C.G.The small wheels are used as training wheels and to keep the machine from falling over when not in use. Now is a good time to paint the frame You can make the frame any size you want. I designed this one to fit through an interior door. Cut the seat from 3/4" plywood . Pad and upholster a seat cushion to be bolted to the frame uprights.

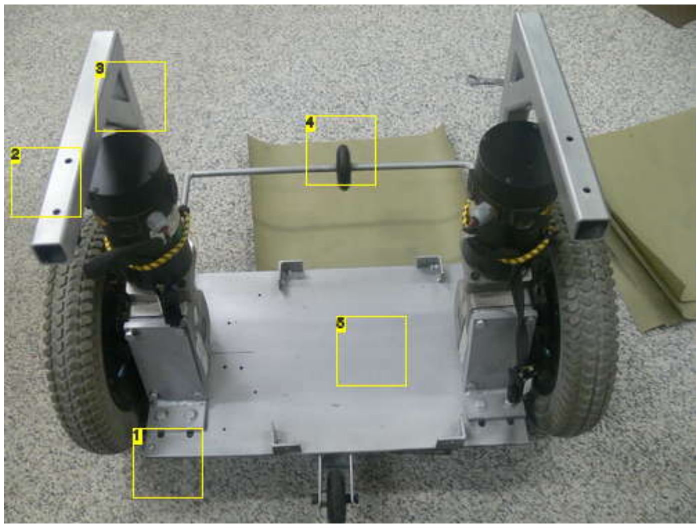


Image Notes
1. 6" slotted holes to mount motors and adjust the C.G.
2. holes to attach seat support
3. 45 deg. braces
4. foot rest and leveling wheel
5. battery position

Step 4: STEERING CONTROLLER

I originally used John's hand held steering controller, but found it was not practical for a sit down balancing machine. You need something very rigid to hang onto when riding. The machine does a good job of balancing if you don't fight it by trying to balance it your self..
The stick is rigid in the fore and aft position , and will move side to side in the lateral position. a compression spring centers the stick.
Two set screws provide stops, and two more act as limit switches for the two micro switches that control the steering.
This could be done simpler by mounting the micro switched on the outside of the stick, but I wanted to have them hidden inside the stick.
I have access to a vertical mill and I never miss a chance to use it. The micro switches are available from Radio Shack for 3 or 4 dollars.

The adjustable front training wheel serves two purposes. first it keeps you from pitching forward during any sudden stops, and secondly it limits the forward speed by limiting the pitch angle until you get comfortable with riding it. The rear wheel is fixed limiting reverse travel to a safe speed.

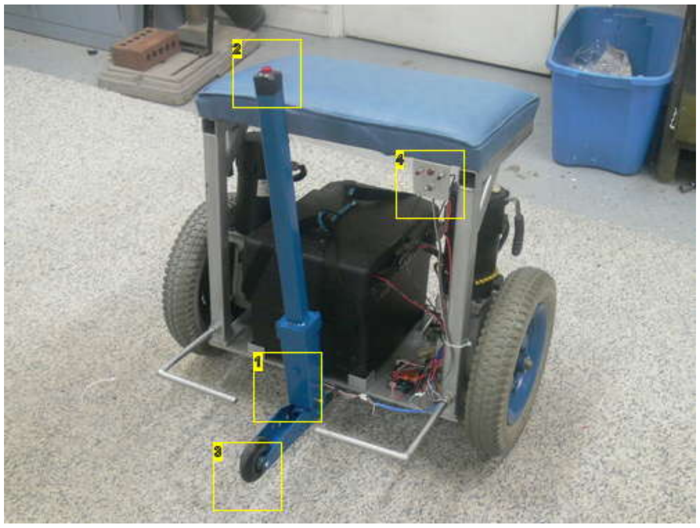
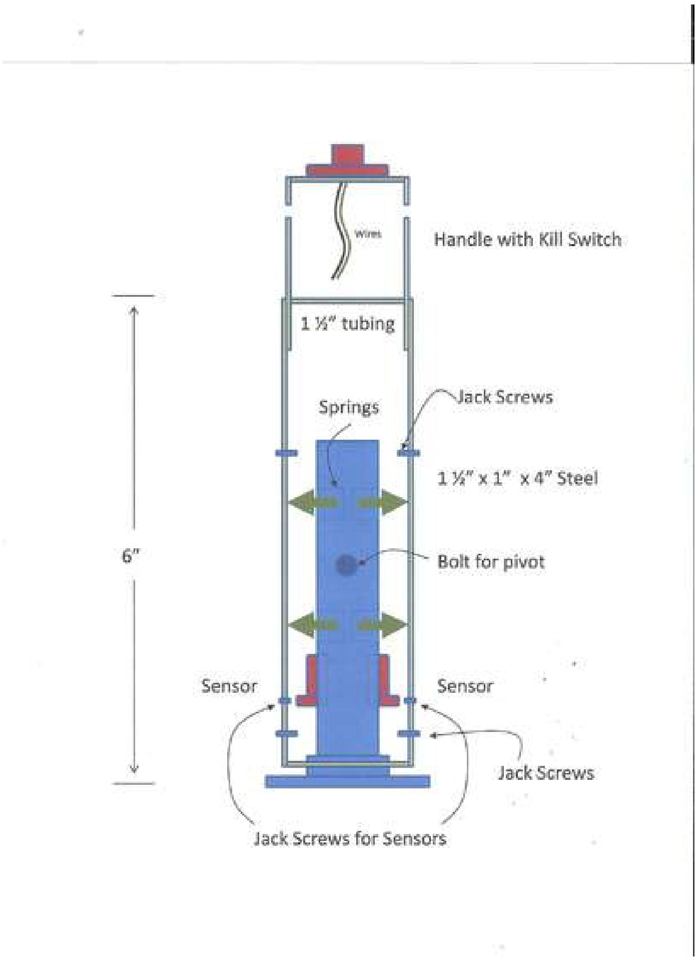


Image Notes
1. Pivot bolt. and tapped screws to adjust micro switch contact points
2. Dead man switch
3. Adjustable front wheel
4. Main power and balance trim switches



Step 5: ELECTRONICS

The electronics consist of the following
Arduino Uno
Shield
Sabertooth 2X25
5 Degree's of freedom IMU
Asst. LED's and switches
10K pull down resistors (5)
4 conductor cable and hook up wire

I don't like to solder directly to my Arduino. Instead I used a shield. This allows me to make solid solder connections instead of plugs that can come loose due to handling or vibrations.

A good place to begin is to solder the (5) 10k resisters to the shield. These are the pull down resisters for the balance trim, steering, and dead man circuits.

(Note) The following wires are connected to the Arduino digital pins
pin 9 is for the dead man switch circuit
pin 7 is for nose down trim circuit
pin 6 is for nose up trim circuit
pin 5 is for steer left circuit
pin 4 is for steer right circuit
The other end of the resisters goes to circuit ground
pin 13 connects to the S1 input of the Sabertooth Motor Controller

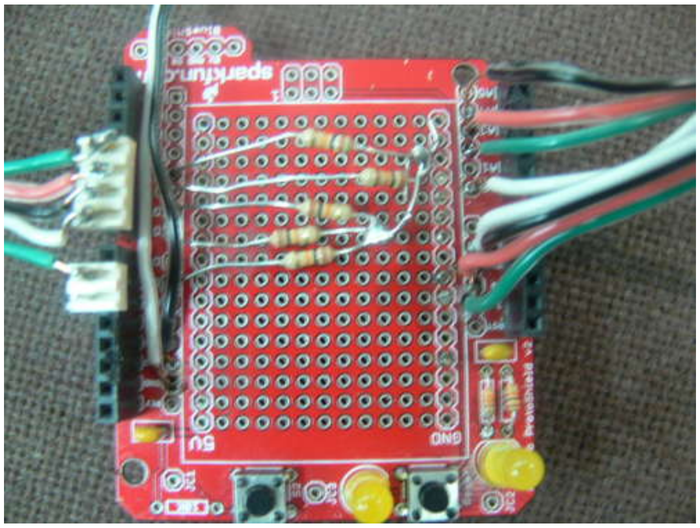
The following wires are connected to the Arduino Analog pins
pin 0 to Y Rate 4.5 on the IMU
pin 2 to X Rate on the IMU
pin 3 to Y Rate on the IMU
pin 4 to ZACC un the IMU
+5 volts to he Steering controller
+3.3 volts to the IMU (NOTE) do NOT apply 5 volts to the IMU
GND to the IMU

All the Analog connections can be soldered directly to the shield.
The Digital connections can be made either by plugging directly into the headers or using a connecter. I found some in the Sparkfun catalog that fit snugly into the headers (See above pic)

I found some four conductor ribbon cable at Radio Shack that worked well for me. It is stiff enough to hold it's shape, and the color coding makes life easier. You can use ribbon cable from an old computer just as well except for the color coding .

Mount the IMU to a small block either wood or phenolic to the floor of the machine at approximately the center line of the axles.
Be sure to mount it correctly.
The factory drilled mounting hole must be pointed towards the ground
The component side of the IMU must face forward
If you mount the IMU incorrectly bad things will happen.
The plate can be shimmed fore and aft to adjust for level balance

Fashion an aluminum plate to hold the power switch, and the trim and steer switches and mount it to be reached conveniently while seated. I installed a power indicator LED (with a Pull down resister) to verify Arduino power (12 volts)



Step 6: WIRING

Generally wheelchairs operate on 24 VDC.. Main power is obtained from two U1 type lead acid batteries. They are used in lawn mower or lawn tractors. They are cheap ($40) and easy to get .
There are four voltages used in this project
24 VDC for the motors
12VDC for the Arduino Uno
5 VDC for the steering and trim circuits
3.3 VDC for the IMU
24 VDC is connected to the Sabertooth 2X25 . A power switch is installed in the negative leg. Be VERY careful to maintain the correct polarity.
The Sabertooth will be permanently damaged if you hook it up with the wrong polarity. The warranty will also be voided.
The 12 VDC is obtained with a tap between the two batteries terminated with a plug for the Arduino power input. Do not use the USB circuit as a power source, strange things happen to the gyro when you use the USB for power.

The 5VDC and 3.3VDC is obtained from the Arduino Uno.
A 5 volt source powers the dead man, left and right turning, and both trim switches.
3.3 volts from the Arduino powers the IMU

Bolt the Sabertooth, the IMU mounted to a block, and the Arduino Uno and it's shield to the floor of the frame
Connect the motors to the Sabertooth. The left motor connects. to M1A and M1B. The right motor to M2A and M2B.
S1 on the Sabertooth connects to Arduino pin 13. Connect Sabertooth ground to Arduino ground. This completes the Sabertooth wiring for now. These connections will be verified during the Motor test procedure a little later. Be sure and set the Sabertooth DIP switches for Simplified Serial operation. Set switches 1,3,5, and6 to the on position Switches 2, and 4 are set to the off position. These settings support using lead acid batteries.

Route the five wires from Digital pins 4,5,,and 9 plus a 5 VDC source to the steering handle.
Connect the 5 volts to one side of the two momentary on steering switches and to one side of the dead man switch.
Connect the other side of the switches to the wires coming from Arduino pins 4, 5, and 9.

Route the wires from Arduino pins 6, and 7 to the little plate with the trim switches. Connect one side to 5 VDC and the other to Arduino pins 6, and 7.Route the wires from the main power switch to this plate and attach them to a 40 amp SPST switch. Install a surface mounted LED indicator and power it with 5VDC to ground through a 10k resistor.

Install the batteries using the battery box or covers you salvaged from the wheelchair and this pretty much completes the basic construction of the project.. If you saved the charging plug from the wheel chair, install it in a handy place and wire it up to the 24 VDC source. Pad and upholster a seat cushion to be bolted to the frame uprights.

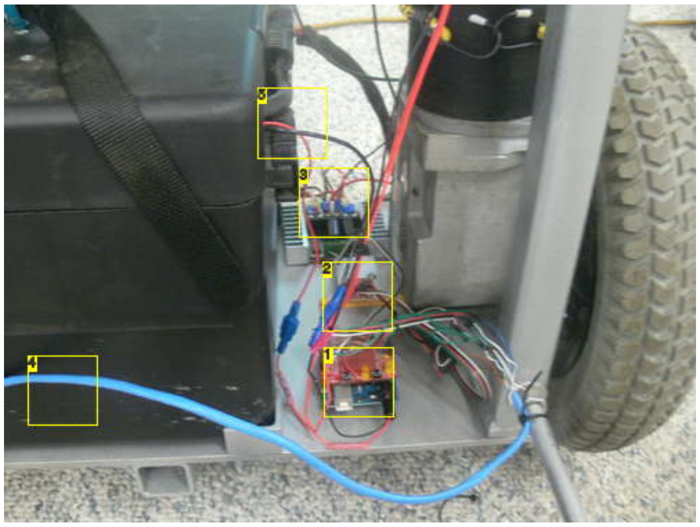


Image Notes
1. -Arduino Uno and shield
2. 5 degrees of freedom IMU
3. Sabertooth 2X25 Motor Controller
4. Cable to hand controller
5. 12 VDC tap from batteries

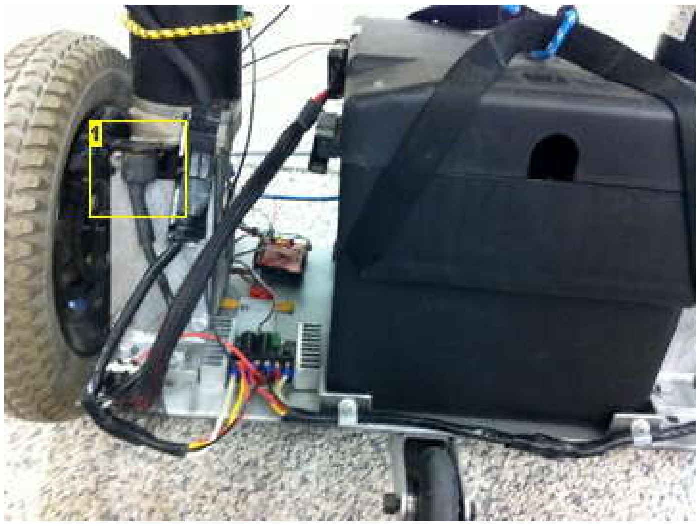
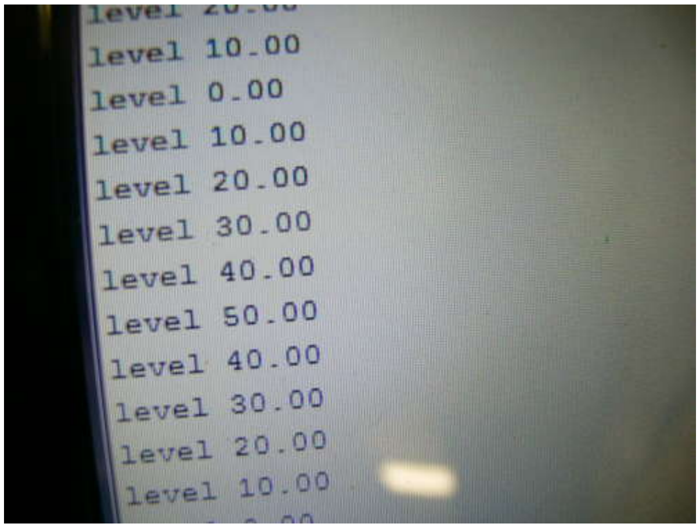


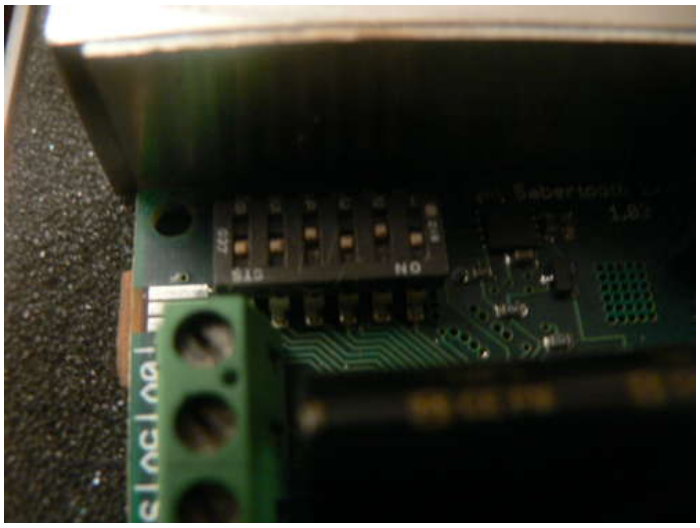
Image Notes
1. Optional 24 VDC charging plug

Step 7: MOTOR TEST

Now that the wiring has been completed, it is time to test the communication between the Arduino and the Sabertooth.
IMPORTANT!!! Maker sure the Sabertooth DIP switches are set for Simplified Serial operation.
Raise the machine up on blocs high enough for the wheels to clear the floor and can spin freely.
Open the Motor Test notebook sketch. Copy and paste this sketch to a new Arduino sketch. Upload the new sketch to your Arduino.
This sketch only uses digital pins 9 and 13. (Dead man and serial input to the Sabertooth)
Turn on the main power switch and depress the Dead man switch. Both motors should start turning in the same counter clockwise direction. If they don't, reverse the wires going to the M1 or M2 connections on the Sabertooth. Both motors should slowly increase in speed in 10% increments from stop to 50% and then from 50% to stop.



```
level 20.00
level 10.00
level 0.00
level 10.00
level 20.00
level 30.00
level 40.00
level 50.00
level 40.00
level 30.00
level 20.00
level 10.00
```



File Downloads

new_motot_test.txt (5 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'new_motot_test.txt']

Step 8: THE FIRST TEST RIDE

Now that you know that the motors are turning in the right direction, and communicating with the Sabertooth, it is time to upload the full rocker switch code to the Arduino.. Copy the full test sketch into a new Arduino sketch and upload it to your Arduino.
This is a good time to exercise extreme caution as these motors are very powerful and can run across the room and do bad things.
This code makes use of a TIP START routine that lets the machine start gently at first and then after about 5 seconds comes up to full power.. With the machine tipped to its parked position turn on the master switch. Count to five slowly. This allows the IMU to calibrate itself.
Next depress the dead man switch. (Note) I installed an LED to light up when pin 9 the dead man circuit is activated. Slowly bring the machine to an approximately level position. If all goes well the machine should balance it's self. Any time you sense a problem let go of the dead man and the machine will stop. If you De-activate the dead man you will have to start from the beginning and go through the TIP START routine again. Make these first tests with out getting on the machine. Tip the machine forward and backward a little and see if the machine tries to move accordingly Try a gentle turn in both directions. If all goes well you are ready for the first test ride.

Do not take the first ride when you are alone. Have some one there to help you if something goes wrong. At my age my bones are getting pretty brittle so I let my best friends teen age son do the honors. We made him wear a helmet and we brought him up to level after the TIP START by hand. He did a great job and mastered the machine in just a couple minutes.
For this first test we used the hand held controller that Xenon John uses in his skate board. It became obvious that for the SITWAY we would need a different control method. I built the new joy stick out of 1" steel tubing. Like the original Segway you need something to hang onto. The joy stick houses the two steering switches and the dead man circuit. The stick does not move in the fore and aft plane, the only movement is from side to side. This system really works well and has not given me any trouble. I moved the trim switched to a little aluminum plate mounted whitin reach of my left hand.

File Downloads

full_balance_rocker_test.txt (25 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'full_balance_rocker_test.txt']

Recent_mild_setup_code.txt (29 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Recent_mild_setup_code.txt']

Step 9: ADDING 3D PRINTED OBJECTS

To spruce up the project I decided to add some caps and plugs to the frame to make it look a little better' The deadman switch is round and did not fit very well into the square hole of the joystick.. I designed a cap to attach it using "123D". I also printed out four plugs to cover the holes from the 1: tubing forming the frame. Not really necessary, but it looks good.. I am always looking for good uses for the 3D printer.

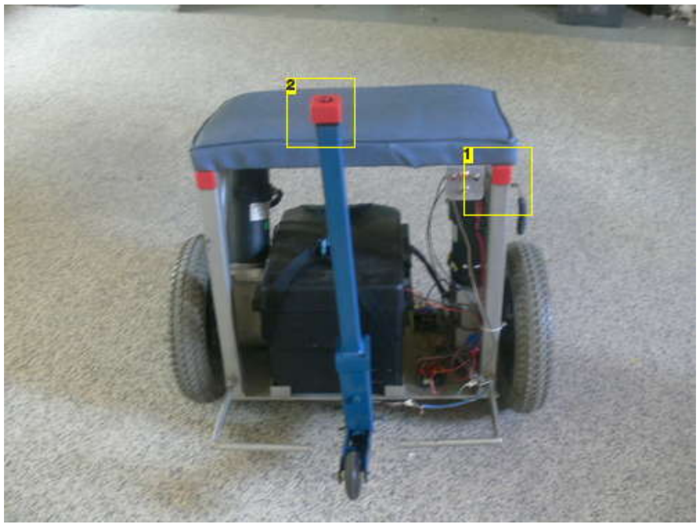


Image Notes
1. 1" tubing
2. Deadman switch

Step 10: CONCLUSION

I built the SITWAY entirely out of steel, mainly because I had the steel available and I love to weld. It could be built out of wood, but I don't think it would be quite as substantial.. This is an on going project. I am still tweaking the code to get the smoothest performance. I will publish improvements as the are ready. The present code will at least assure that the machine will Balance, Move, and Steer.

This version is a little slow at this time and that is alright by me for now. I plan to raise the front wheel a little for more forward speed.
Eventually I might eliminate the front training wheel all together when my confidence factor builds up.

The most important to learn is that the machine should balance you, not the other way around. Try to remain in a stiff position and let the machine do the work instead of fighting it.

SAFETY SAFETY SAFETY I take baby steps with any changes I make If you try to go too fast, this thing could hurt you. There is a lot of power with these motors. I have not tried to navigate over any large objects as yet. I plan to test it by running it over a 2X4 and see what happens.

It is a fun project. The neighborhood kids are crazy about it and zip around the yard with ease. Thar's the main reason I have limited the speed for now. The wheel chair motors are very frugal with battery use. I have actually run on 12 VDC. It was a little sluggish but operated just fine.

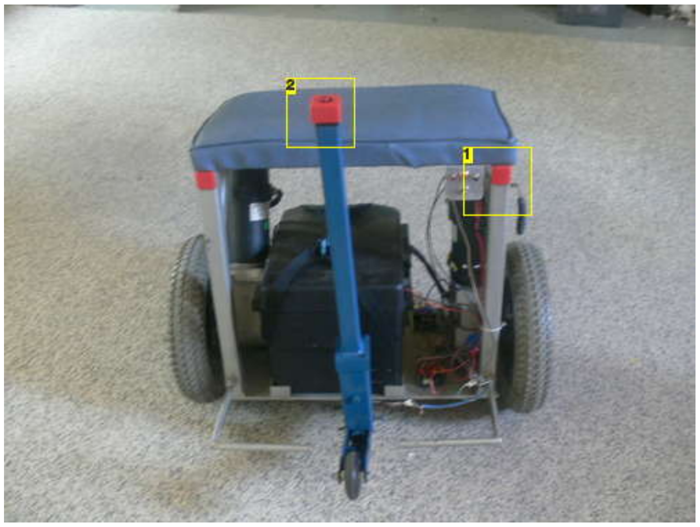


Image Notes
1. 1" tubing
2. Deadman switch

Related Instructables

Easy build self balancing electric skateboard by XenonJohn

Self Balancing Scooter Ver 1.0 by ScitechWA

Balance-BOT (Photos) by daniel2008

Steampunk Segway (Legway) by bdring

Domo Kun WobblyBot, Simple Self Balancing Robot by Chein

LEGO iPod touch dock. by Gun Fun

Seg...stick. by scolton

Polystyrene throw airplane by njacksx

A Makers Wedding - Photo booth

by **letMeBeFranks** on November 5, 2011

Author:letMeBeFranks
Im a interaction designer. I work at frog design :)

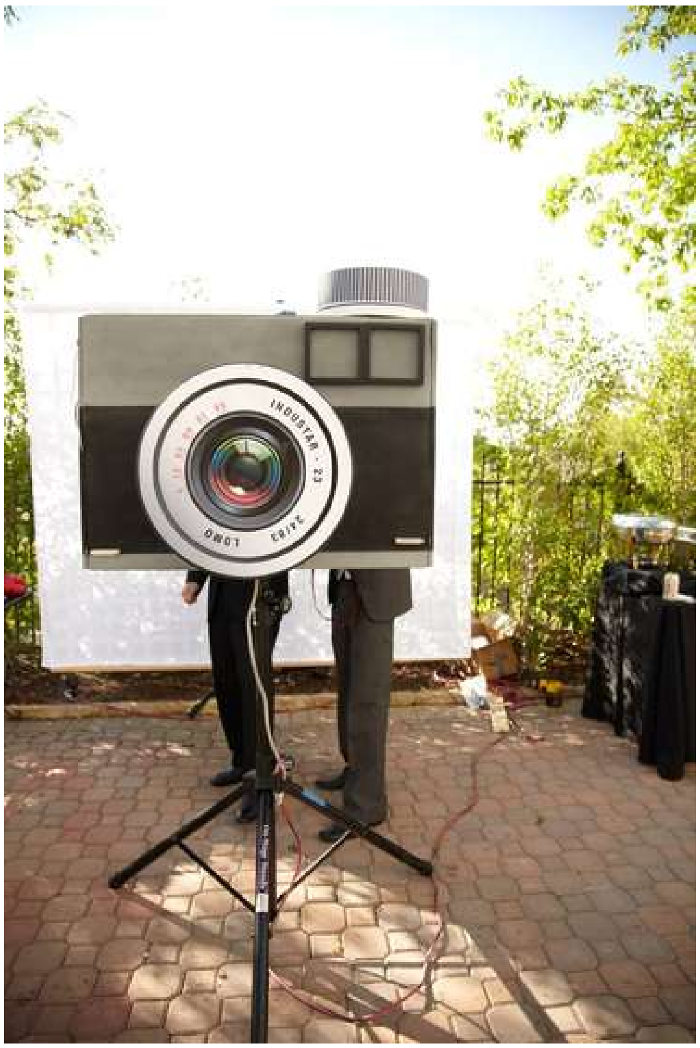
Intro: A Makers Wedding - Photo booth

This Instructable is about:
building an automated photo booth. The total build cost was around $150 as I re-used a lot of the components and materials I already had in my garage - in addition to what I could salvage from scrap yards.

Why? - I decided to build my own photo booth after trying to rent one from local photography studios. The going rate for a rented photo booth is around $600 in addition to the hourly rate of the attendent to watch over the equipment. As this was not in my wedding budget, and I did not want to deal with an additional vendor, I decided to build my own for under $200.

My Goal - Build an automated photo booth for under $200 - that could be easily operated by anyone at a party - and is durable and compact enough to fit into a compact car.

(Note* this photo booth does not print pictures. I have been working on a script that automatically uploaded the photo to flickr, but I did not finish it in time for the wedding. I'll try and include that in a future post)



Step 1: How it works

The photo booth operation is simple. Users walk up to the back side of the camera - See themselves on the screen - Press a button and strike a pose.

The mechanics of the photo booth are a little bit more complicated, but ideally, the user never has to know what is going on under the hood.

The guts behind this photo booth are based on OSX lion. With Lion, the photobooth application can be extended to full screen and it can be set to use an external camera. So I connect a logitech web cam and an external monitor to a laptop running OSX Lion. The only thing i needed to build in addition to this hardware setup was an array of lights and a button (mapped to the enter key) to trigger the photobooth application to take a picture.

The "business end" of the photo booth can be seen in this step.

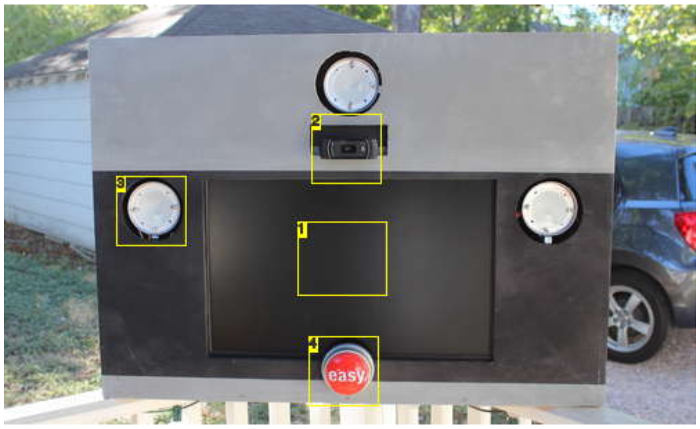


Image Notes
1. LCD Monitor
2. Logitech Webcam
3. Halogen Lights from Ikea
4. Modified Staples Easy Button

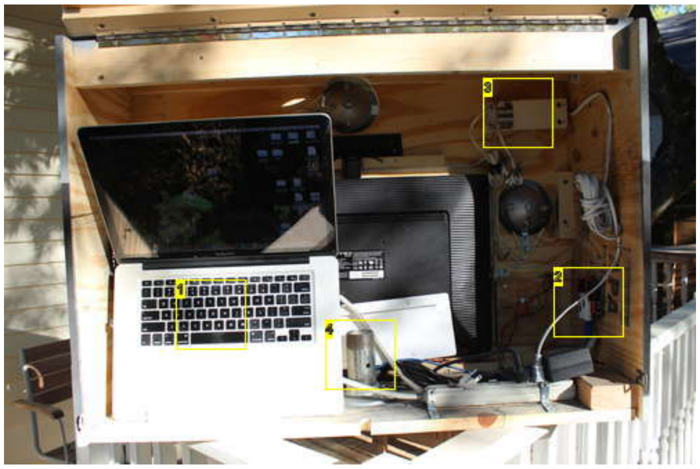


Image Notes
1. MacBook Pro - OSX Lion
2. Arduino
3. Light Controller

4. Tripod Mount

Step 2: Software and Trigger Button

A Brief Overview
As previously mentioned, this photobooth uses the OSX Photobooth application. The OSX Photobooth application was chosen because it was the most stable software i could find - and it comes with every MAC computer. Like most applications, users can trigger features and functions with mouse clicks and keyboard commands.

Triggering the Photobooth Application
With OSX Photobooth, pressing the Enter Key triggers the program to take a photo. I didn't want to expose my computer to people hammering on the keyboard (espeically if they had been drinking). This is why i decided to use an external button, connected to an arduino microcontroller, to trigger the photobooth application.

This is how it works:

The button is pressed - A Staples Easy button was modified to act as a regular button. It's really durable, so people can beat on it without breaking it.

An Arduino registers the button press - When it registers a button press, it sends a serial command to the computer. In this case, it sends the [enter] serial command.

AAC Keys listens to the serial port for serial commands - AAC keys is a free application which litens for serial commands and emulates mouse and keboard events. You can download it here. In this case, when AAC keys receives the [enter] serial command, it tells the computer (and the photobooth application) that someone has just pressed the enter key on the keyboard.

When the photobooth application registers the enter key being pressed, it takes a photo.

Wiring the circuit - If you do not know how to make a button circuit for an arduino, read this tutorial - http://www.arduino.cc/en/Tutorial/button

Be sure to connect the button to pin 10 on your arduino. If you choose to wire your button to a different pin, be sure to change [int buttonPin = 10] in the arduino code to match the pin number you selected.

Writing the code - Here is the code i wrote to send an [enter] serial command to the AAC Keys. If you are not familiar with writing arduino code, use this tutorial here. It's pretty easy once you get the hang of it. http://arduino.cc/en/Guide/HomePage

```
const int buttonPin = 10; // the number of the pushbutton pin

int buttonState = 0; // variable for reading the pushbutton status

void setup() {

pinMode(buttonPin, INPUT);
Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop(){

buttonState = digitalRead(buttonPin);

if (buttonState == HIGH) {
Serial.println();
}
else {
// nothing
}

}
```

Installing AAC keys - As previously mentioned, AAC keys is a free program. "That receives commands through your computer's serial port and translates them into keystrokes and mouse movements, giving you full control of your computer from another device such as an [arduino]". You can download the program here: http://www.oatsoft.org/Software/aac-keys

Using AAC Keys is quite simple. Make sure you have an arduino plugged in via usb, running the code seen above. Open AAC keys application and access the applications preferences. When the dialogue appears, check to see that you have selected the serial port associated with the connected arduino (generally it's selected by default, but it is good practice to check), and that it is running at 9600 bps.

If you've done this, AAC keys should be interpreting the button press from the arduino as an [enter] command on the keyboard. open a text editor and give it a shot. Type a few lines of text and press the button attached to your arduino instead of using the enter key. You can also open photobooth at this time and see that pressing the button triggers the program to take a picture.

Image Notes
1. Testing the connection between the easy button and the computer.

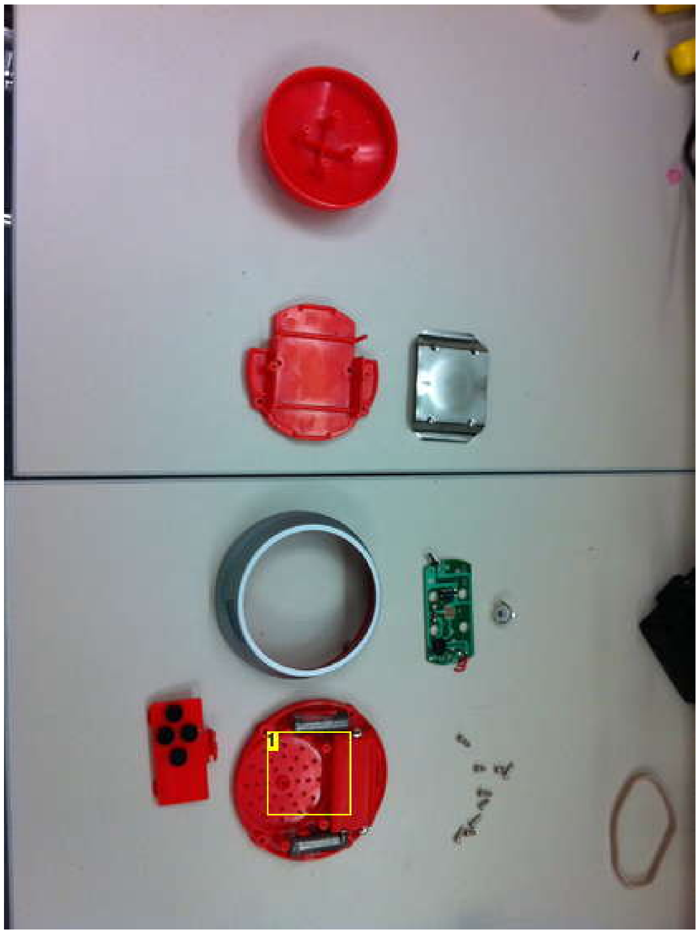


Image Notes
1. Disassembled Staples Easy Button

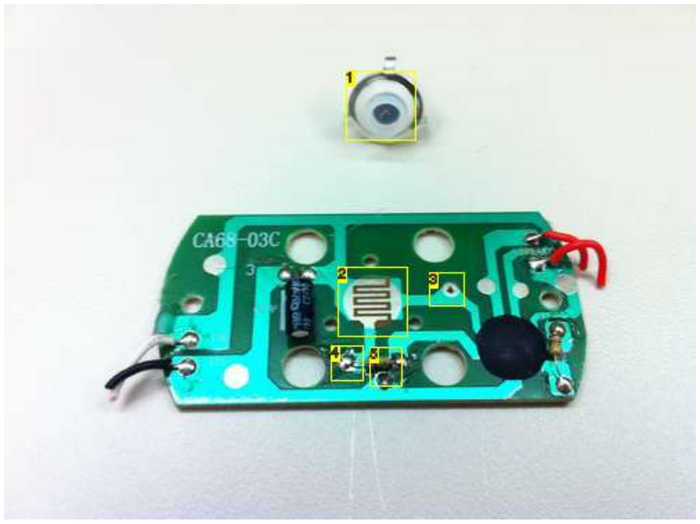


Image Notes
1. Rubber button which makes contact with the circuit board

2. Circuit board from the staples easy button. Inorder to use this button to trigger an arduino, Solder wires to the following locations that are labeled in this image.
3. Solder one wire here
4. Solder Another Wire here
5. Cut this resistor off the board. This prevents any errors (false trigger events) as this pathway on the circuit board is still connected to the original microprocessor (seen as the black dot in this picture)

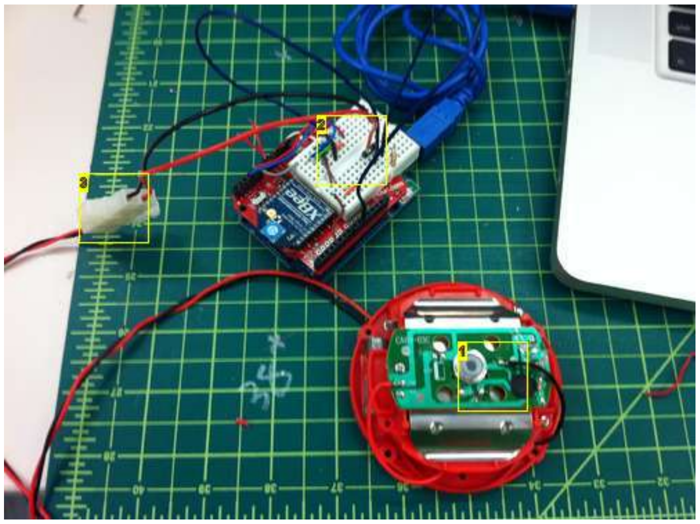
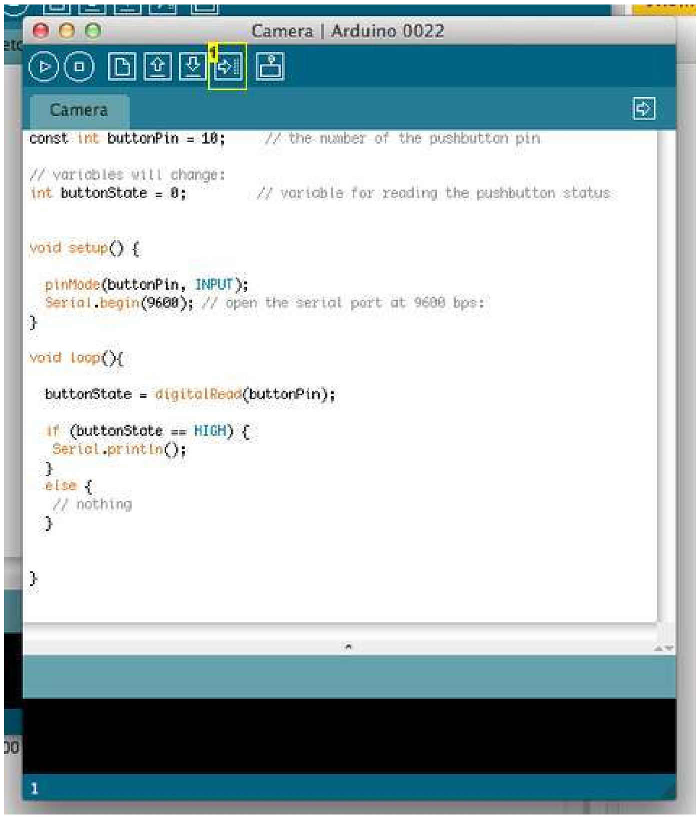


Image Notes
1. wires soldered to the circuit board from the staples easy button. These wires lead back to the breadboard on an arduino
2. Arduino registers when the button has been pressed. It then sends a serial command to the computer.

The button is connected to Pin 10 on the arduino
3. I added a quick disconnect so that i could disconnect the button from the arduino when it came time for installation.



```
const int buttonPin = 10;     // the number of the pushbutton pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status


void setup() {

  pinMode(buttonPin, INPUT);
  Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop(){

  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
   Serial.println();
  }
  else {
   // nothing
  }


}
```

Image Notes
1. This is the arduino code in the editor. Click this button to send the code to the attached arduino

Step 3: Booth Design

The photo booth was modeled after an old school LOMO camera, instead of the traditional box with a curtain, for three reasons:

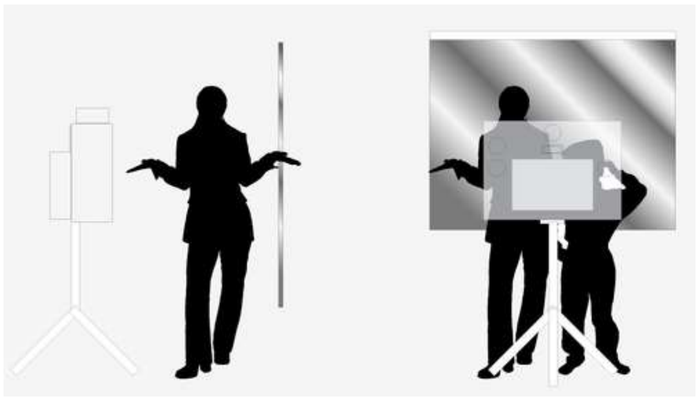
- Ease of construction - Its basically a box with a fake lens on it
- Recognition - People will be able to see a large camera from far away, and they might easily deduce that it must take photos in some form or fashion.
- Novelty - The accentuated size of the camera will create a conversation piece in addition to eliminating the fear of social contract (the users fear of approaching and using it without permission or instruction)

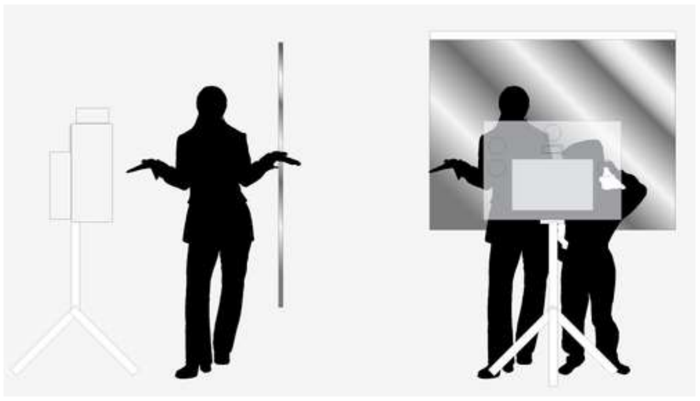
After settling on the design, I sketched it out in adobe illustrator (the .AI file is included on this page). Illustrator is a good tool to make quick "blueprints" which can be easily scaled and printed. if you don't already own adobe illustrator, you can get the demo right here http://www.adobe.com/cfusion/tdrc/index.cfm?product=illustrator

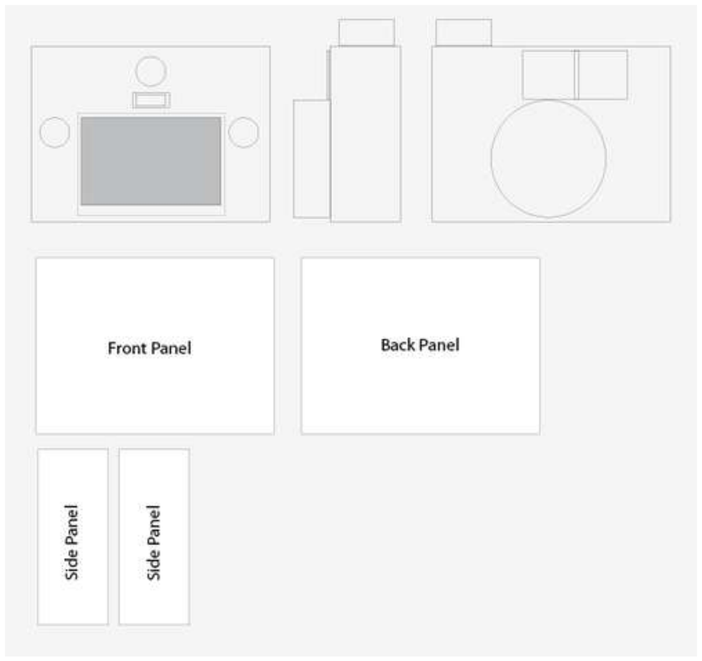
With this quick blueprint, I created a front and side view of the camera. The size of the camera is dictated by the size of the LCD monitor and a a varying height range of users - something that would be easily accessible for people who are 5'2" - 6'4".

After completing the design, I measured the size of the panels and started building.

The Adobe Illustrator file (.AI) is attached to this page.







File Downloads

Camera.ai (1 MB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Camera.ai']

Step 4: Cut The Panels

The panels were cut out of 1/2" Plywood. They are thin enough to be light weight, but thick enough to screw / nail into without worrying about additional reinforcements inside the case.

Image Notes
1. Each panel was cut out and labeled so I didn't lose it or accidentally chop it up for another project.

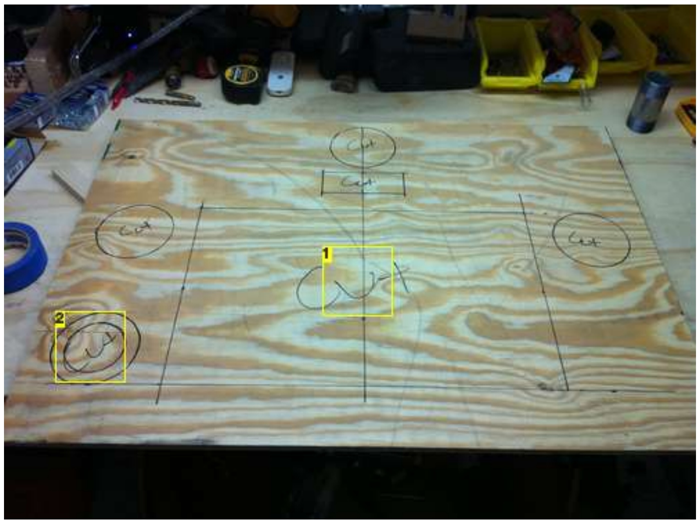


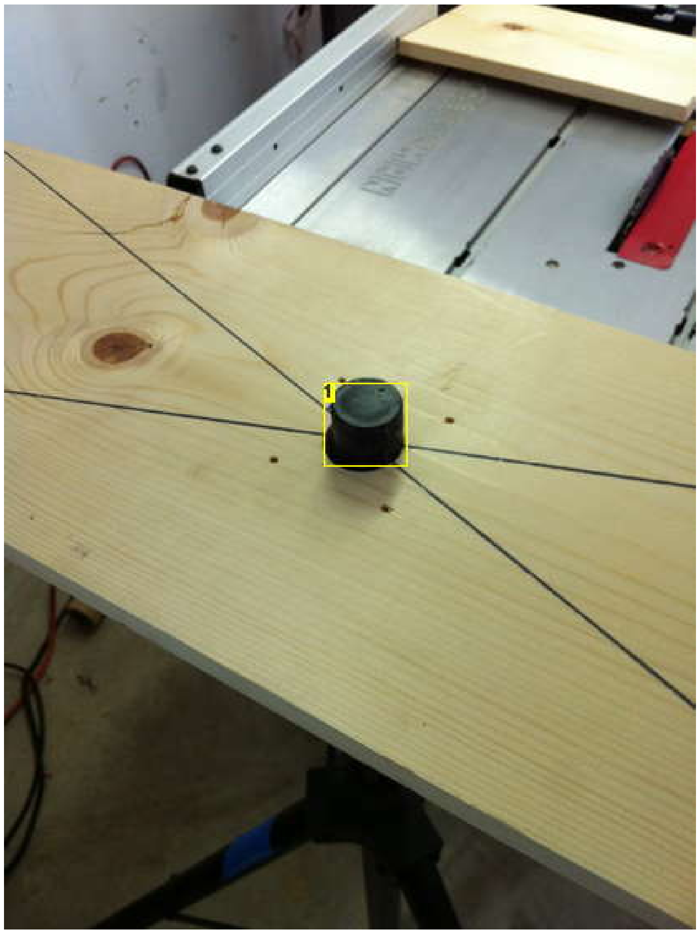
Image Notes
1. The holes to be cut in the back panel. Their size and placement were checked against the illustrator blueprint.
2. Originally the button was placed to the side of the screen. Unfortunately this placed an odd amount of torque on the mount when it was pressed, causing the camera to spin. It was later moved to the center, just under the screen.

Step 5: Bottom Panel - Tripod Mount

The camera body was designed to mount onto a heavyweight tripod. Inorder to make it stable, I had to provide an additional support in which the tripod tube could slide into and lock.

I borrowed this tripod from a friend. it has a 1 1/4" Outer Diameter tube which i used to mount onto. Home Depot sells galvanized pipe with an Inner Diameter that is slightly larger than 1 1/4" in addition to the necessary surface mounting hardware.

The galvanized pipe has a hole drilled into it, allowing me to insert a pin (which intersects with holes in the tripod tube), creating a stable mount that can only be released from inside the camera box.



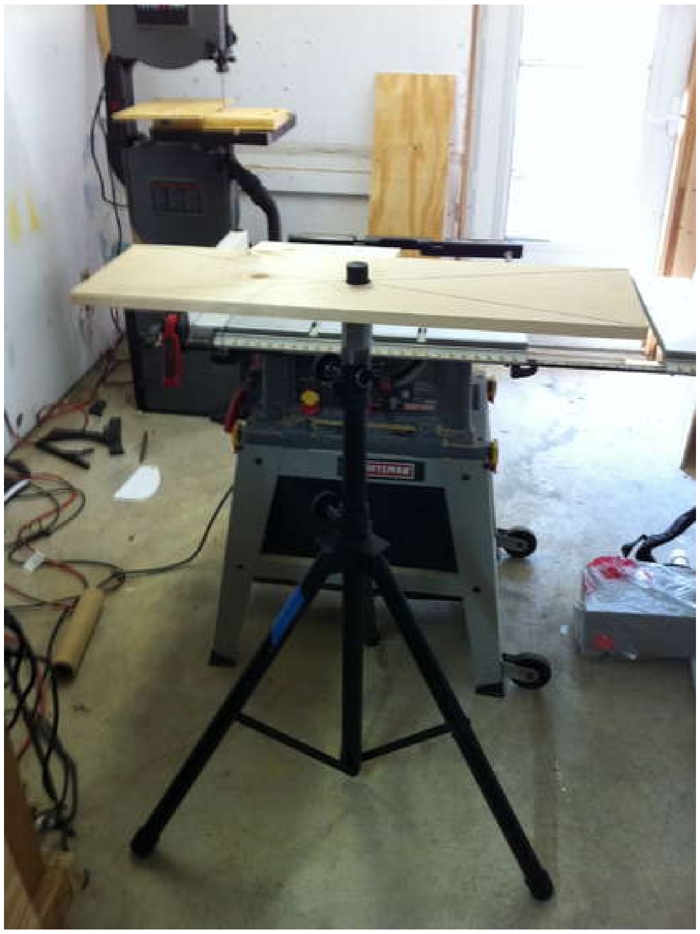


Image Notes

1. A 1 1/2" hole is drilled in the center of the bottom panel

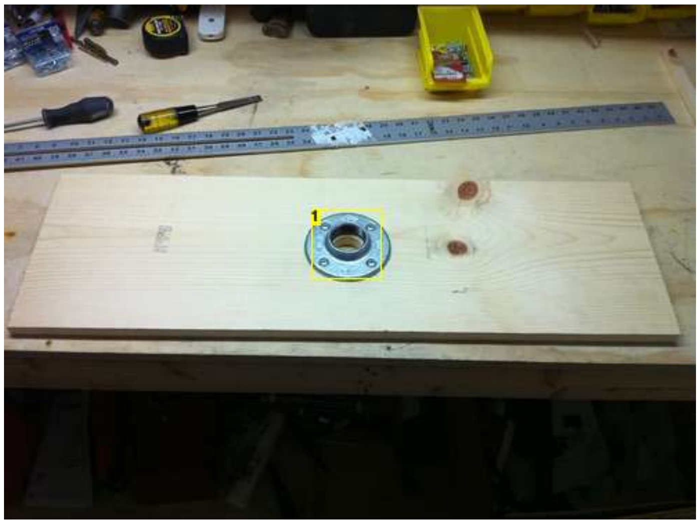


Image Notes
1. Surface mounting hardware is added so that the galvanized pipe has something to secure to

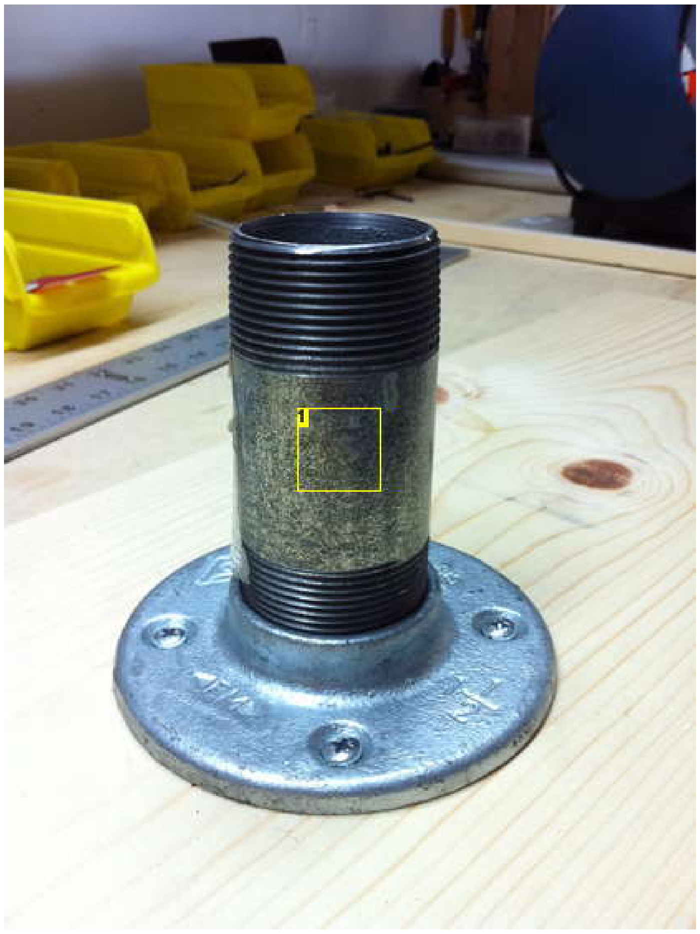


Image Notes
1. A hole is drilled into the galvanized pipe so that a pin can be inserted into it

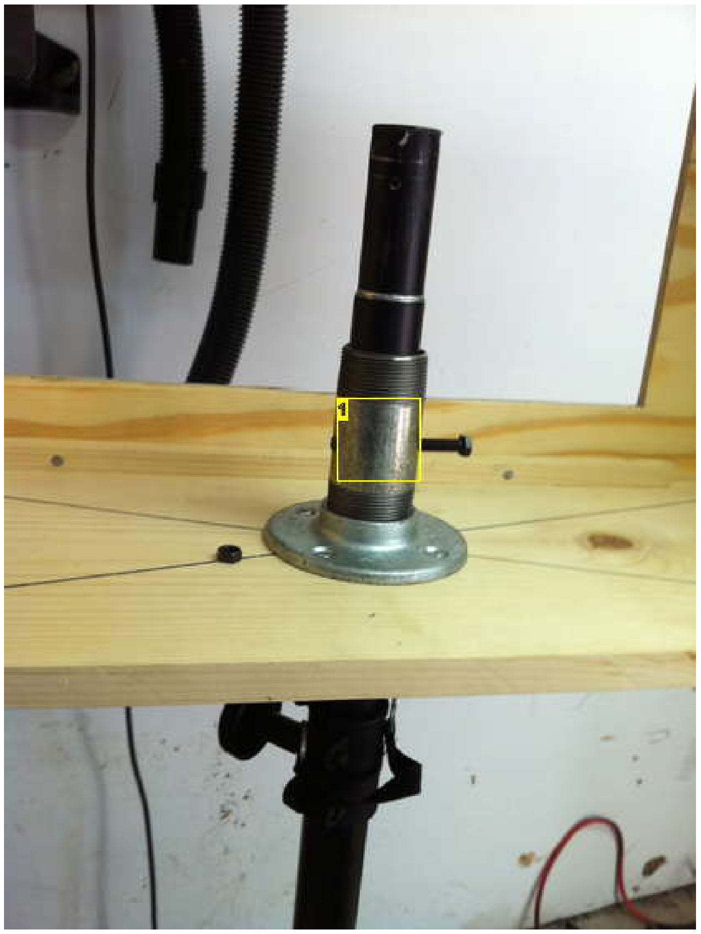


Image Notes
1. The pin is inserted into the pipe, and through the tripod tube. The entire assemble is now safe and secure

Step 6: Box Construction

The camera box was constructed from 1/2" plywood, with a 3/4" Pine base (for added strength).

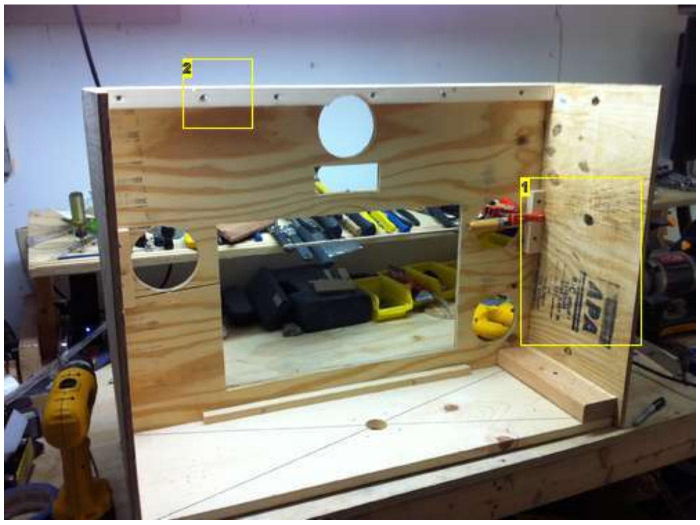
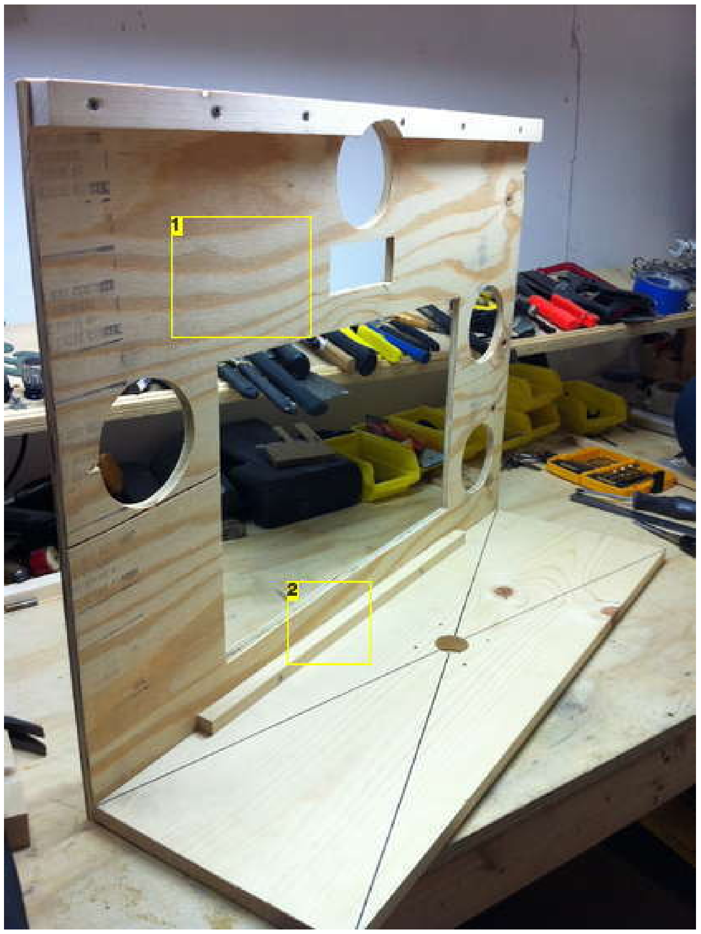


Image Notes
1. The side panels are added
2. This beam is used to give the top panel additional points to attach to



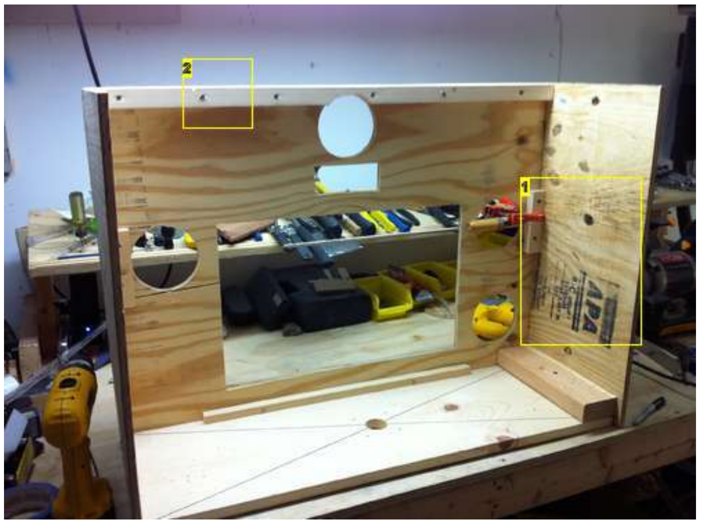


Image Notes
1. The Back of the camera, attached to the base
2. This support raises the LCD monitor to the right height off of the base, keeping it centered in the window

Image Notes
1. This is the door for the front of the box. I added a beam to the top of this panel so that I can secure it to the camera box with a piano hinge.

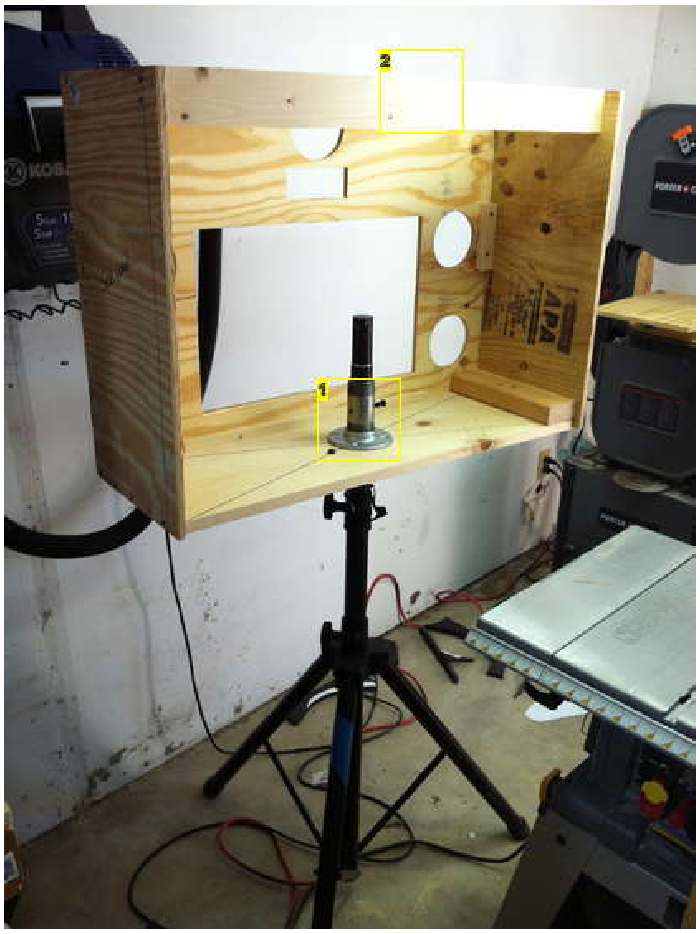


Image Notes
1. The camera box is secured to the tripod
2. This beam provides support between the two side panels, and will also be used to secure the front panel onto via a piano hinge

Step 7: Adding Components

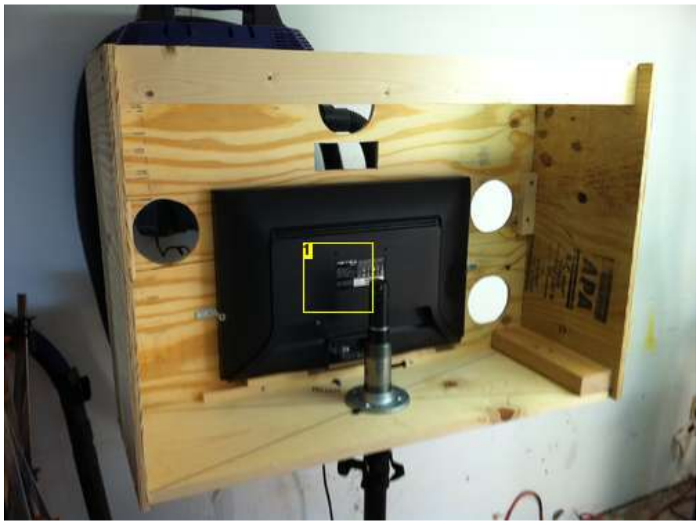


Image Notes
1. The LCD is added to the Camera Box. It is secured by a few L Brackets

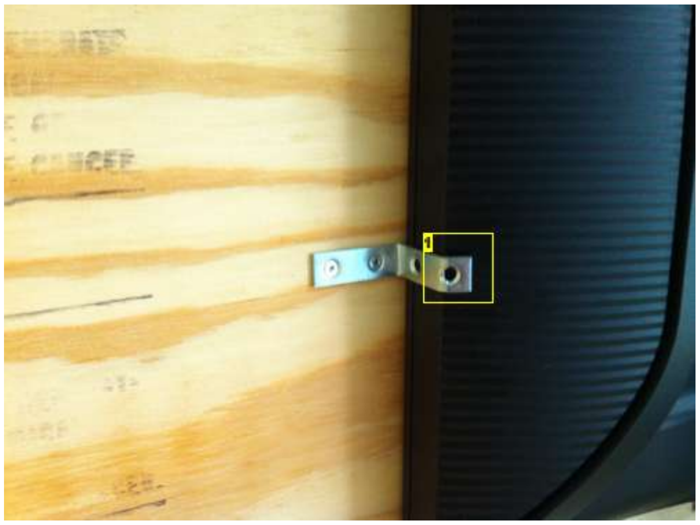


Image Notes
1. The L Bracket is bent at the tip to hold the LCD monitor still.

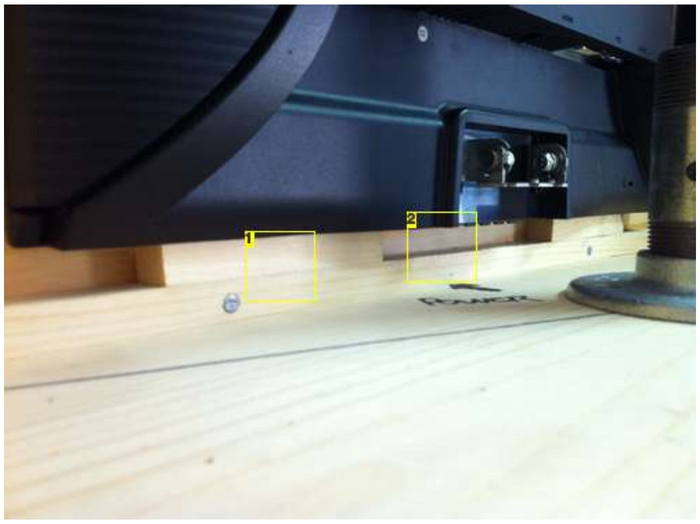


Image Notes
1. Additional shims were needed to center the screen in the cut out window.
2. Be mindeful of where the buttons on your monitor are. you will still need to be able to fit your fingers in here to turn on the screen when its ready to be used

Image Notes
1. A little note so i knew which button to press once the camera was all put together and the LCD monitor is locked in place.

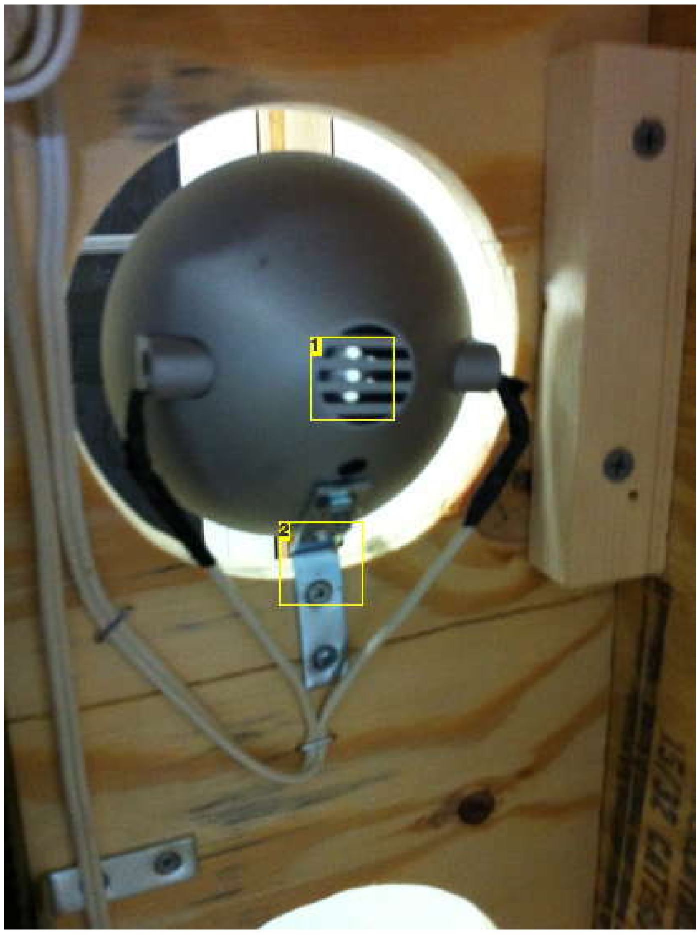


Image Notes
1. Halogen lights from IKEA were modified to attach to the camera box.
2. A bent L bracket is used to secure the light to the box

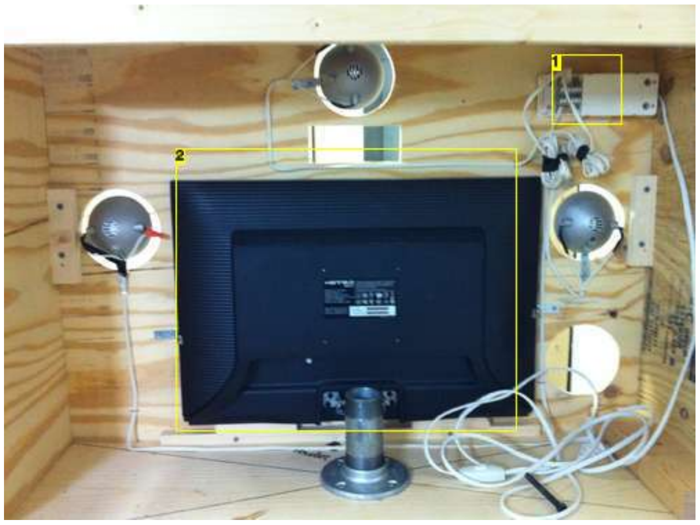


Image Notes
1. The light control box is placed out of the way, and the extra cable is wound up.
2. All of the lights and the LCD monitor are mounted and working.

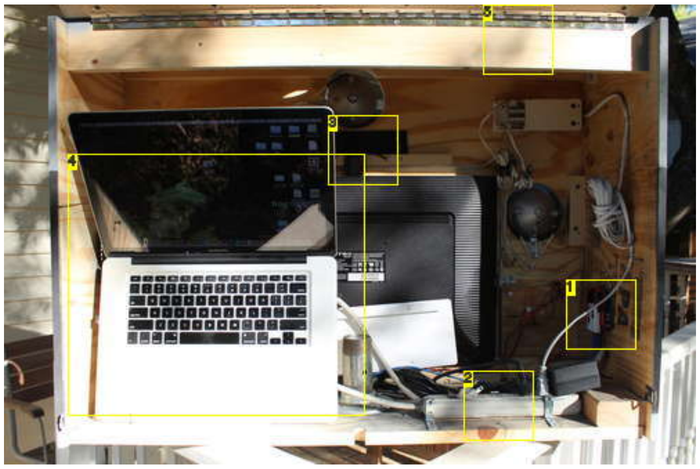


Image Notes
1. The arduino Micro Controller is mounted to the side panel here
2. I needed to add a power strip
3. The Logitech Camera is mounted just above the LCD monitor
4. The MacBook Pro is positioned here. After some initial Testing, i found that it overheats when it was mounted with the lid closed. Keeping the lid open prevented the computer from overheating and shutting down.
5. In this picture, you can also see the piano hinge which secures the door to the camera box.

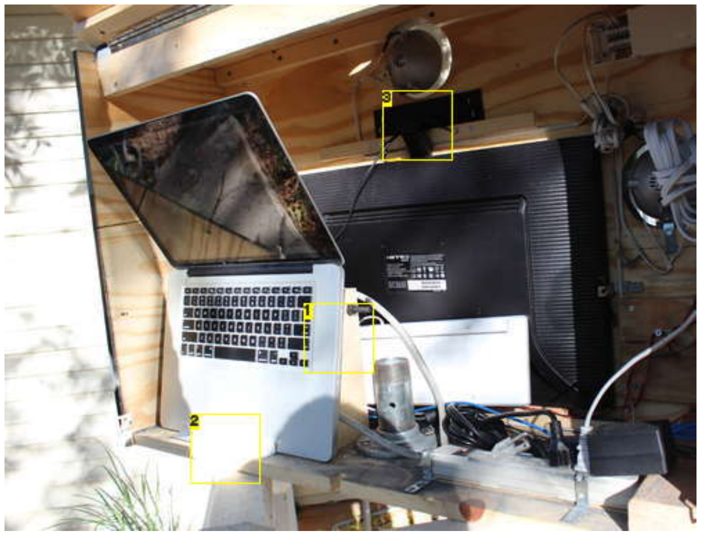


Image Notes
1. Here you can see the supports that hold the computer. A simple bungie cord is used to secure the computer to the supports
2. A piece of angle iron stops the computer from slipping forward
3. The webcam

Step 8: Testing

Once all of the components were added, I decided to stress test the photo booth. I wanted to see how long the device would continually run without any intervention, in conditions that it would likely encounter (sitting outside in the sun with +90 degree temperatures).

Note to you - TEST OFTEN AND EARLY!!

Here is a video of my early test - this is what i learned

http://anotherfrog.tumblr.com/post/12445371555/photobooth

Heat is an issue - After watching the computer overheat with about 15 minutes of use, I realized that heat was a big issue. I installed some old PC fans to help get rid of the heat.

Mac computers heat up with the Lid closed - Even with installing the fans, the compute was getting too hot. My original mouting system secured the computer in a vibration proof rig - with the lid closed. It was only after opening the computer so that the processor fans could run unobstructed that the device no longer overheated. This discovery led to 1 more fan (in the base of the camera box) and a mount which held the computer open.

The button was in the wrong place - with the button on the edge of the camera box, pressing it too hard caused the box to rock back and forth. Potentially an issue with inebriated guests, I moved the button to the center of the camera box, so that the force is directly perpendicular to the tripod.

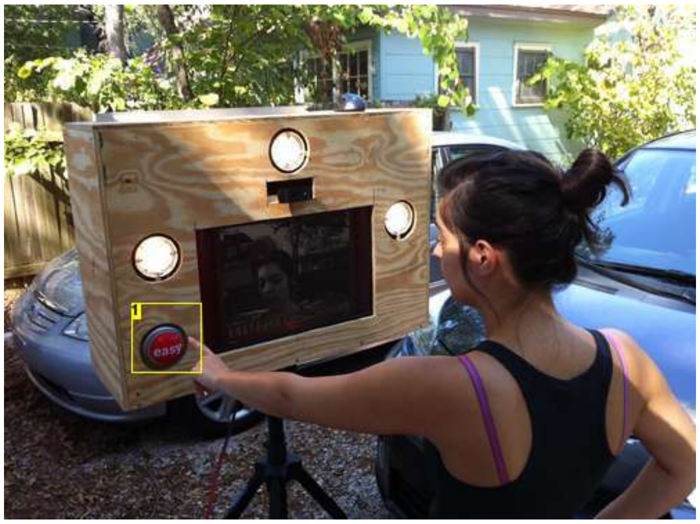


Image Notes
1. With the button mounted on the side of the box, pressing too hard would cause the camera box to rotate back and forth

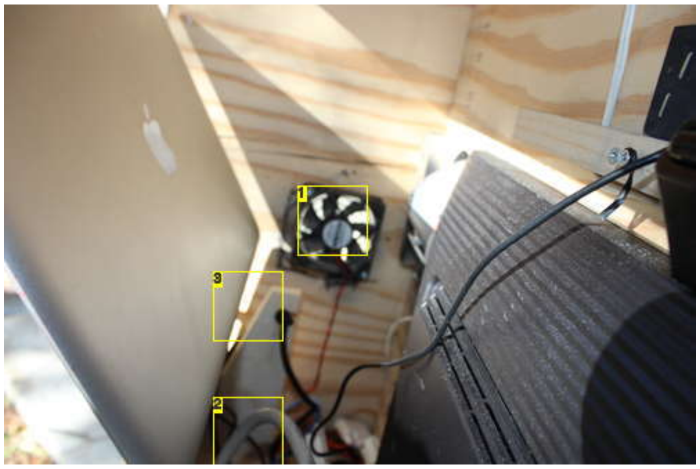


Image Notes
1. The first PC fan directs air directly across the back of the computer
2. The second PC fan directs air up from the bottom of the computer
3. This mount holds the computer open during operation - allowing the internal cooling fan to operate without obstruction

Step 9: Details and Finishing - Part 1

In this step, the components are removed, the box is primed and sanded. A coat of spray adhesive is applied to the surface to give the camera box a rough texture. The box is then primed and painted with Duplicolor paints - available at most auto supply stores.

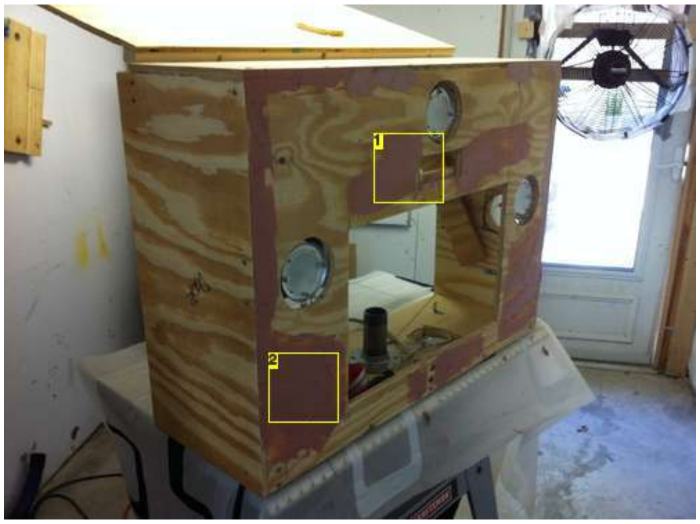


Image Notes
1. A little bondo and a lot of sanding to smooth out all the imperfections of plywood
2. The old button hole has been refilled and sanded smooth

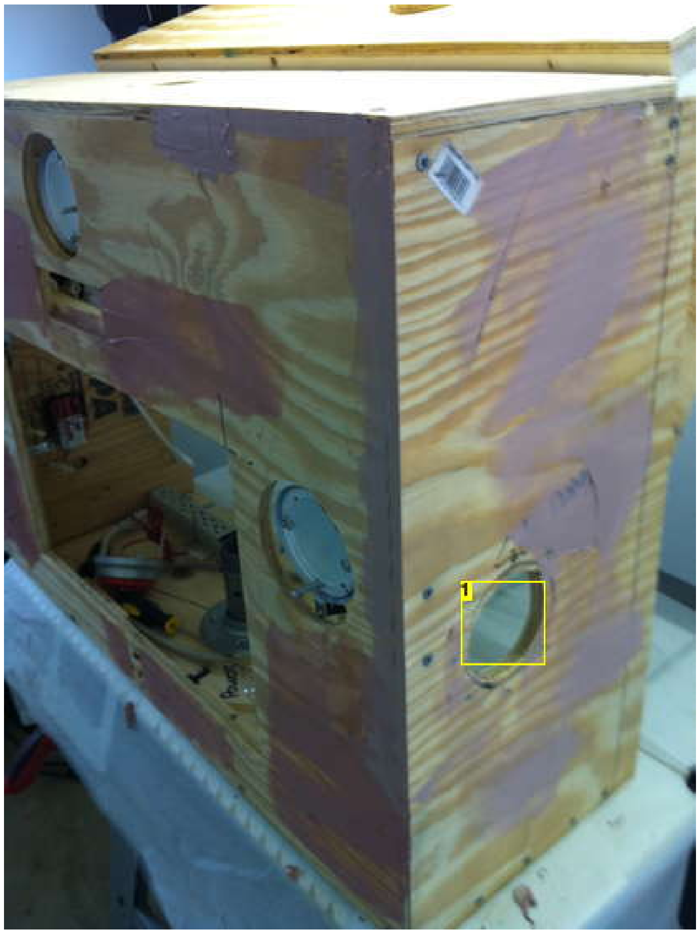


Image Notes
1. This is the hole for the side mounted PC fan. It will need some type of cover. I found one at Frys Electronics for $2.

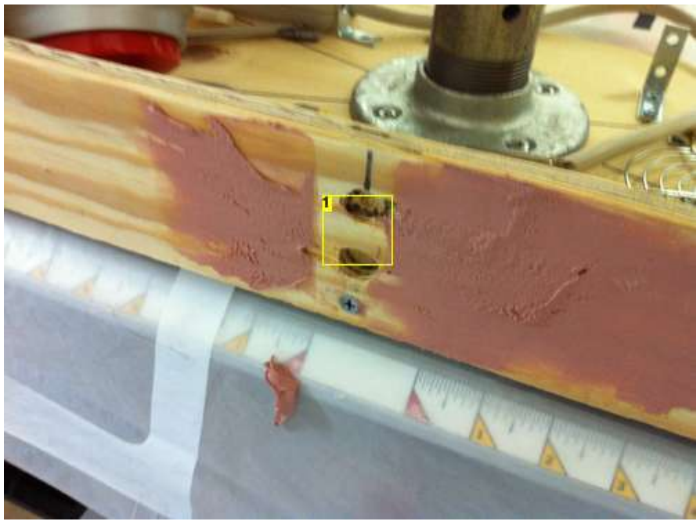
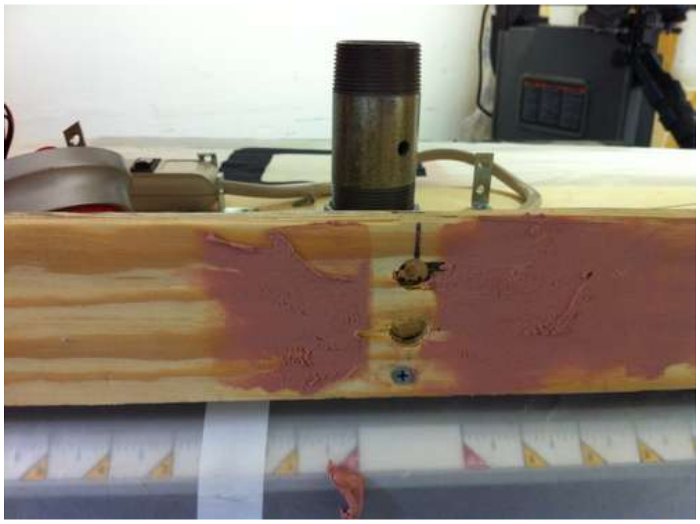


Image Notes
1. New attachment points for the button



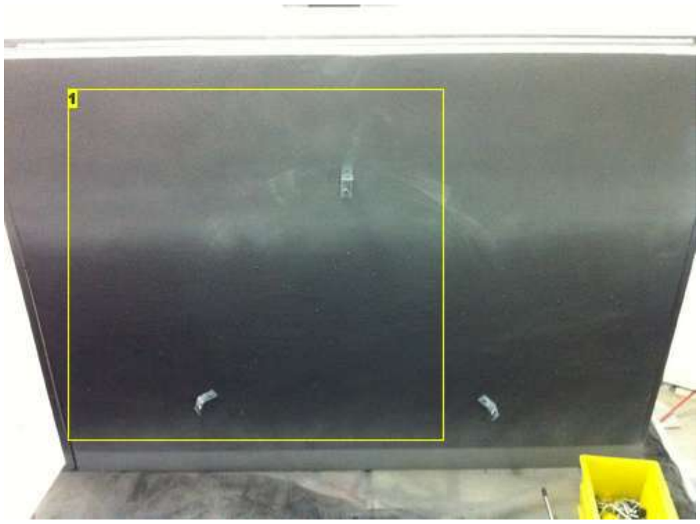


Image Notes
1. I used a trick to texturize the paint - and make it look like plastic. If you put down a layer of spray glue, holding the can far away from the surface, the glue will dry before it hits the panel - allowing for a nice texture to paint over, further obscuring any imperfections in the wood

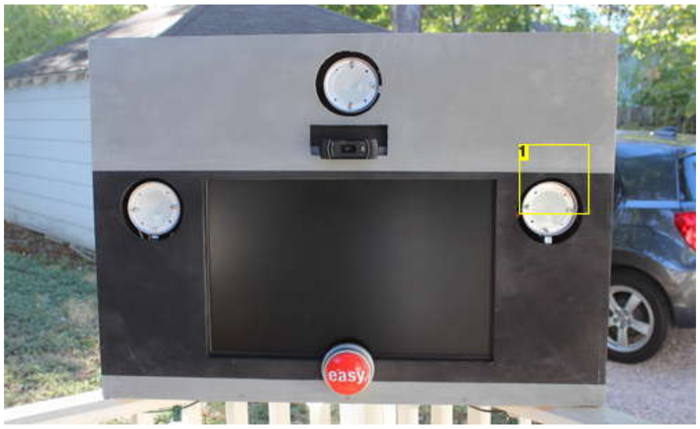


Image Notes
1. The final paint job with the components re-installed

Step 10: Details and Finishing - Part 2

In this step, additional embellishments are added to make the camera box appear more like a camera.

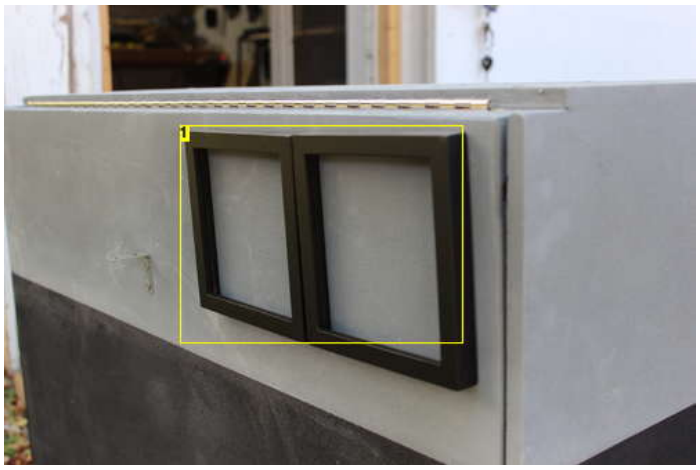


Image Notes
1. Some old photo frames were used to give the appearance of a view finder

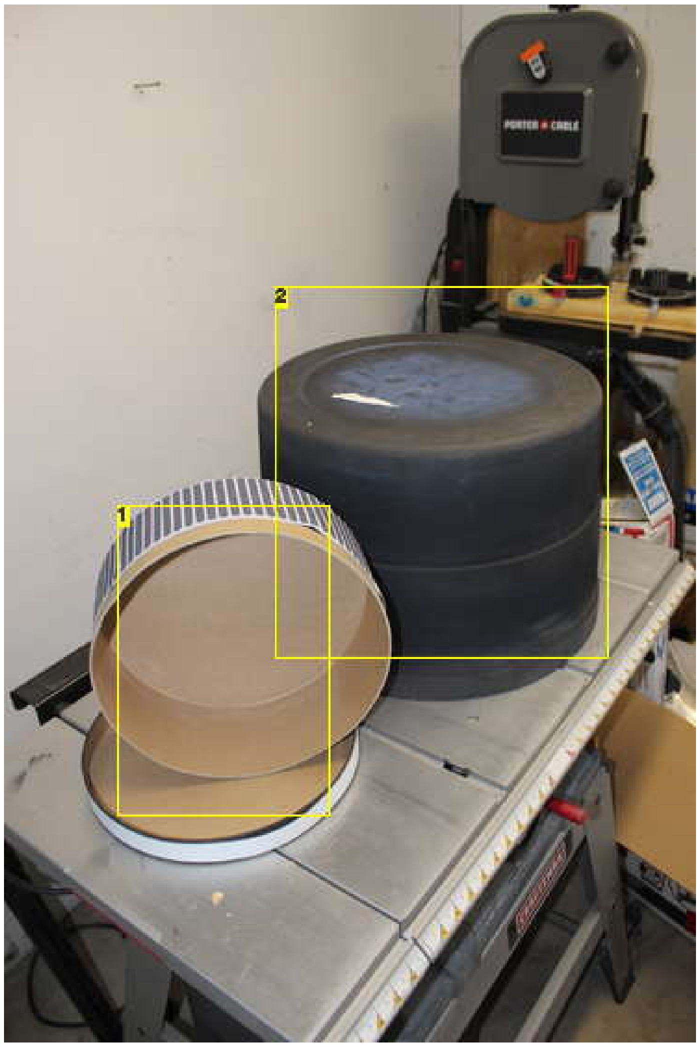


Image Notes
1. A hat box from the craft store is used to make the settings dial
2. This plastic drum from big lots is used to make the lense - a $5 part

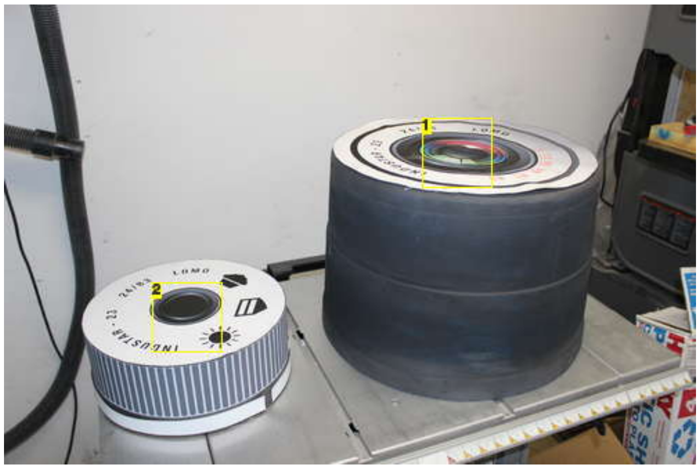


Image Notes
1. Lens with decal applied
2. Settings dial with decals applied

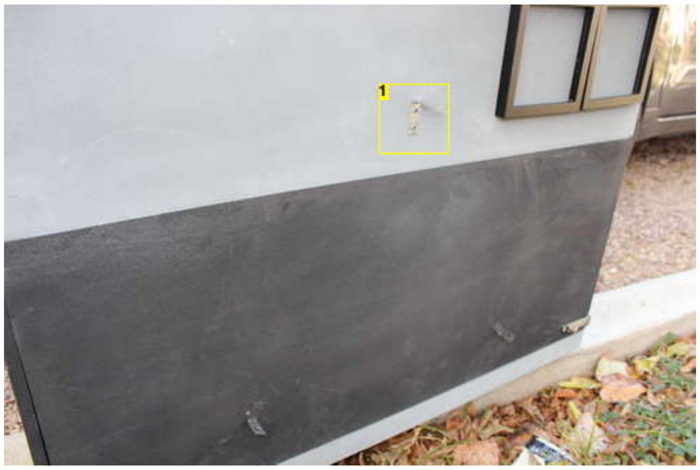


Image Notes
1. The drum is mounted onto the front of the camera with l brackets and small screws. This method allows the lens to be removed for travel

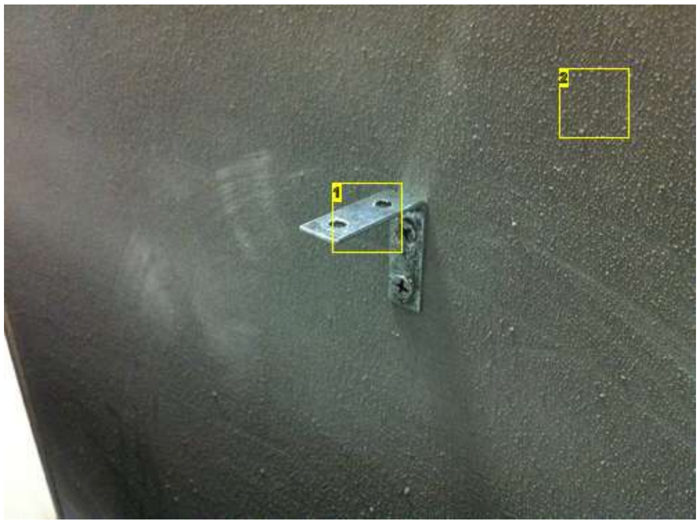


Image Notes
1. Close up of the l bracket
2. Here you can see the texture of the paint - achieved with a light dusting of spray adhesive before the paint and primer are applied

Image Notes
1. A small screw in the bucket fits perfectly into the hole in the L bracket

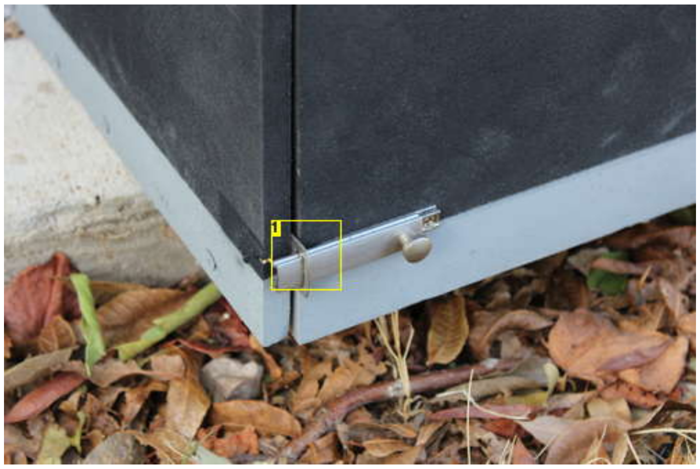
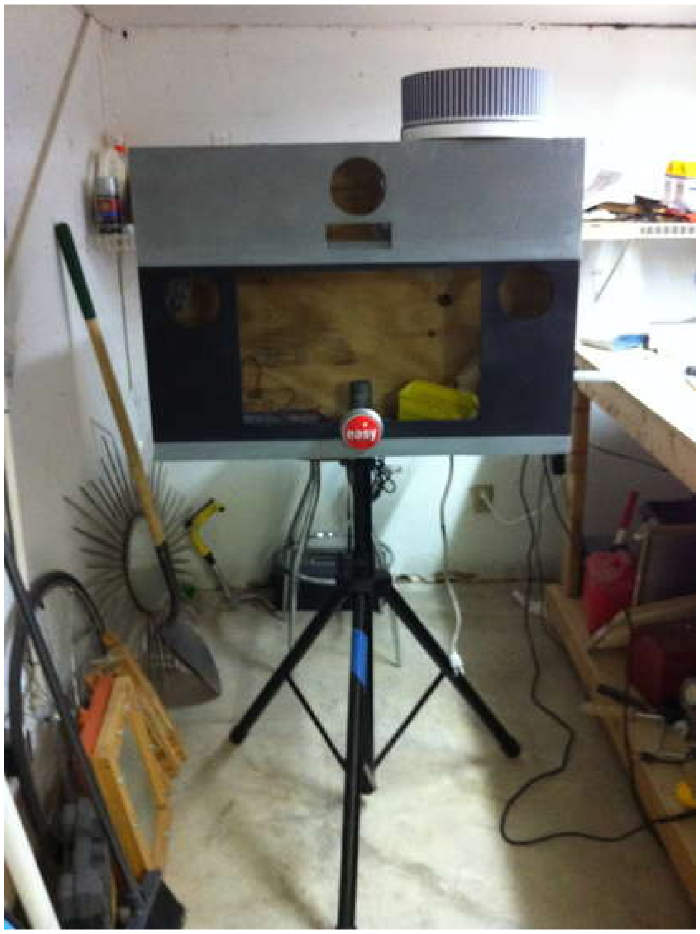


Image Notes
1. Locks are added to the camera door

Image Notes
1. a cover is added to the PC fan hole

Image Notes
1. A few more coats of black primer, and the lens will be finished



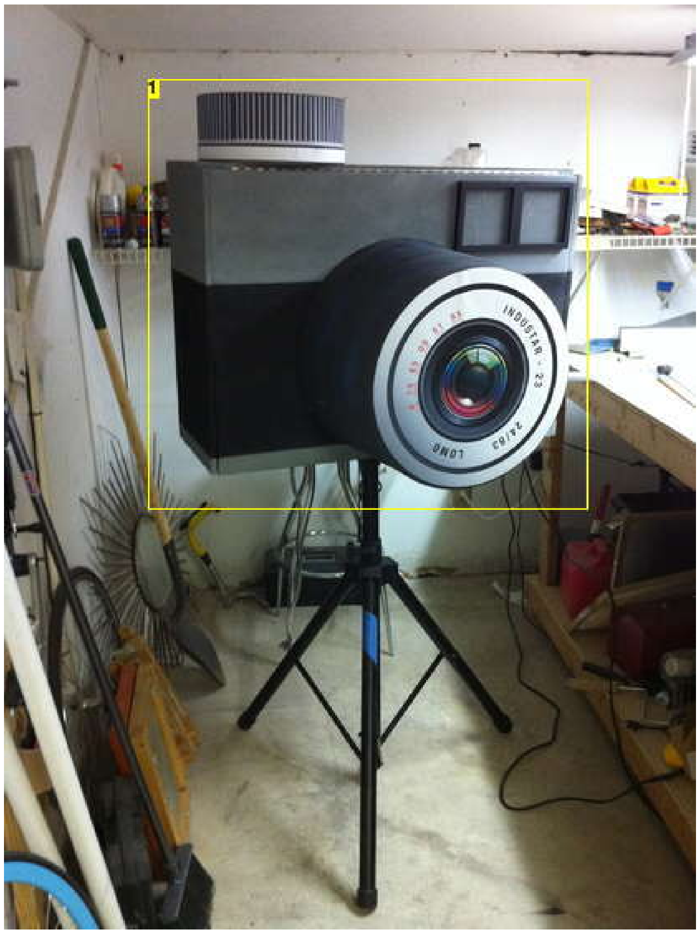


Image Notes
1. Everything is finished!

File Downloads

lens vector2.ai (915 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'lens vector2.ai']

Step 11: Usage

Here are some of my favorite photos from the wedding (The guests took over 800 photos in a 4 hour period). My wife and I also rented a bunch of props from a local theater company, in addition to making a few of our own.

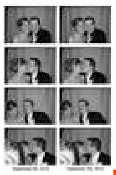
I hope you enjoyied this post. Please let me know if you make this yourself and especially if you make any big changes. I'm always excited to see how people make things their own.

Matt Franks

Mfranks at famunited dot com

Image Notes
1. Me and the little lady!

Related Instructables


DIY Portable Wedding Photo Booth by dohjoe

Automatic Photo Booth Using Arduino Board (Photos) by jordanpacker

How to build your own Photo Booth by MoonRaker

DIY Wedding Photobooth with Easy Disassembly (pics only)

Photo Booth with a Live Slideshow by lightingCat

Make Your Own Tabletop Photobooth by jumpfroggy

Photo Booth by leuff

DIY Photobooth by jchorng

(Photos) by
nonoodlez