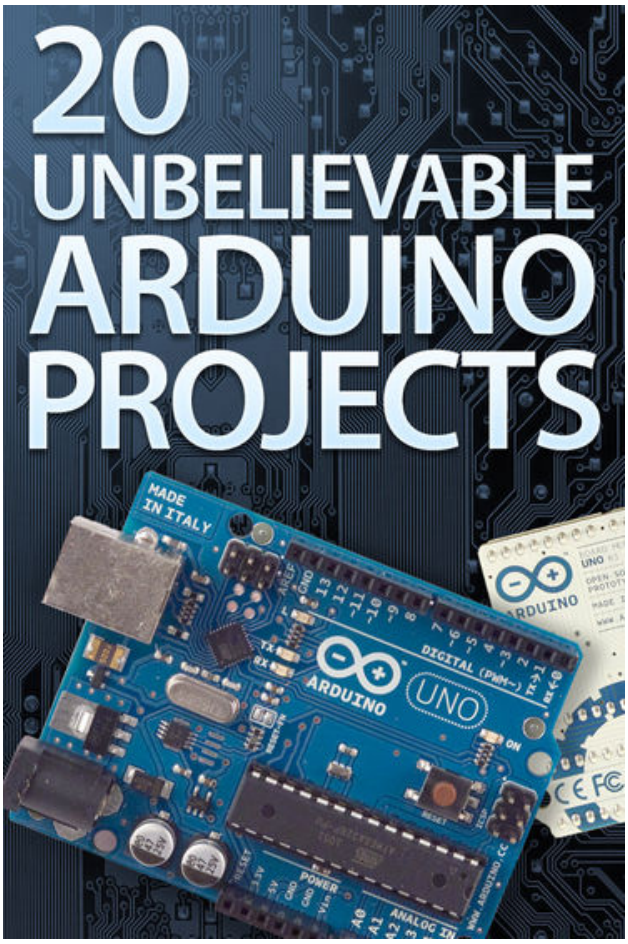


Introduction

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

Instructable: LED Cube 8x8x8
Author: chr
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Power Laces- the Auto lacing shoe
Author: blakebevin
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Plantduino Greenhouse
Author: clovercreature
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: The EyeWriter 2.0
Author: thesystemis
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Twitter Mood Light - The World's Mood in a Box
Author: RandomMatrix
License: None (All Rights Reserved) (c)

Instructable: Flamethrowing Jack-O'-Lantern
Author: randofo
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Make a 24X6 LED matrix
Author: Syst3mX
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Secret Knock Detecting Door Lock
Author: Grathio
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Turn signal biking jacket
Author: leahbuechley
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Tree Climbing Robot
Author: Technochicken
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Rave Rover - Mobile Dance Stage
Author: cwilliamson8
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Type Case, the making of a low-resolution display
Author: Martin Bircher
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Sigh Collector
Author: mkontopo
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Make a Fire Breathing Animetronic Pony from FurReal Butterscotch or S'Mores
Author: lvl_joe
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Tweet-a-watt - How to make a twittering power meter...
Author: adafruit
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Bubblesteen Bubble Machine
Author: belliedroot
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: Arduino R/C Lawnmower (painted)
Author: johndavid400
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: How to Build an Arduino Powered Chess Playing Robot
Author: mJusticz
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: SITWAY
Author: mickydee
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Instructable: A Makers Wedding - Photo booth
Author: letMeBeFranks
License: Attribution-NonCommercial-ShareAlike (by-nc-sa)

Disclaimer

All do-it-yourself activities involve risk, and your safety is your own responsibility, including proper use of equipment and safety gear, and determining whether you have adequate skill and experience. Some of the resources used for these 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. The projects are not intended for use by children.

Many projects on Instructables are user-submitted, and appearance of a project in this format does not indicate it has been checked for safety or functionality. Use of the instructions and suggestions is at your own risk. Instructables, Inc. disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with all applicable laws.

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!

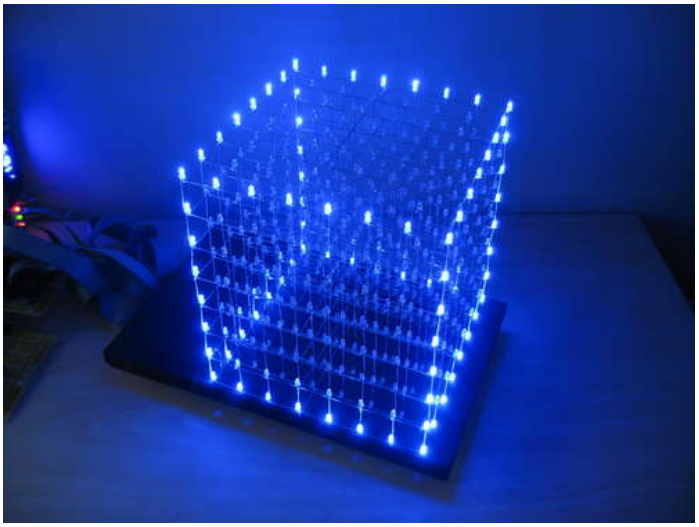
We believe this Instructable is the most comprehensive step-by-step guide to build an 8x8x8 LED Cube ever published on the intertubes. It will teach you everything from theory of operation, how to build the cube, to the inner workings of the software. We will take you through the software step by step, 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.

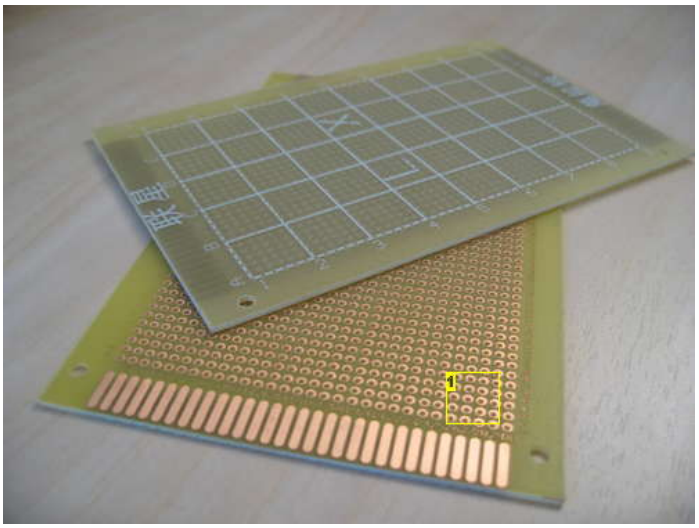


Image Notes
1. Make sure to get this type of prototyping PCB.

Image Notes
1. Lots and lots of LEDs!

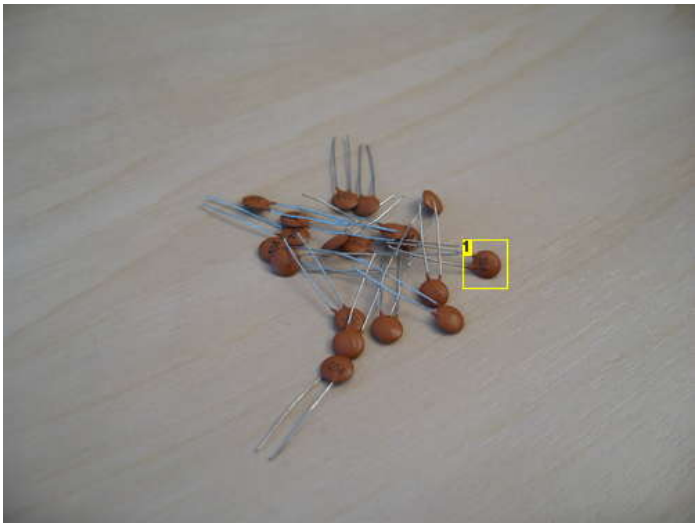


Image Notes
1. 100nF

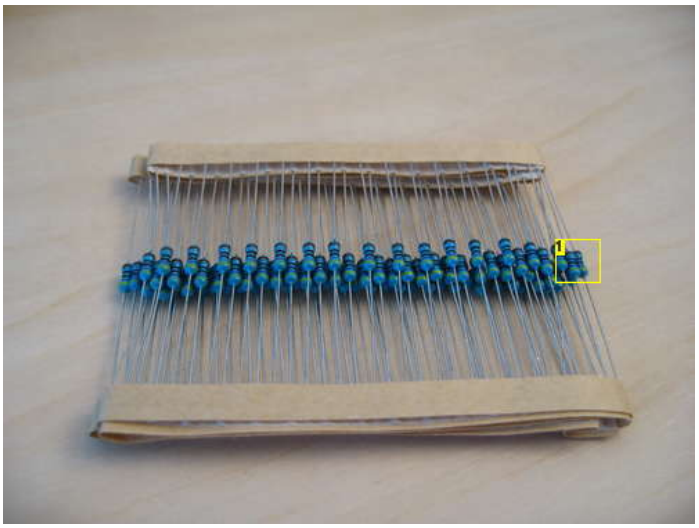


Image Notes
1. Don't look at the color codes. This is not 100ohms.

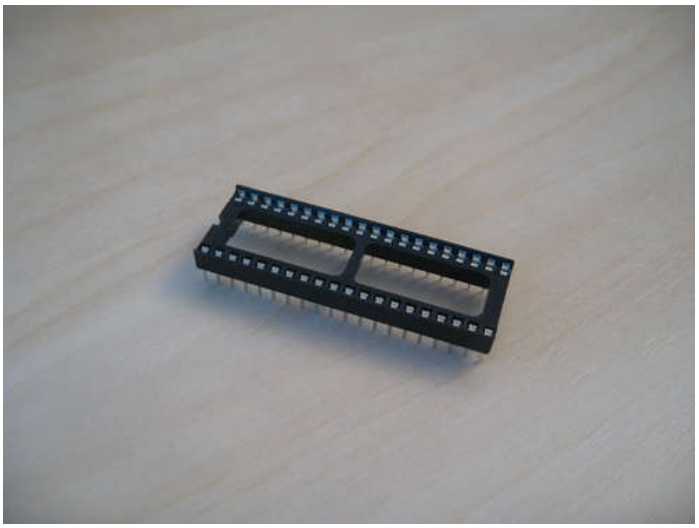


Image Notes
1. Kynar wrapping wire. 30 AWG.

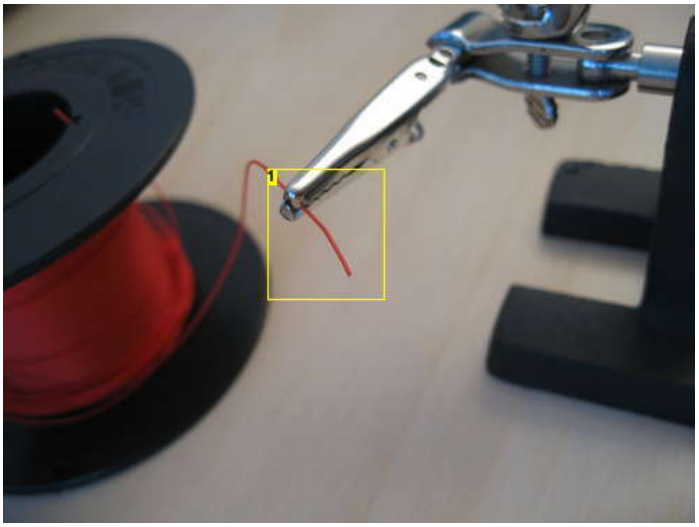


Image Notes
1. Very tiny wire. Perfect for working on prototyping PCBs.

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

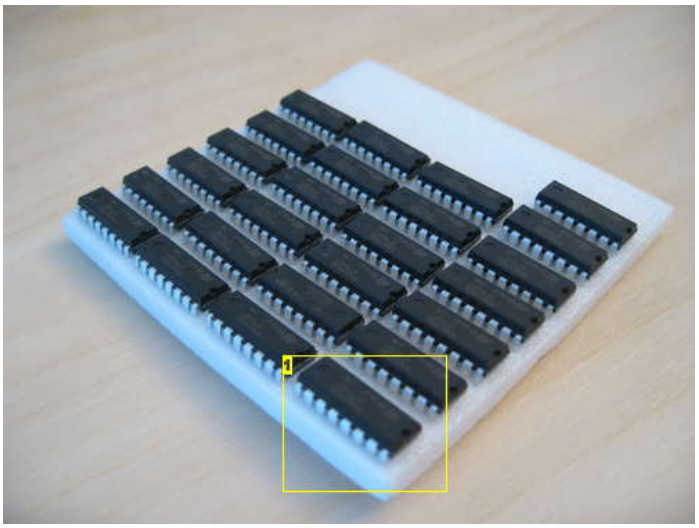


Image Notes
1. Lots of ICs

	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.

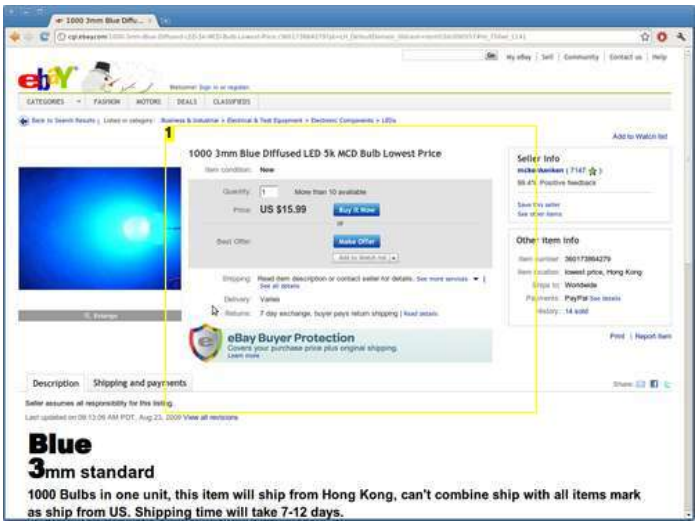
Most of the time, it is more important to actually have the component value at hand when you need it.

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.



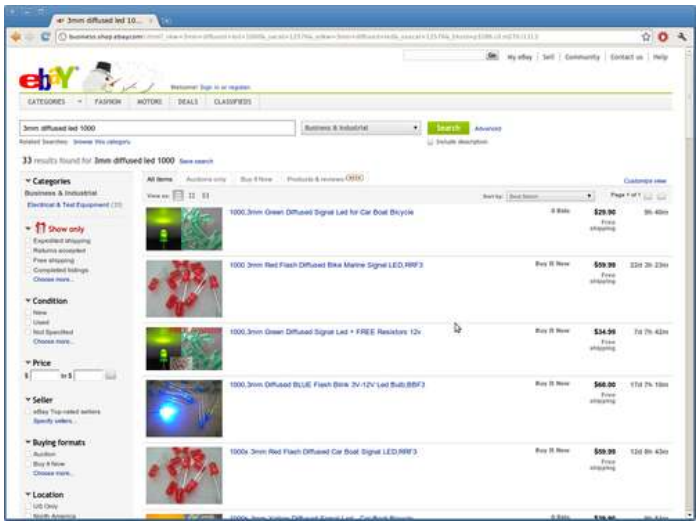
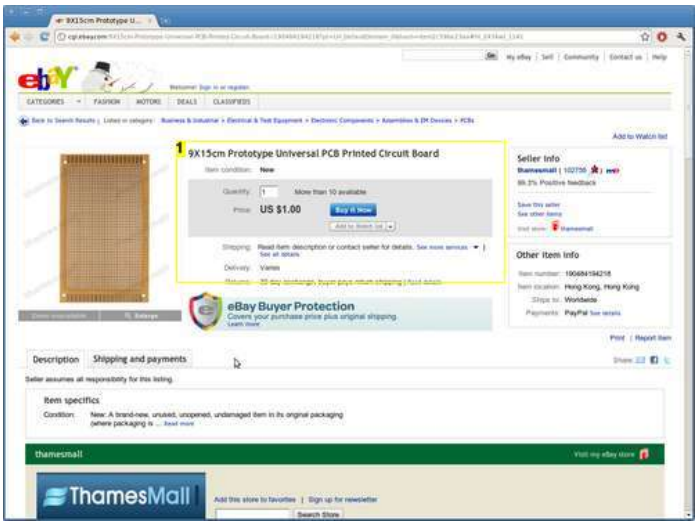


Image Notes
1. 1000 leds for 16 bucks. But beware! The descriptions aren't always that great. We ordered diffused leds and got clear ones :/



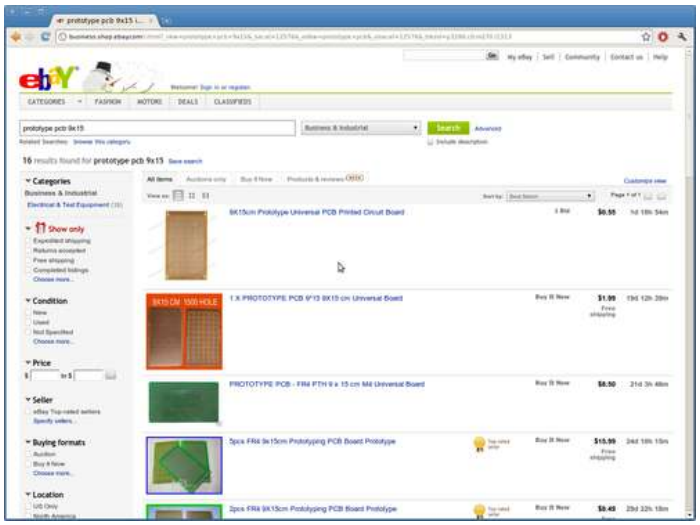
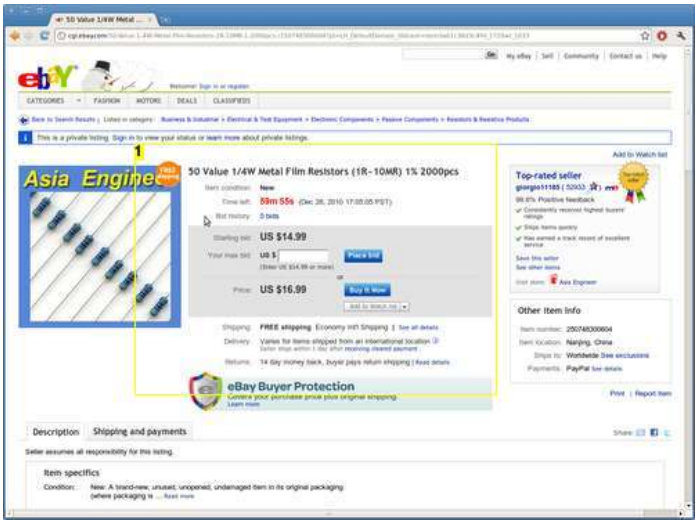


Image Notes
1. This is the type of prototype PCB we used. 1 dollar!



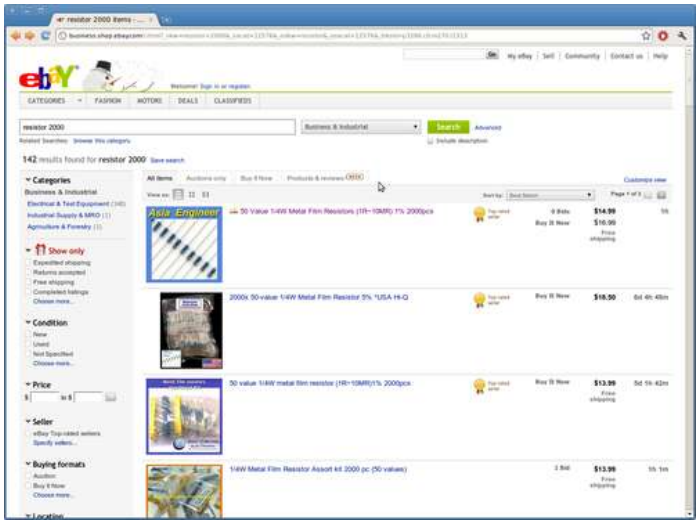


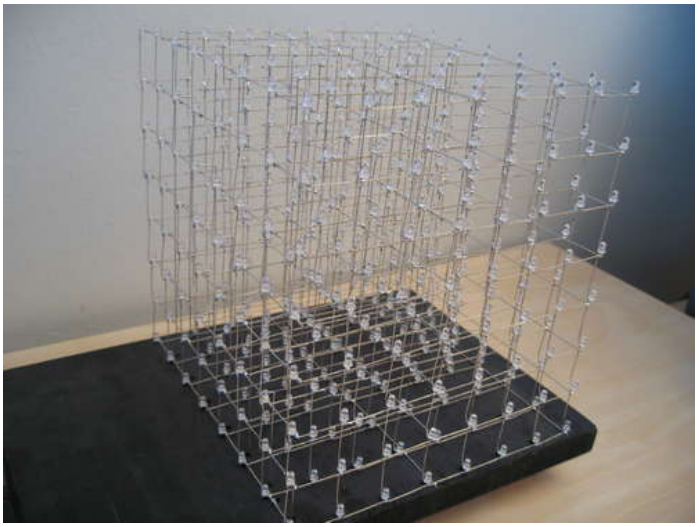
Image Notes
1. 2000 resistors for 17 USD

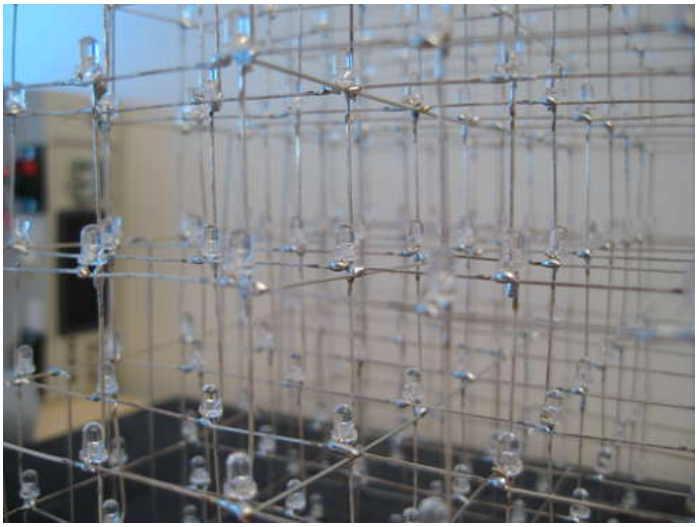
Step 4: What is a LED cube

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.

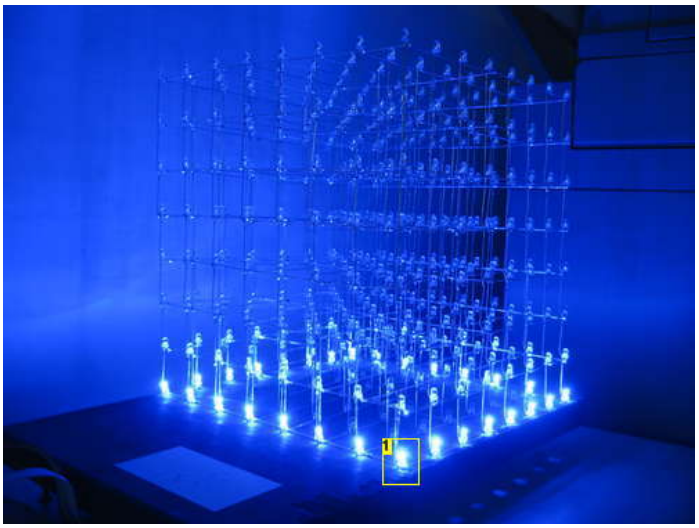


Image Notes
1. We start by flashing the bottom layer. Layer 0.

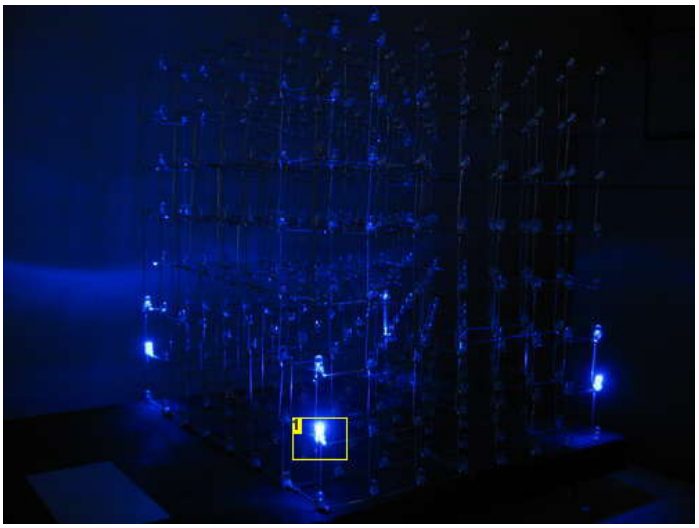


Image Notes
1. Then the second.

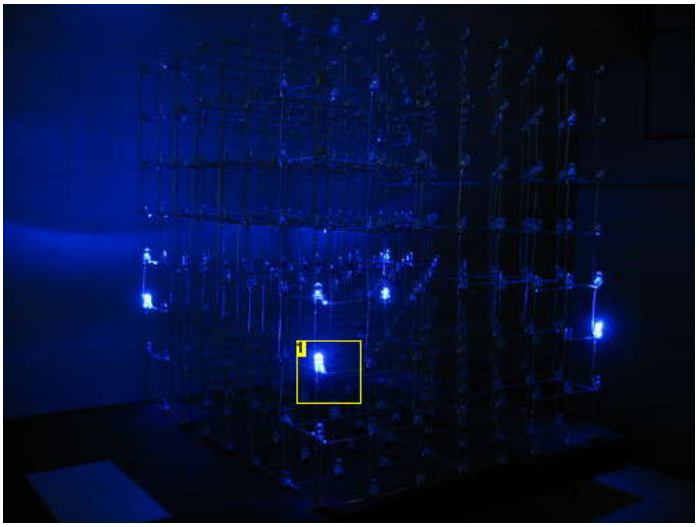


Image Notes
1. And so on..

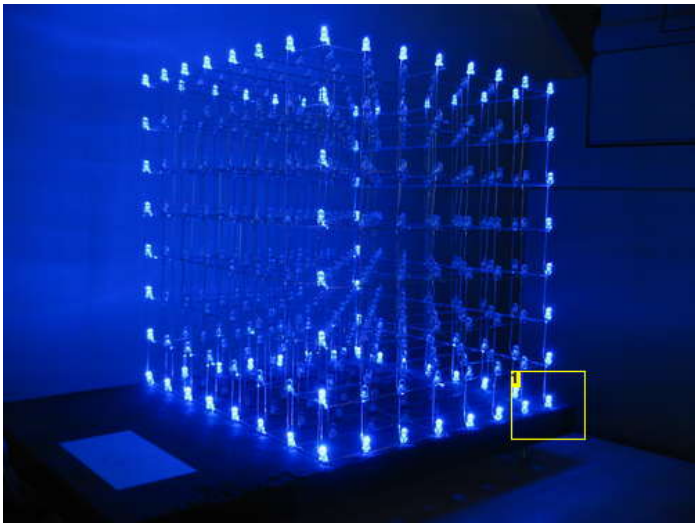


Image Notes
1. Do this fast enough, and your human eyes won't know the difference! Robots may be able to see past the illusion, though.

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.

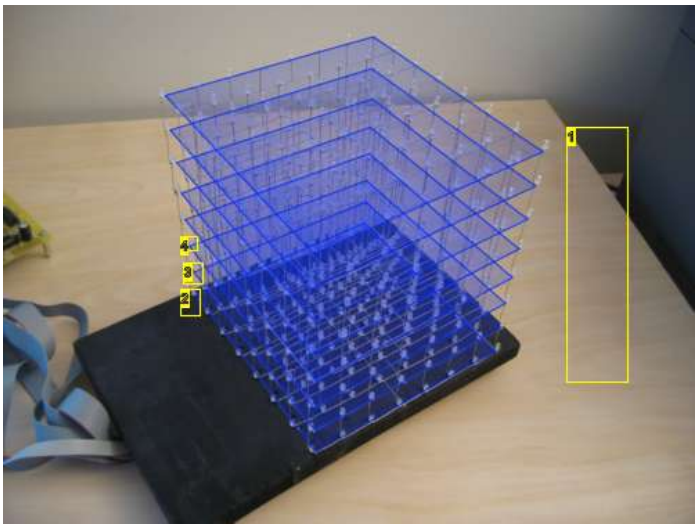


Image Notes
1. 8 layers
2. A 64x64 image is flashed first on layer 0
3. Then another image is flashed on layer 1
4. Wash rinse repeat

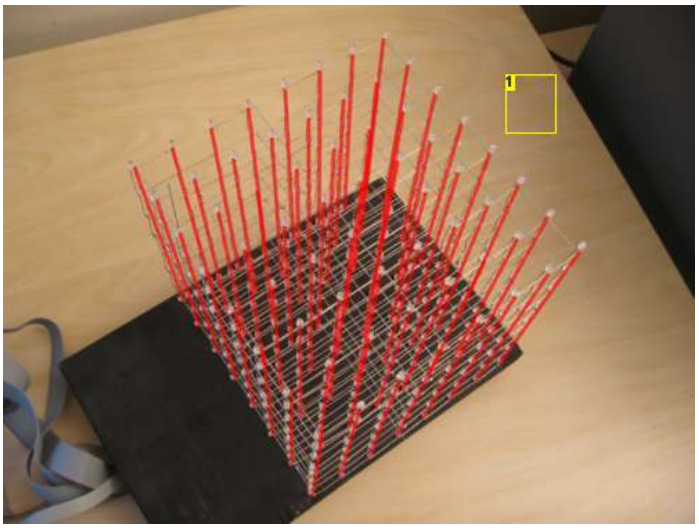


Image Notes
1. 64 columns

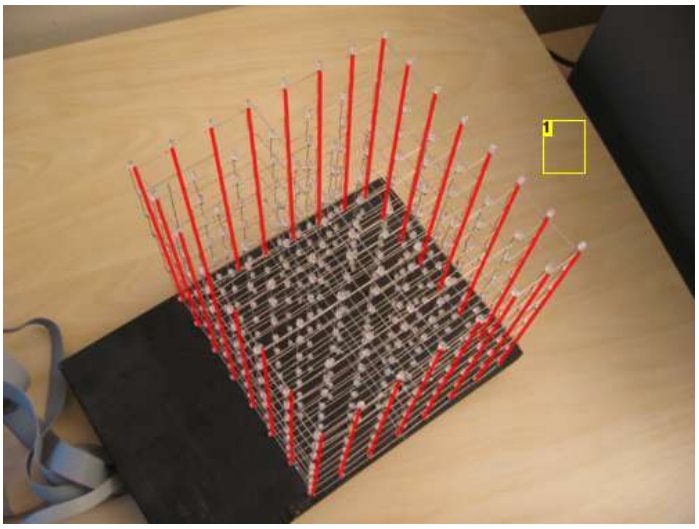


Image Notes
1. Was easier to see when I didn't draw all 64 lines

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.

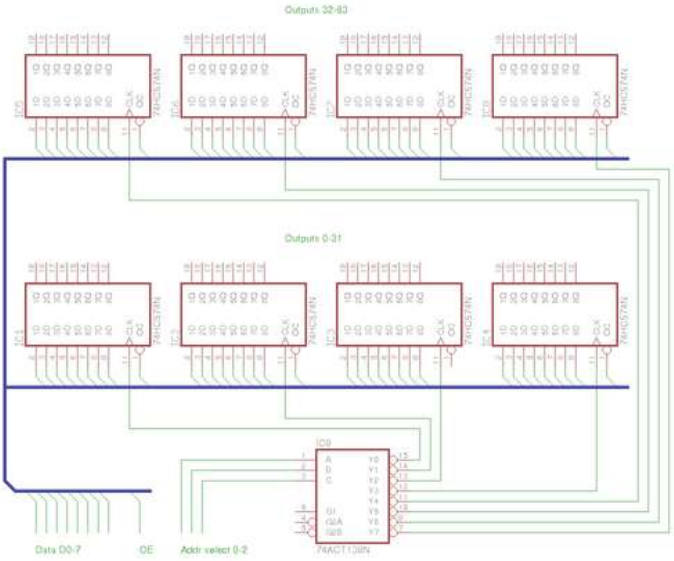
To load the on/off states of all the 64 LEDs we simply do this: Load the data of the first latch onto the bus. pull the CP pin of the first latch low then high. Load the data of the second latch onto the bus. pull the CP pin of the second latch low then high. Load the data of the third latch onto the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

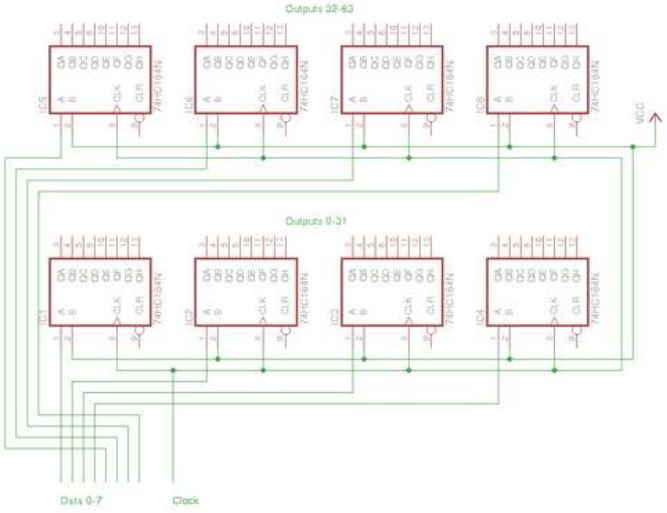
In the previous solution, each byte in our buffer array was placed in it's own latch IC. In this setup each byte will be distributed over all 8 shift registers, with one bit in each.

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.

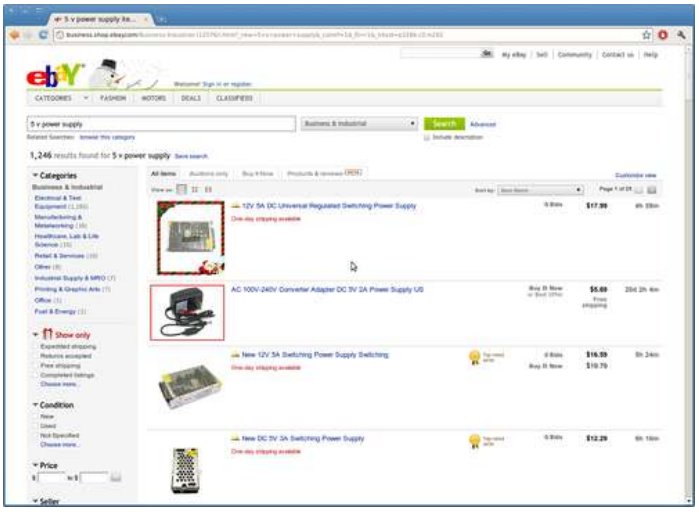
Image Notes
1. Cube drawing almost half an amp at 5 volts.

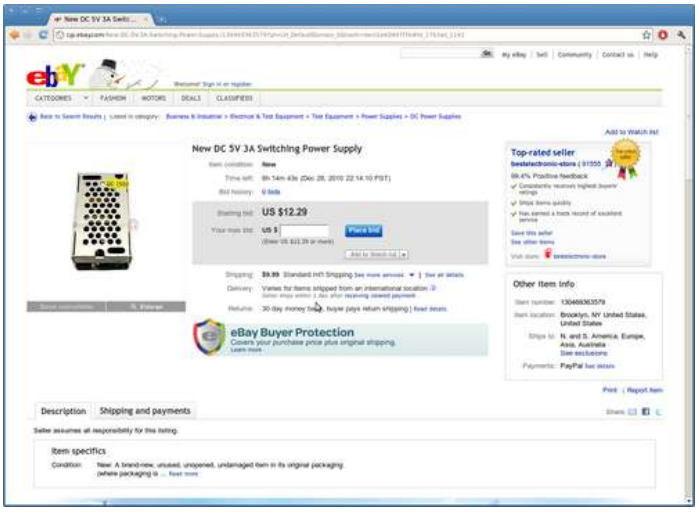
Step 11: Buy a power supply

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.





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.

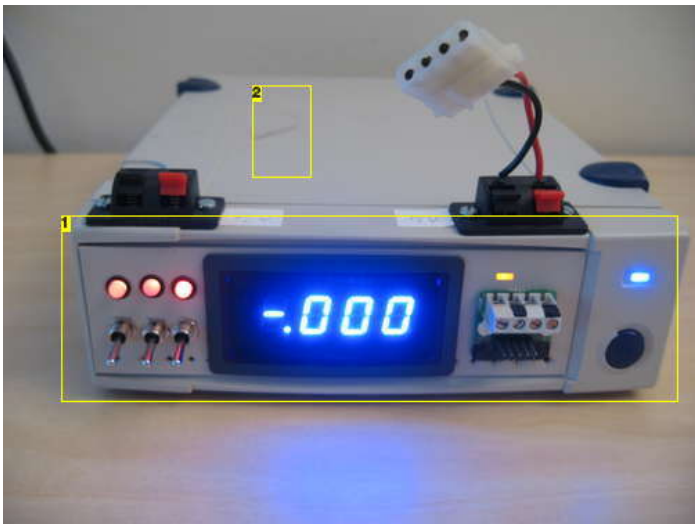


Image Notes
1. Old SCSI disk
2. Inside here is a small powersupply that used to supply the SCSI hard drive that was inside.

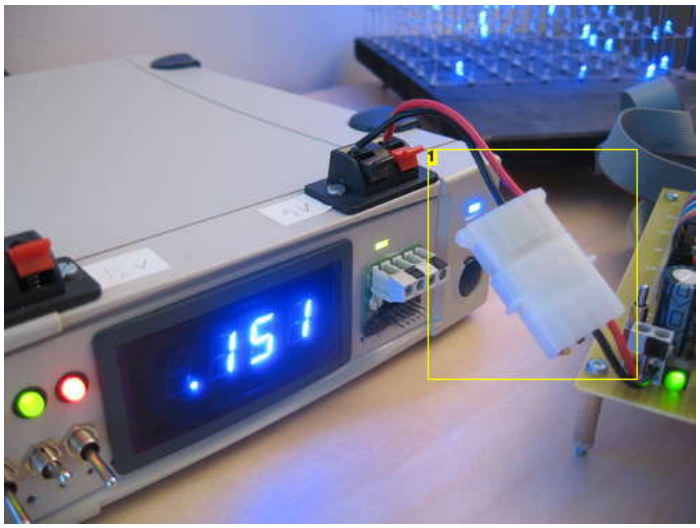
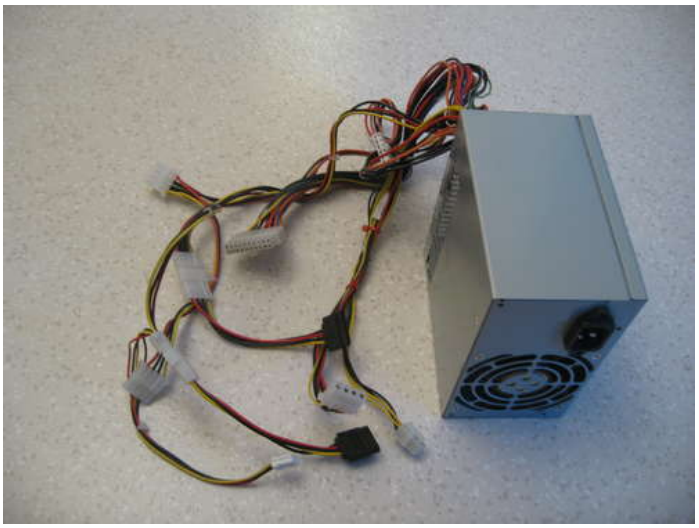


Image Notes
1. Used a Molex connector so we could disconnect the cube easily.



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..

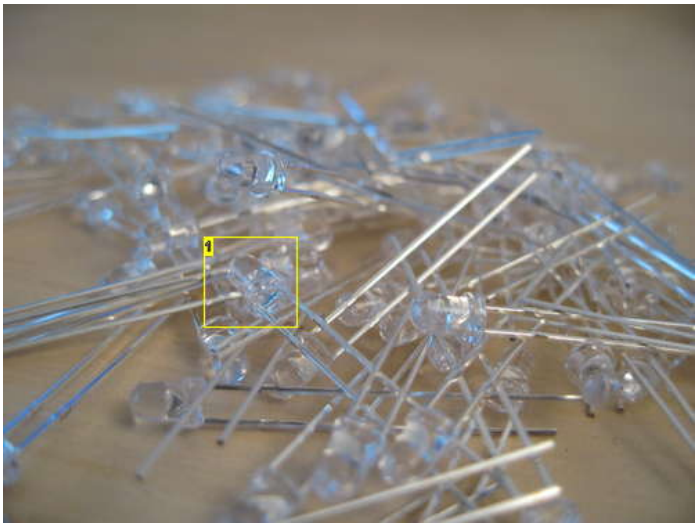


Image Notes
1. These are the ones we ended up using

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

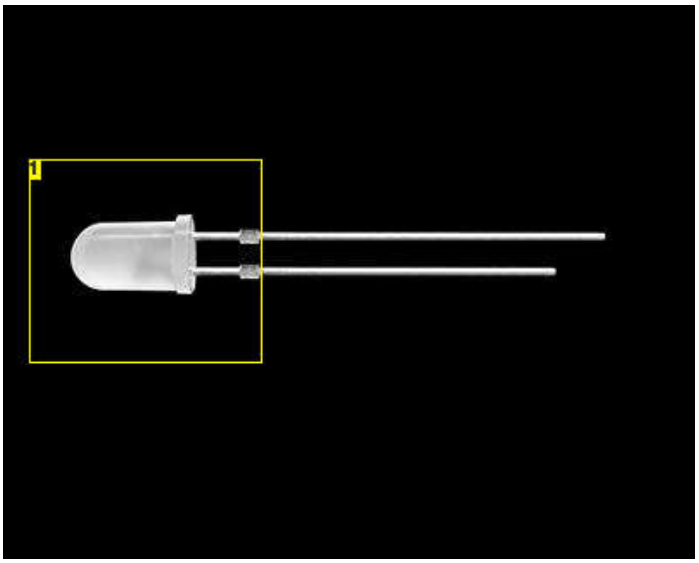


Image Notes
1. GOOD This is what we expected to receive. Diffused LED.

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/

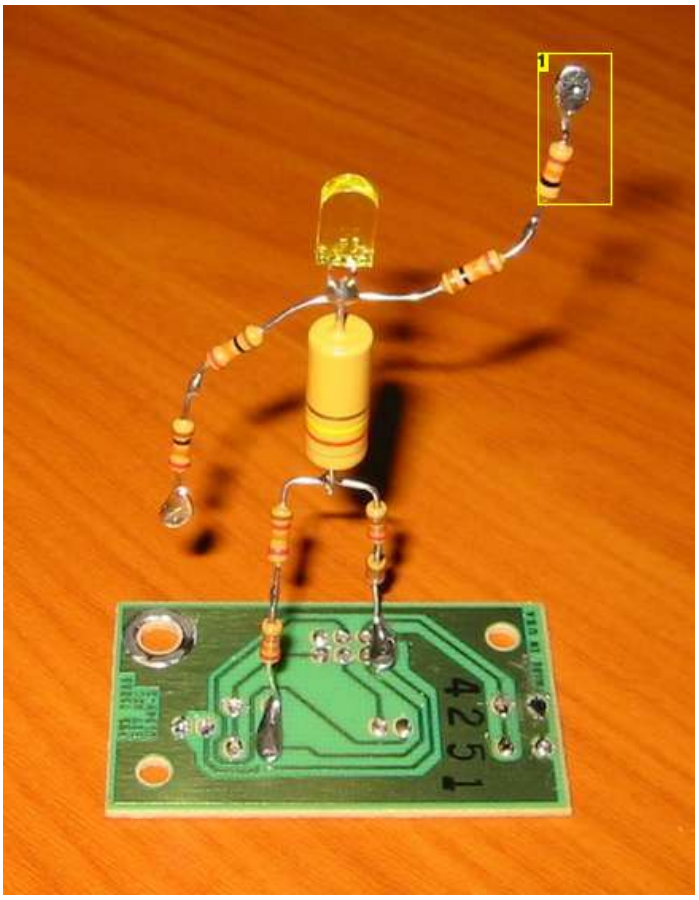


Image Notes
1. Viva la resistance!!

Step 15: Choose the size of your cube

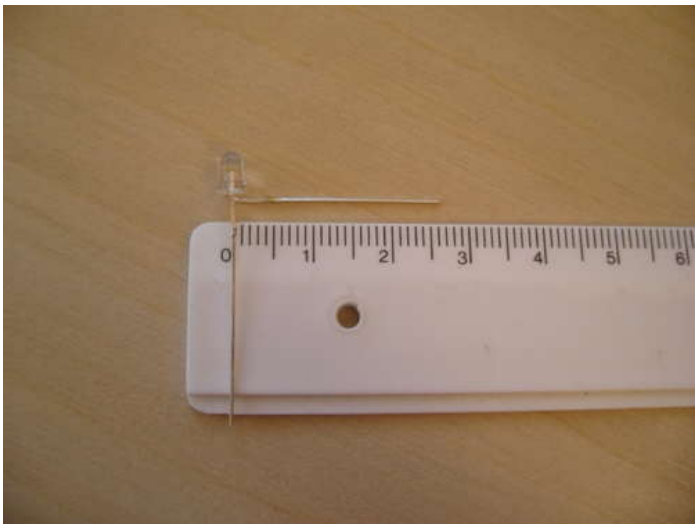
We wanted to make the LED cube using as few components as possible. We had seen some people using metal rods for their designs, but we didn't have any metal rods. Many of the metal rod designs also looked a little crooked.

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.

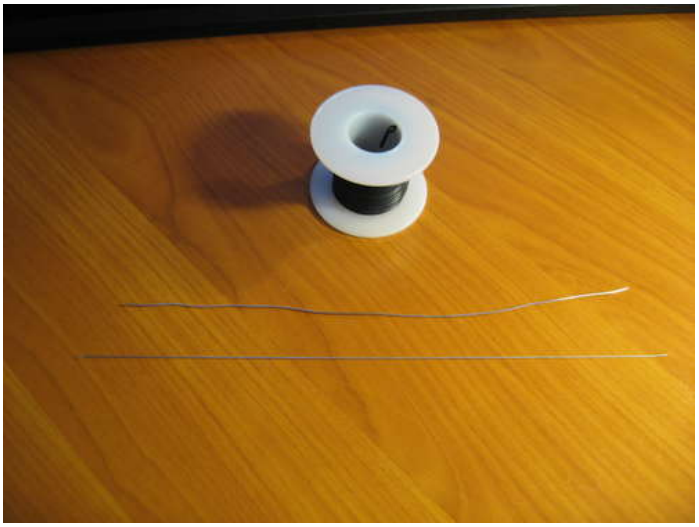
So we figured we should give pulling a try, and it worked! 100% straight metal wire from a spool!

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

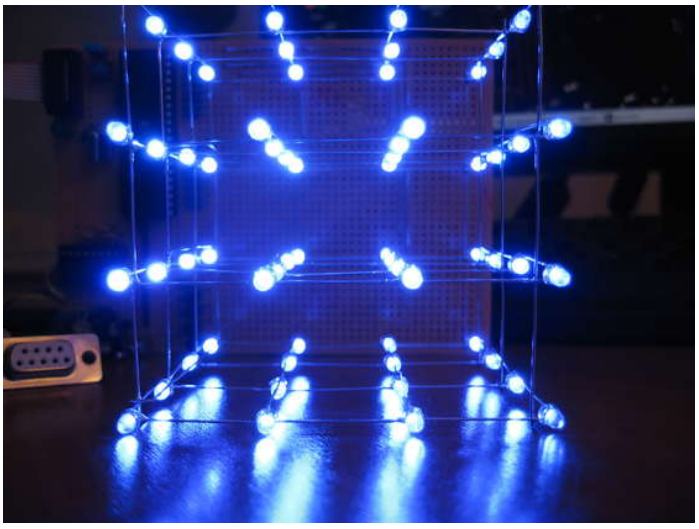
Whenever Myth Busters are testing a complex myth, they start by some small scale experiments.

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.

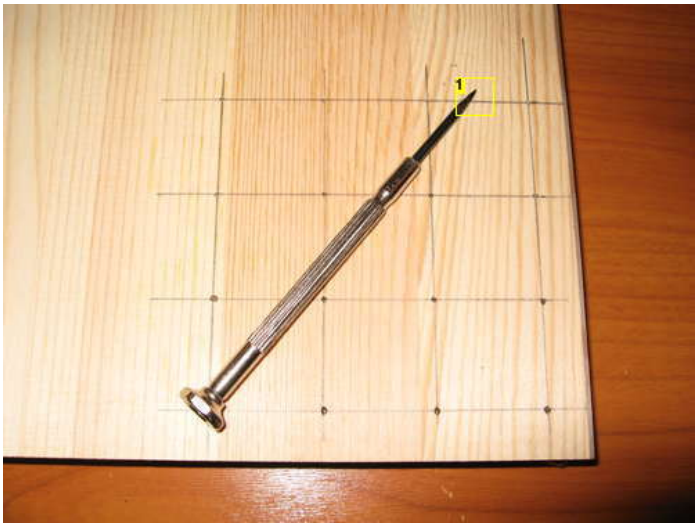


Image Notes
1. If you make a small indentation before drilling, the drill won't slide sideways.

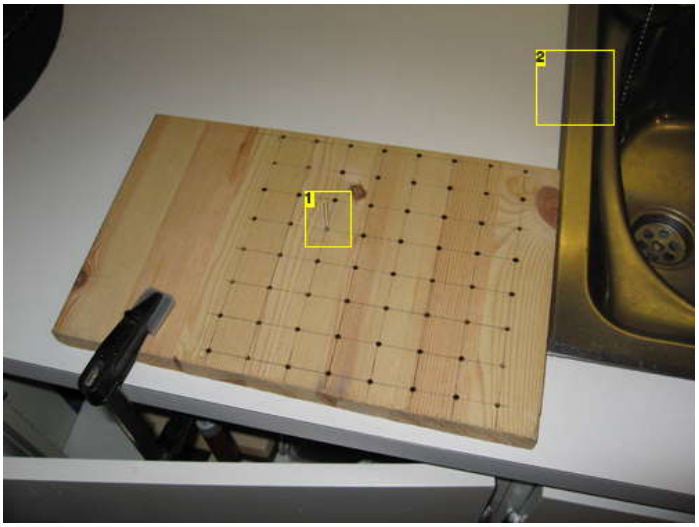


Image Notes
1. All done. We used this LED to test all the holes.
2. Everything but the kitchen sink? We sort of used the kitchen sink to hold the jig in place ;)

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 ;)

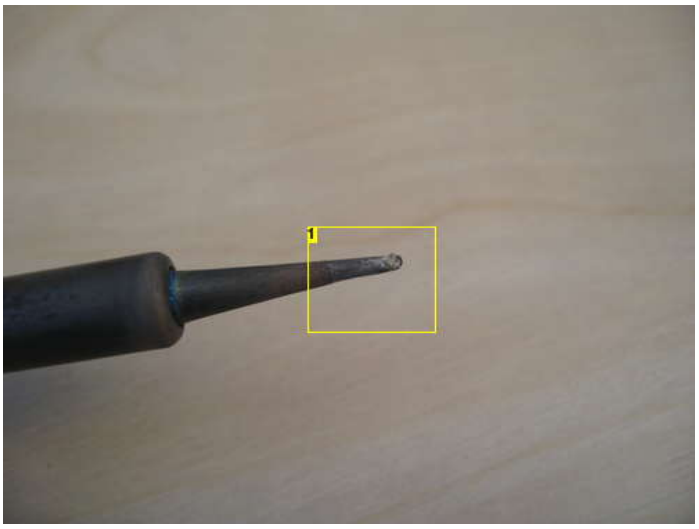


Image Notes
1. If the tip of your soldering iron looks like this, it is time to clean it!

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.

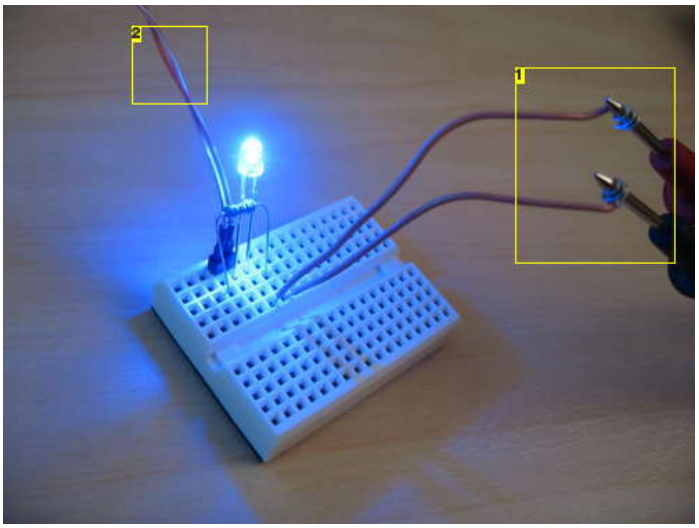


Image Notes
1. Multimeter connected in series to measure mA.
2. 5 volts from power supply

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.

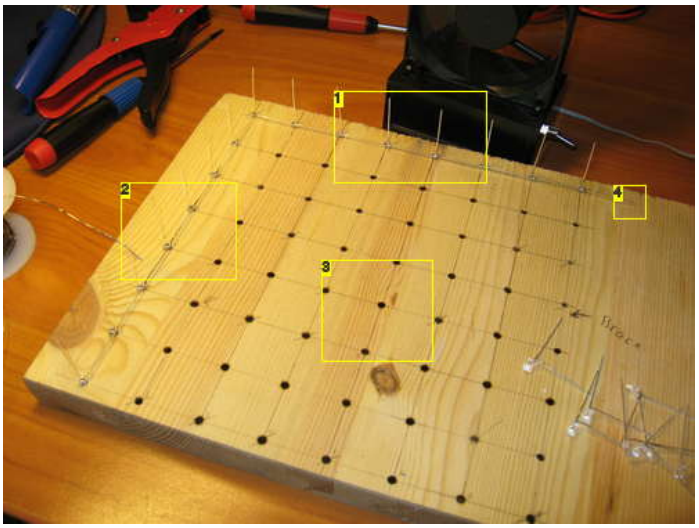


Image Notes
1. Start with this row
2. Then do this column
3. And then the rest..
4. Don't remove the leg that sticks out to the side. It is convenient to connect ground to it when testing the LEDs.

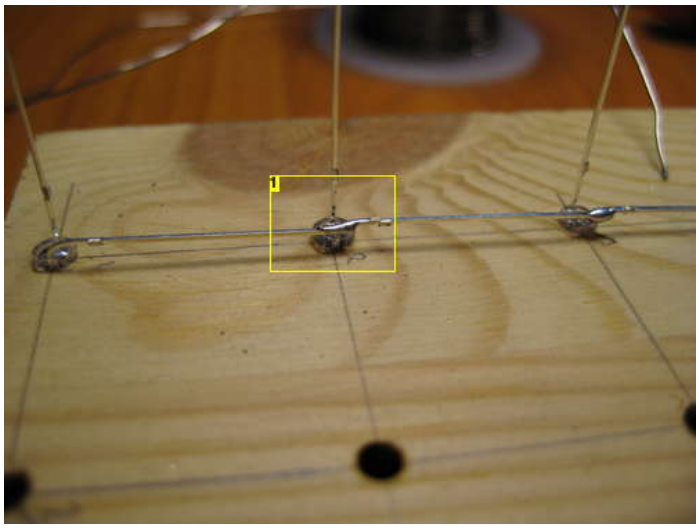


Image Notes
1. About 1mm overlap. Perfect!

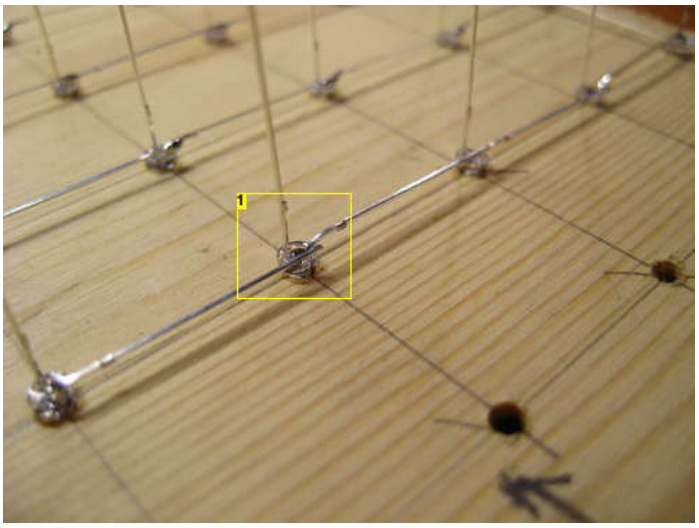


Image Notes
1. LED ready to be soldered. Look how nicely they line up.

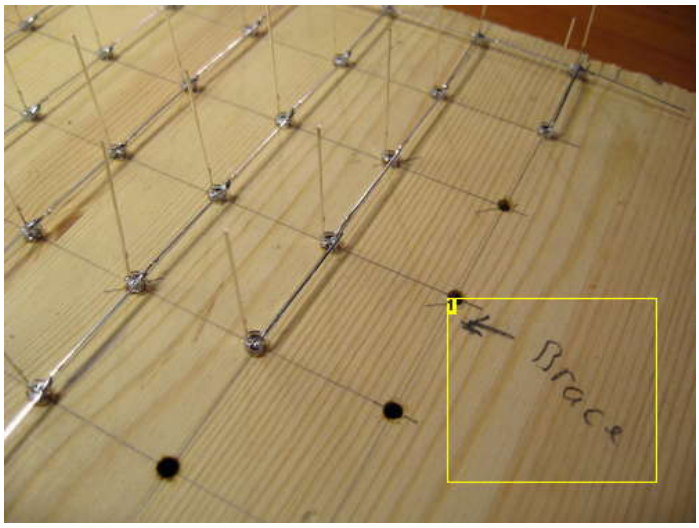
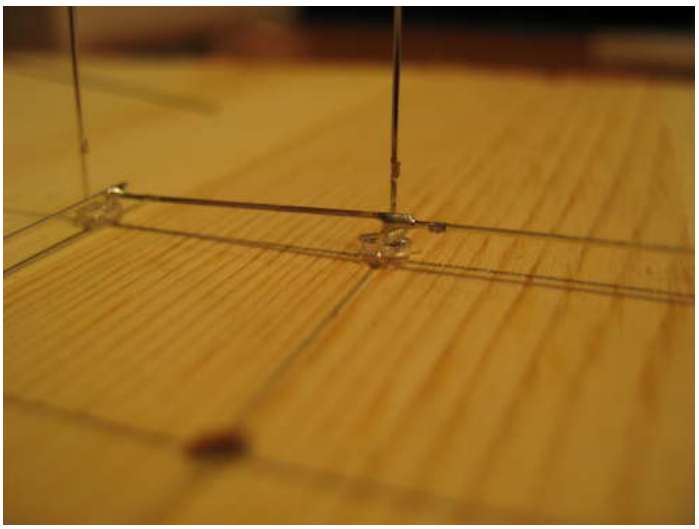
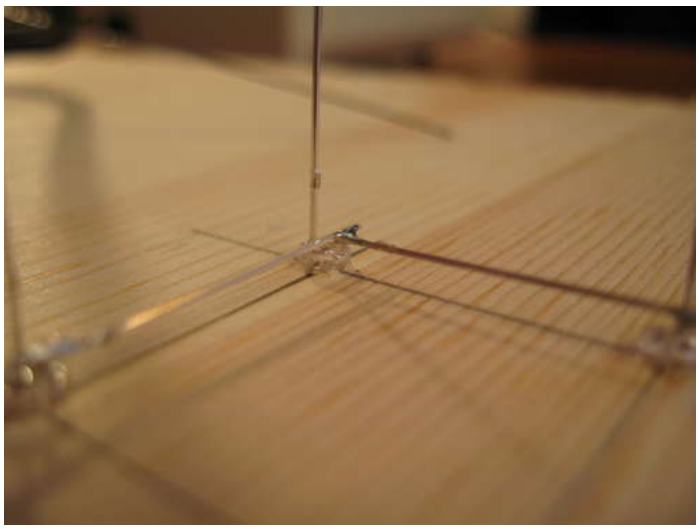


Image Notes
1. We marked off where we wanted to have the midway bracing, so we didn't accidentally put it in different locations in each layer :p





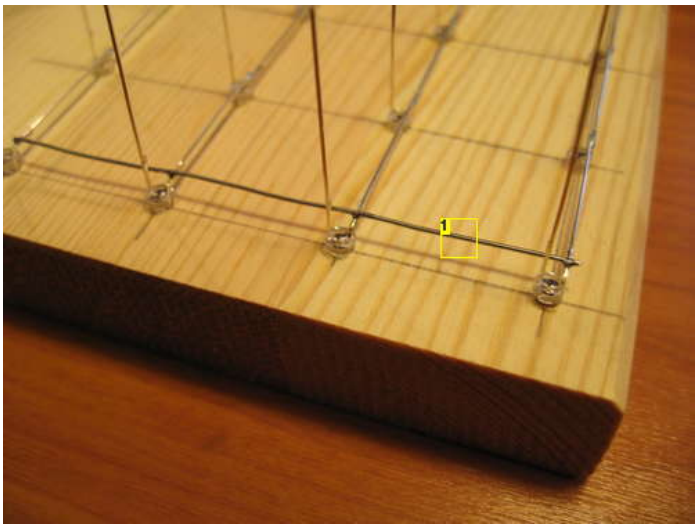


Image Notes
1. Brace

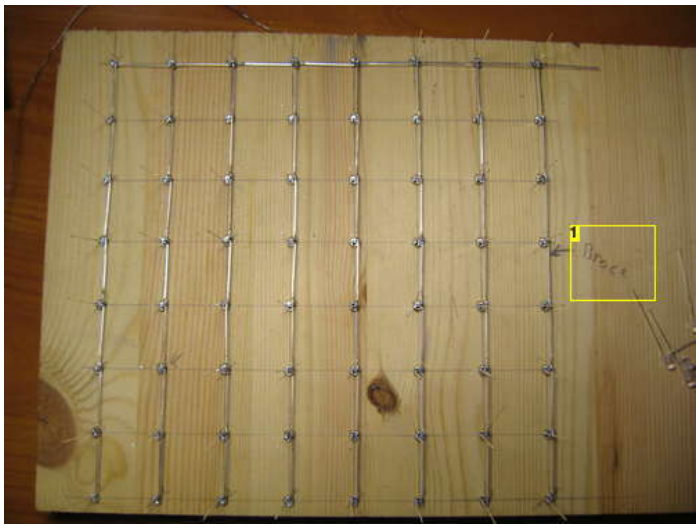


Image Notes
1. Almost done, just need the braces.

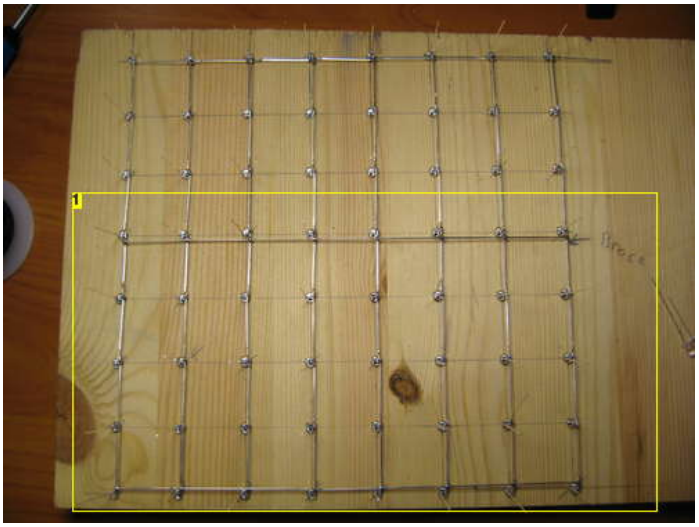
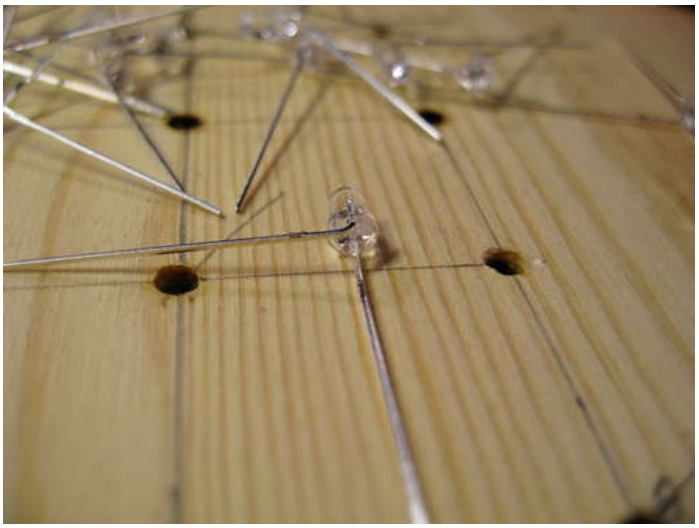
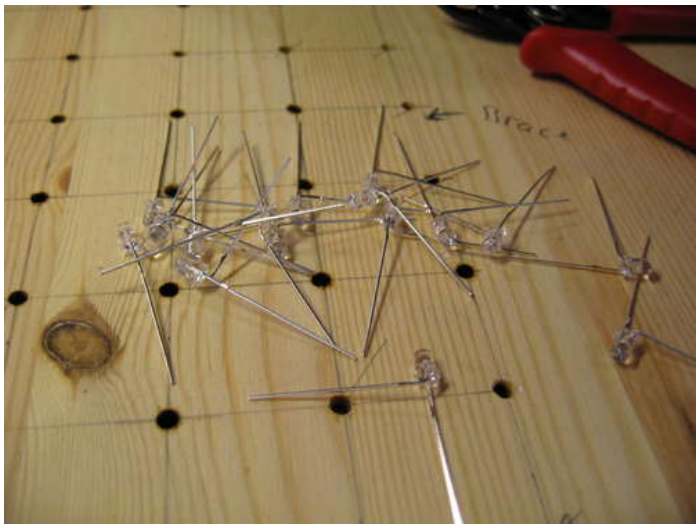


Image Notes
1. All done.





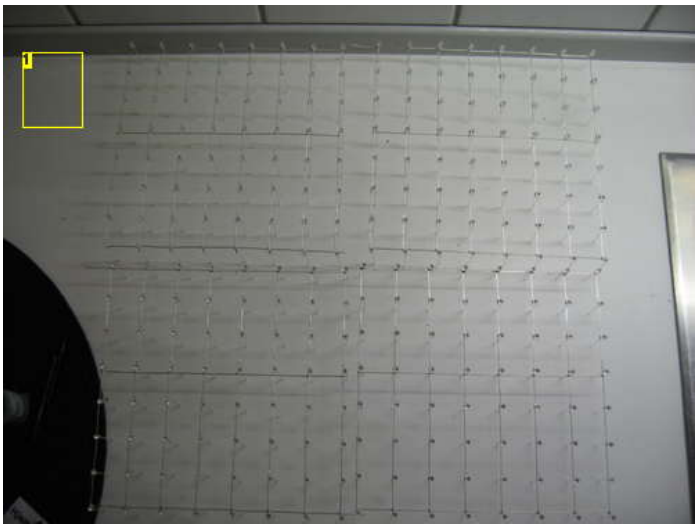


Image Notes
1. 4 down 4 to go!

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.

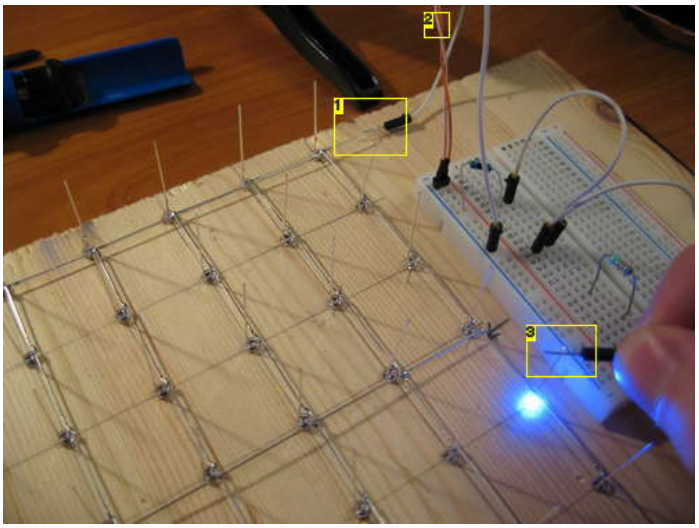


Image Notes
1. Ground connected to the layer
2. 5v from power supply
3. 5 volts via resistor.

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.

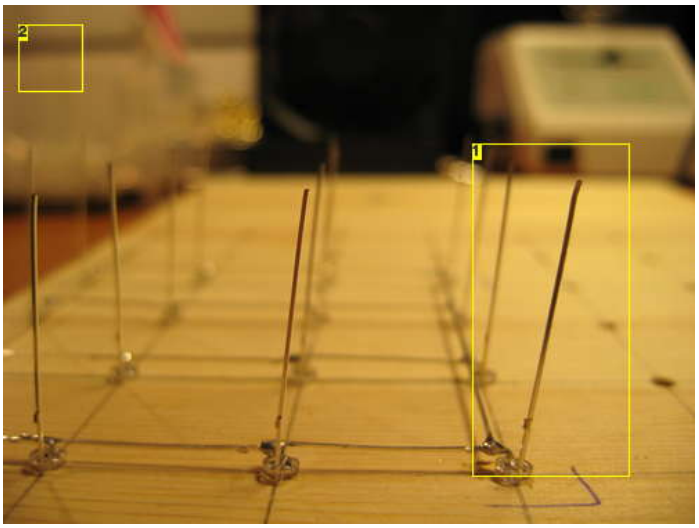


Image Notes
1. This isn't going to be a very nice LED cube!
2. We use a 4x4x4 cube here to demonstrate.

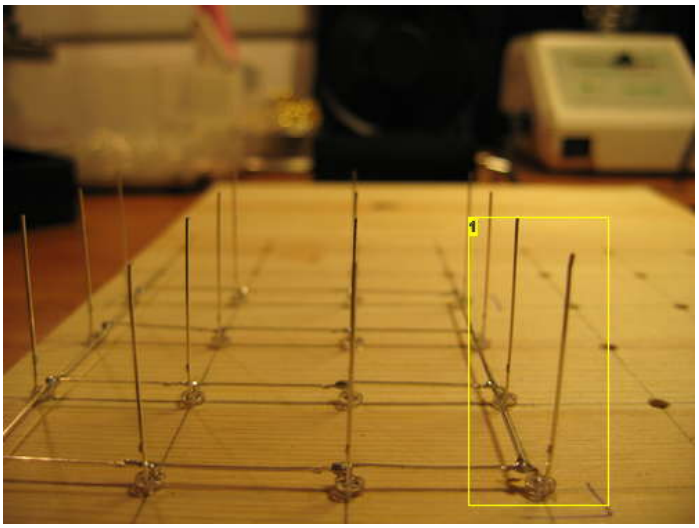


Image Notes
1. This is better

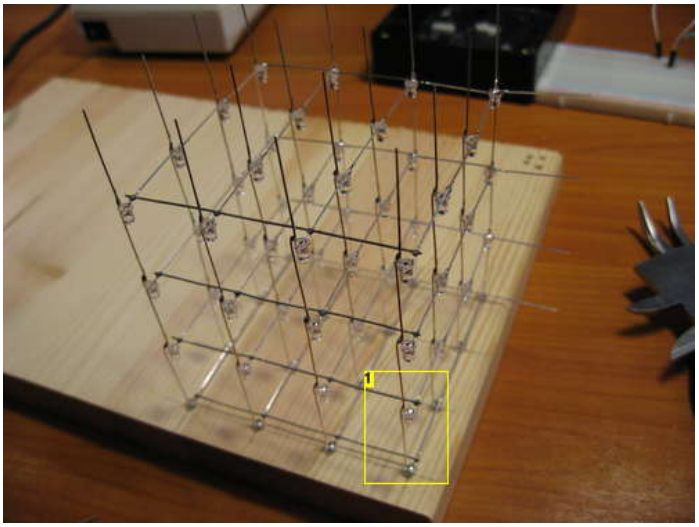
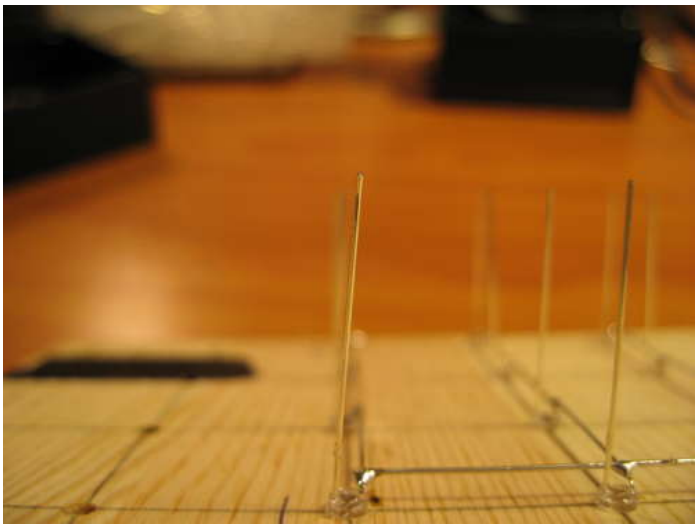
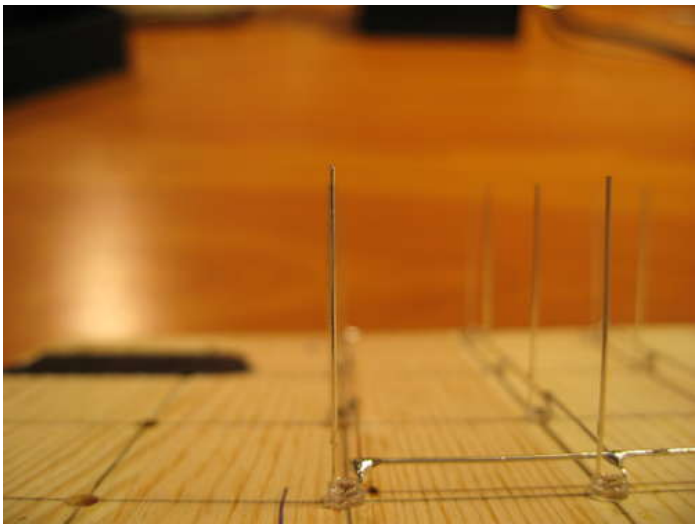


Image Notes
1. Pin straightening paid off.. see how straight the cube is





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.

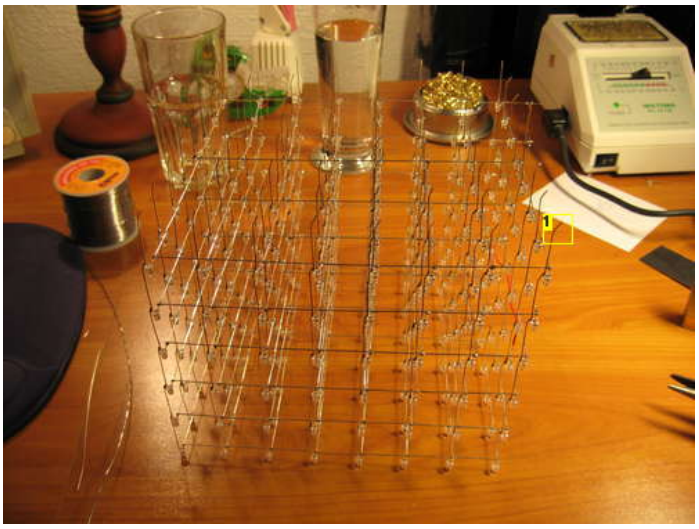


Image Notes
1. Pins are bent in order to make contact with the next LED

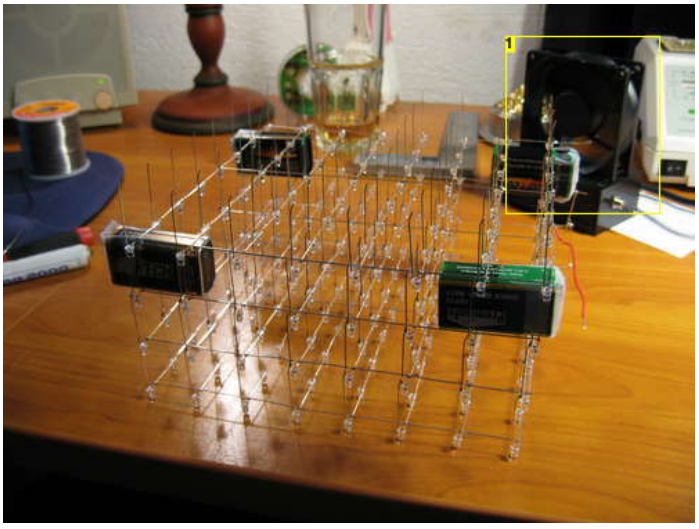


Image Notes

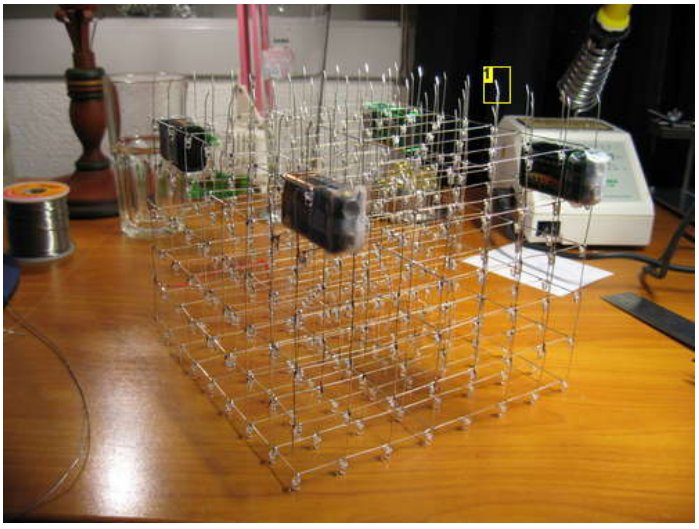


Image Notes

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.

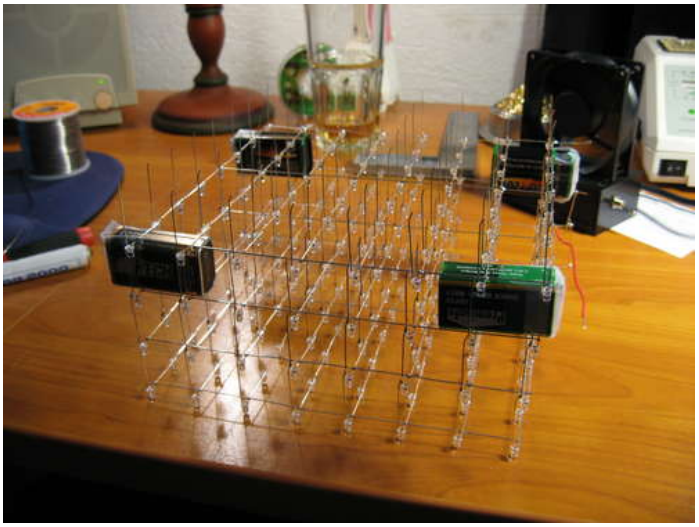
Connect ground to the the ground tab.

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!



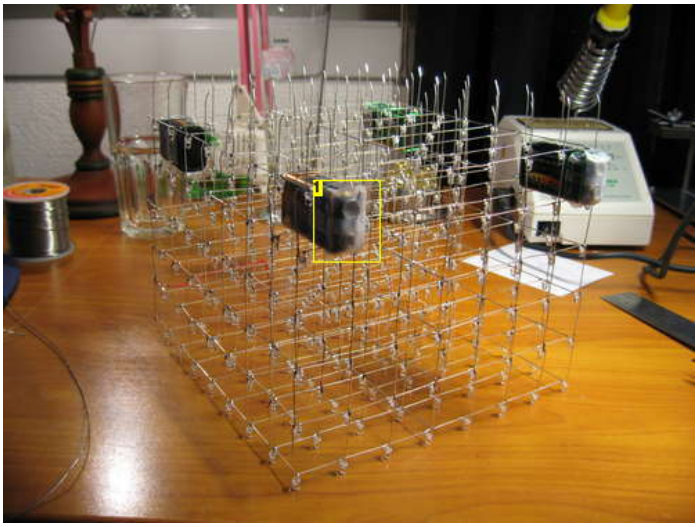
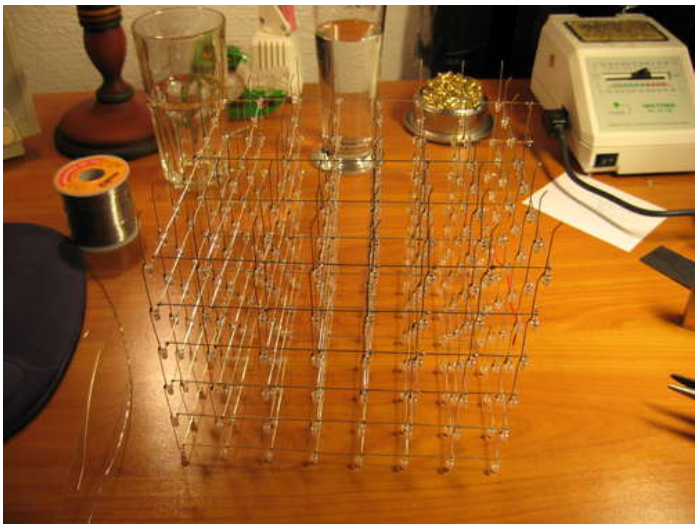
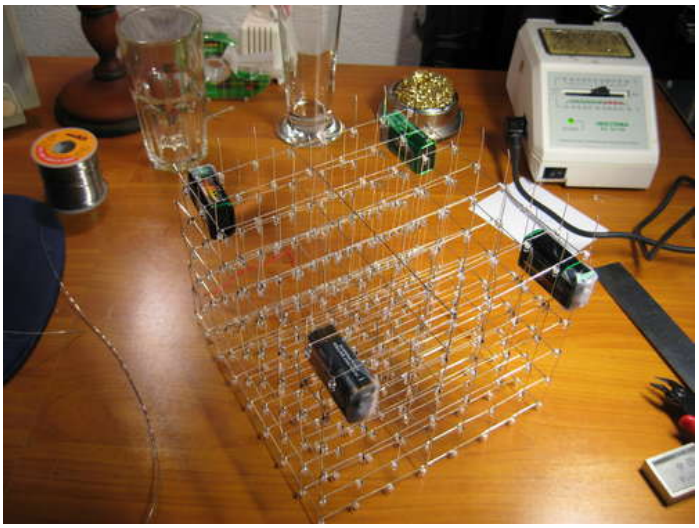
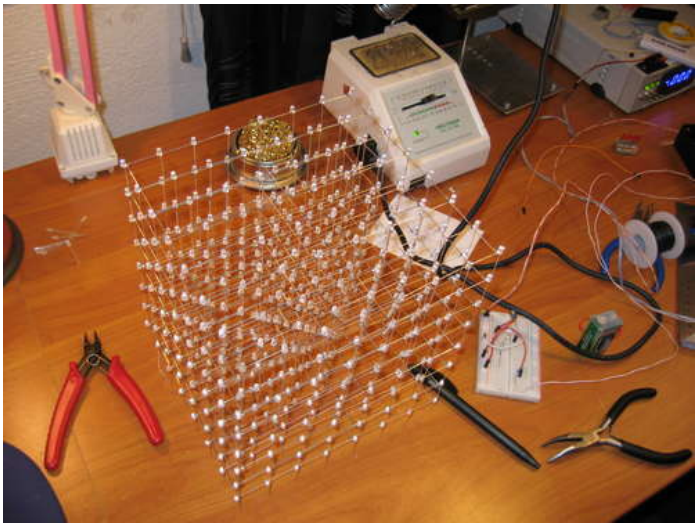


Image Notes
1. We taped over the battery terminals to avoid any disasters!







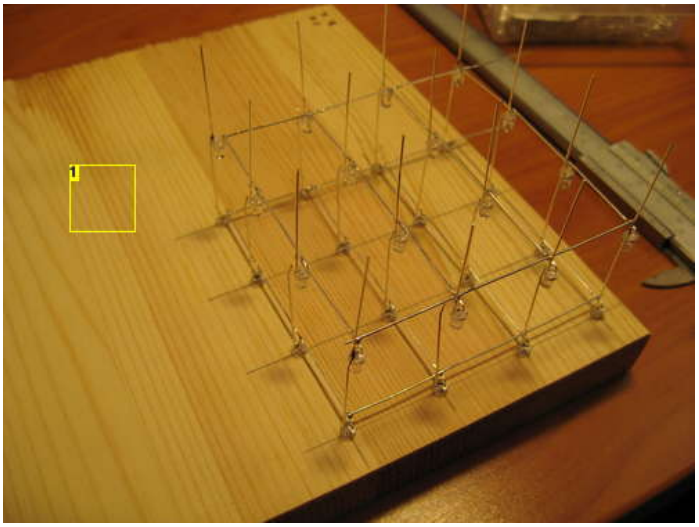
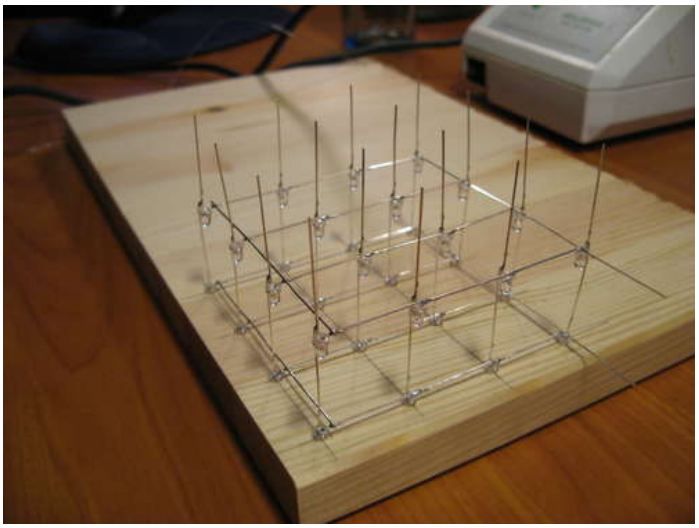
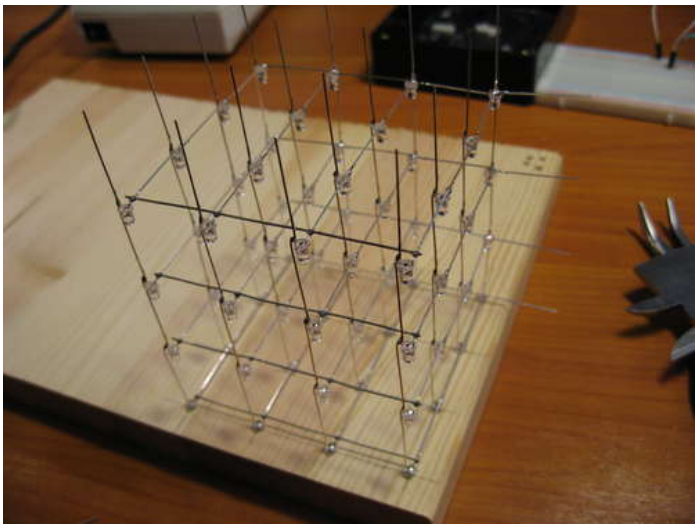
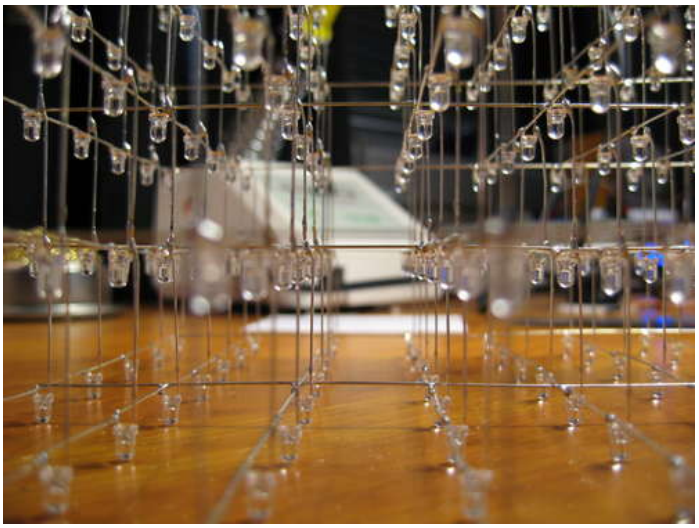


Image Notes
1. We added these 4x4x4 images to help illustrate the process.







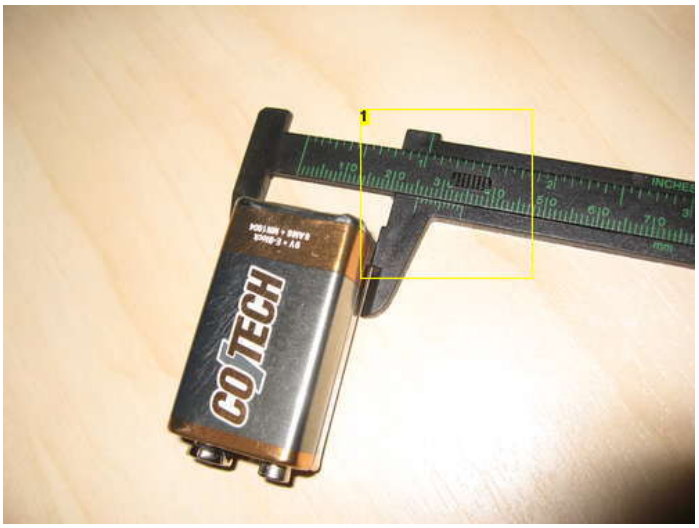


Image Notes
1. Almost exactly 25m!

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.

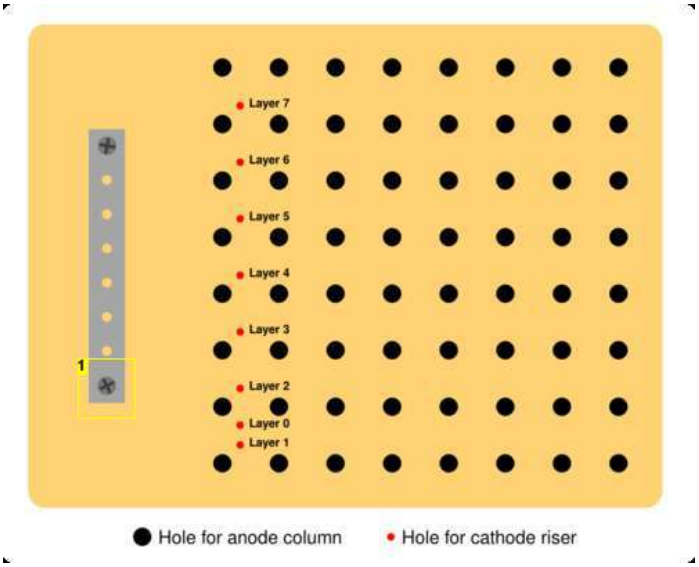


Image Notes
1. This is mounted on the underside of the board to hold the wires in place.

Image Notes
1. Drill all the way through.

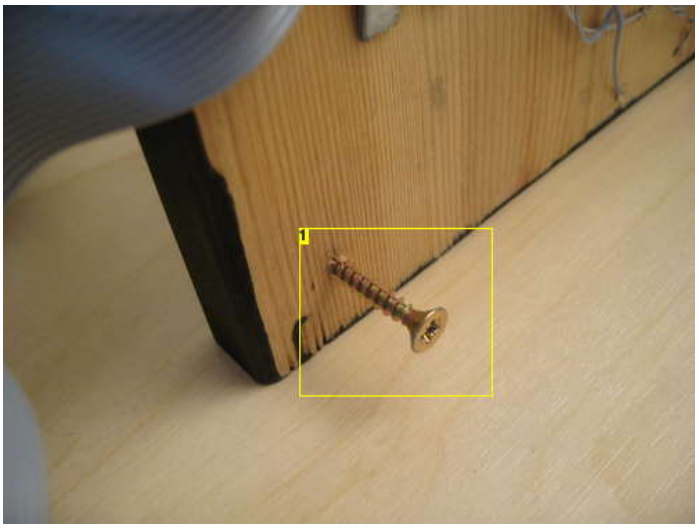


Image Notes
1. Didn't have any rubber feet that were high enough.

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.

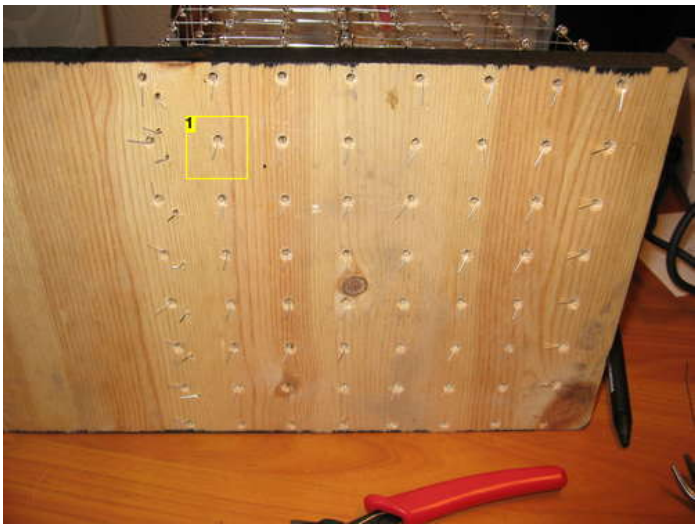


Image Notes
1. All the wires are bent 90 degrees. This is more than enough to hold the cube in place.

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.

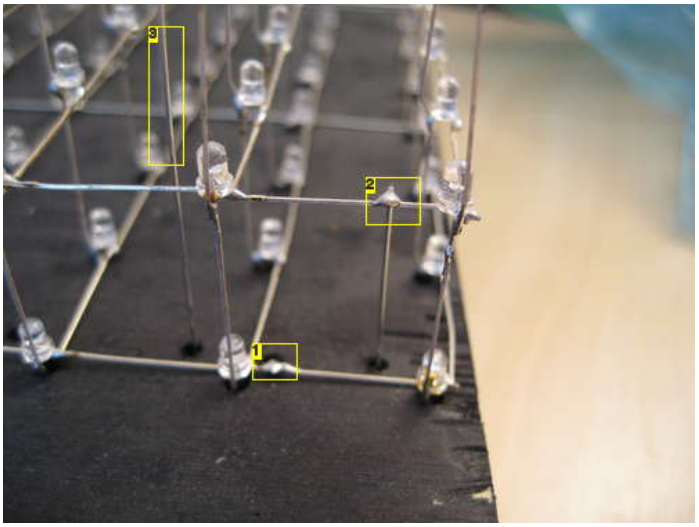


Image Notes
1. Ground wire for layer 0
2. Ground for layer 1
3. Ground for layer 2

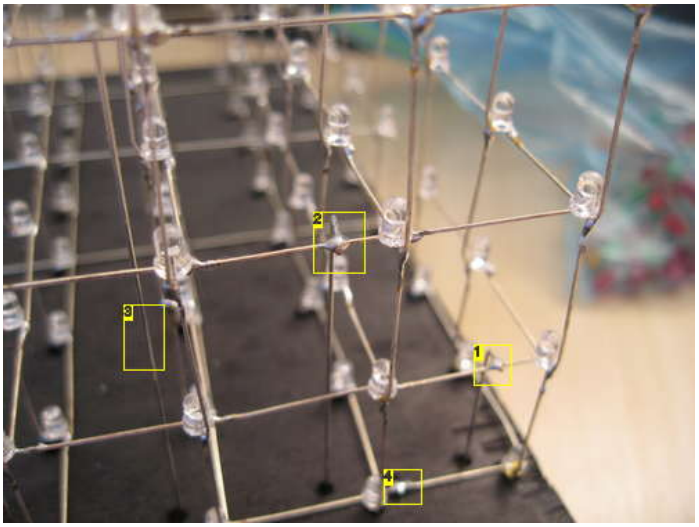


Image Notes
1. Layer 1
2. Layer 2
3. Layer 3
4. Layer 0

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.

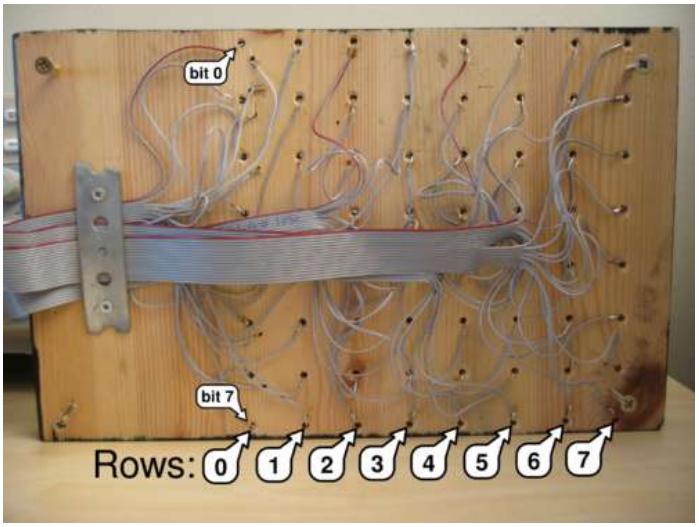
The cathodes are connected with 4 16-wire ribbon cables. Each of these ribbon cables are split in two at either end, to get two 8-wire cables.

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.



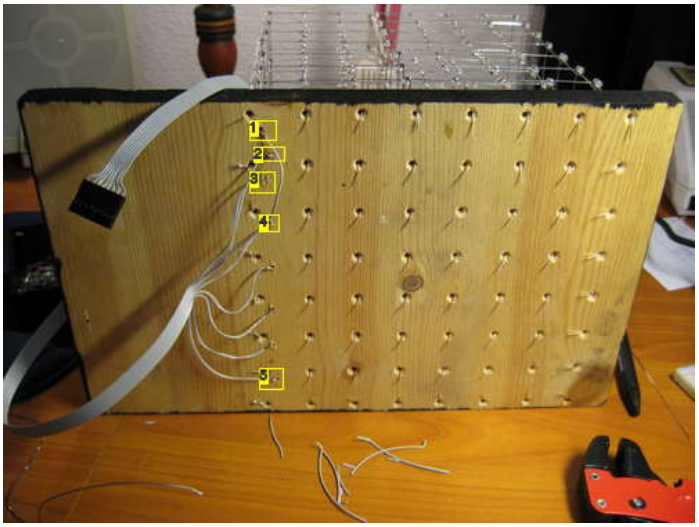
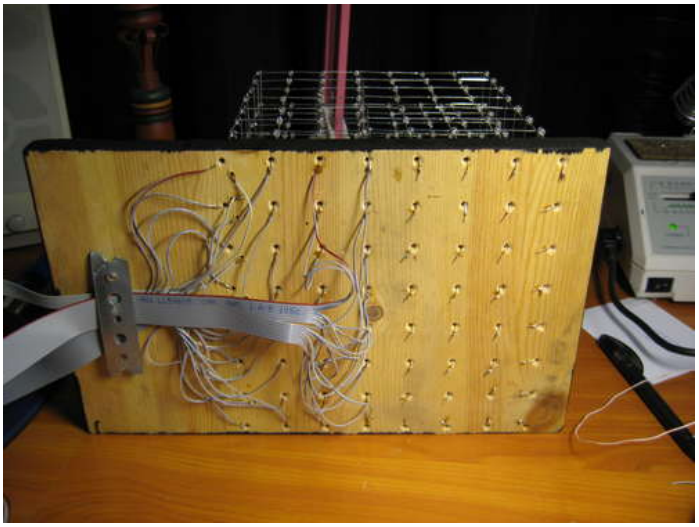
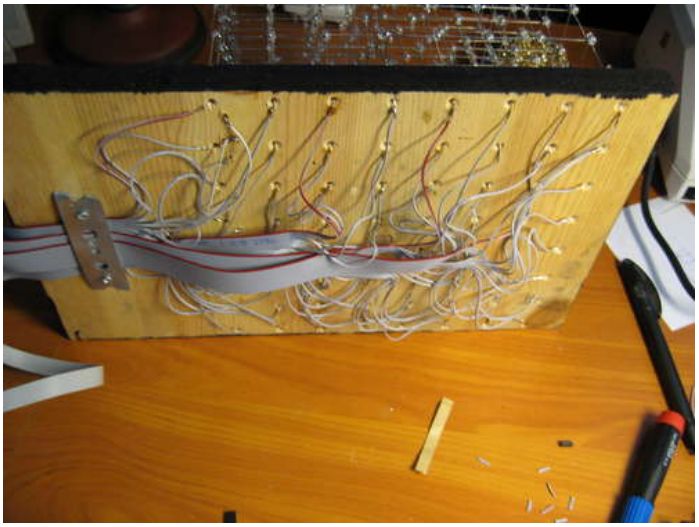
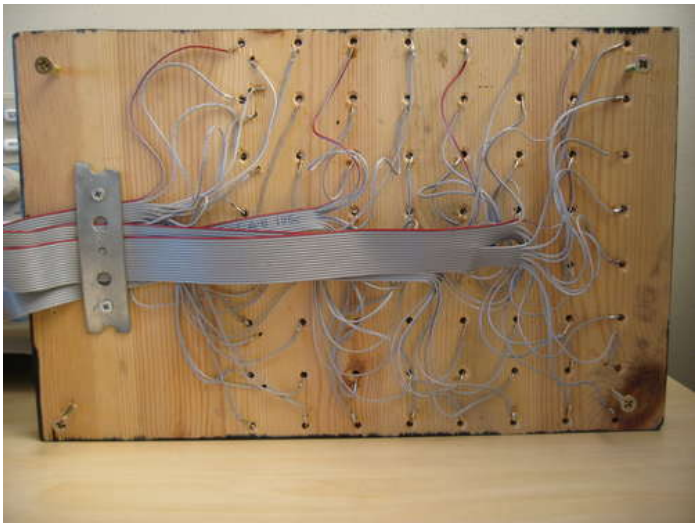


Image Notes
1. layer 1
2. Layer 0
3. layer 2
4. layer 4
5. layer 7







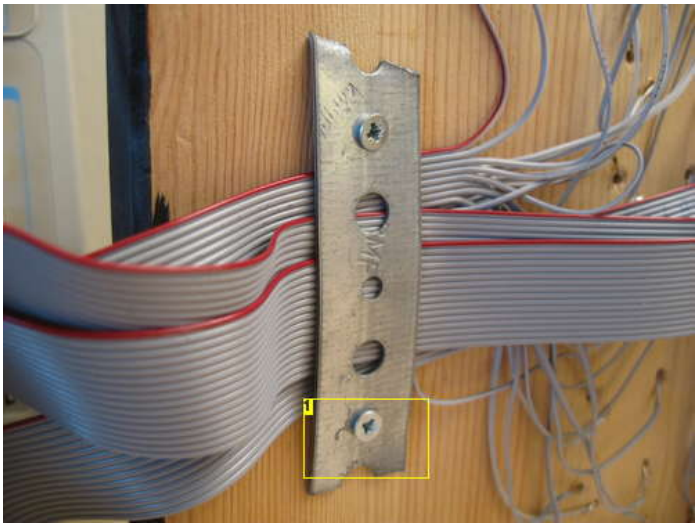
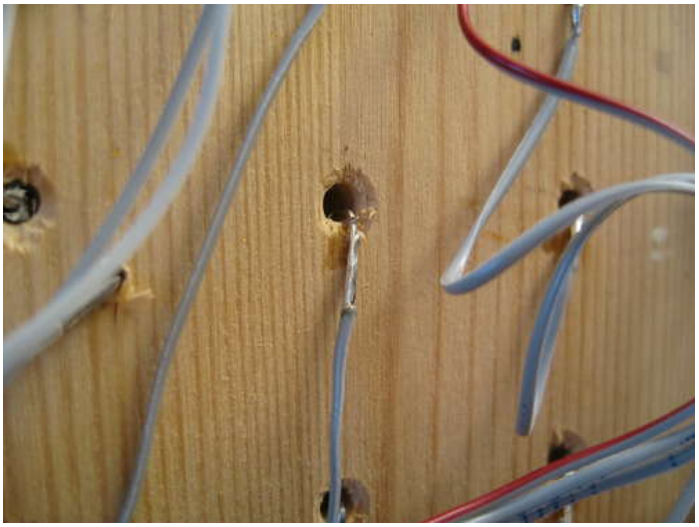
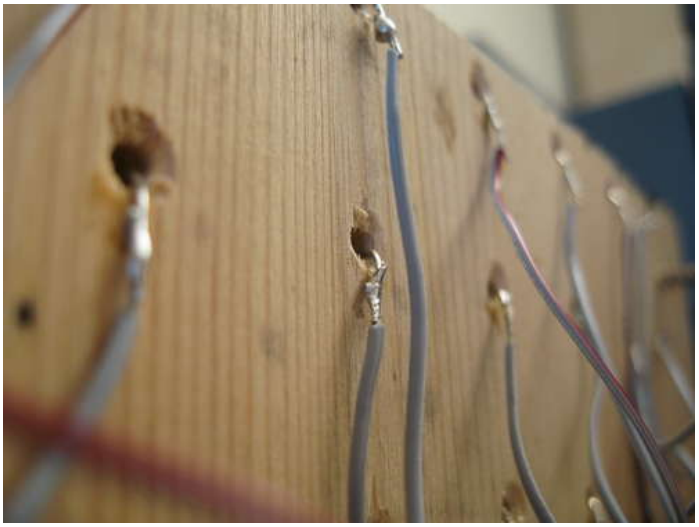
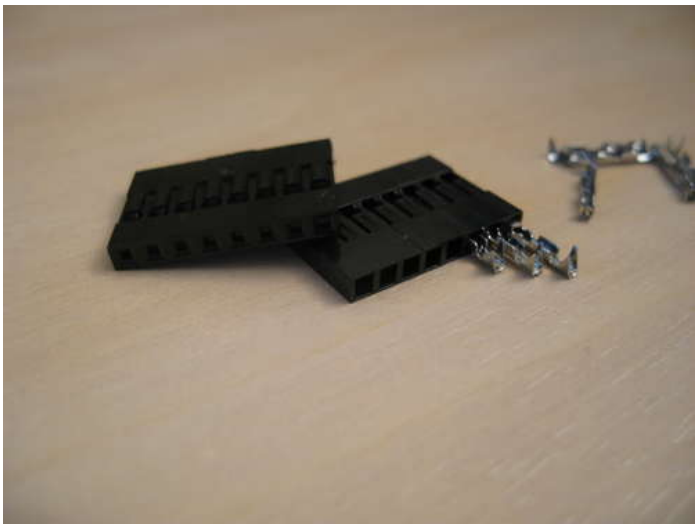


Image Notes
1. The connections are a bit flimsy. The cube will last a lot longer with this strain relief.







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.

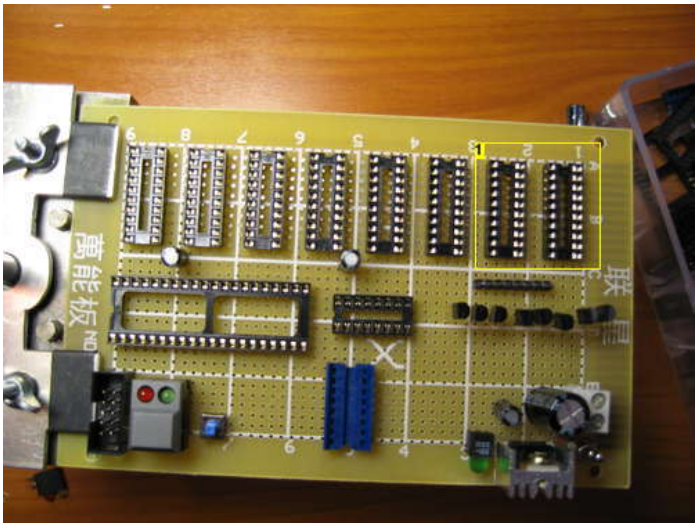


Image Notes
1. Way to little space in between the ICs. No room for resistors and connectors.

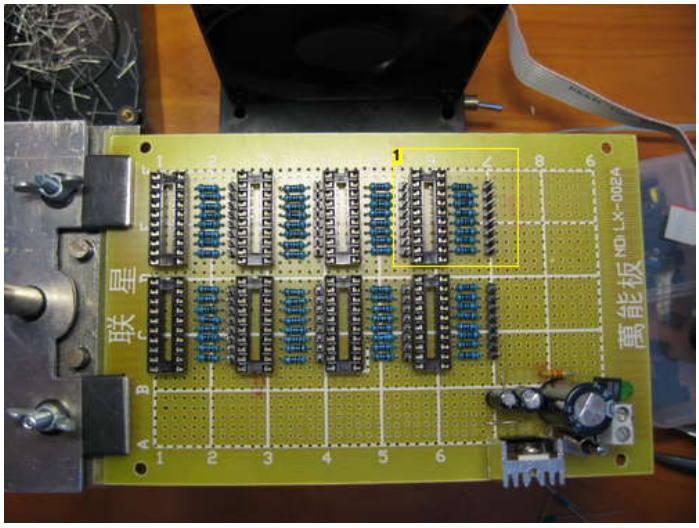
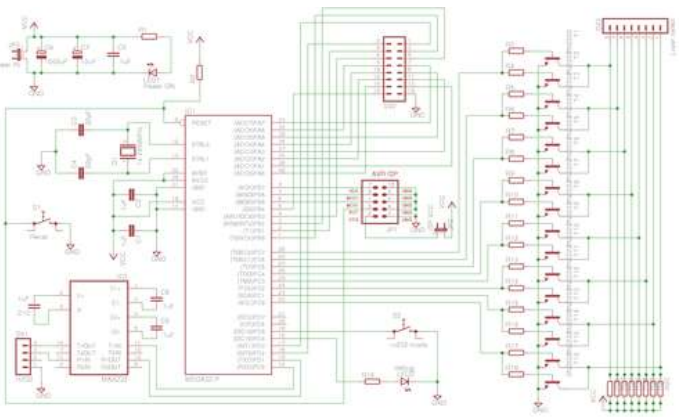
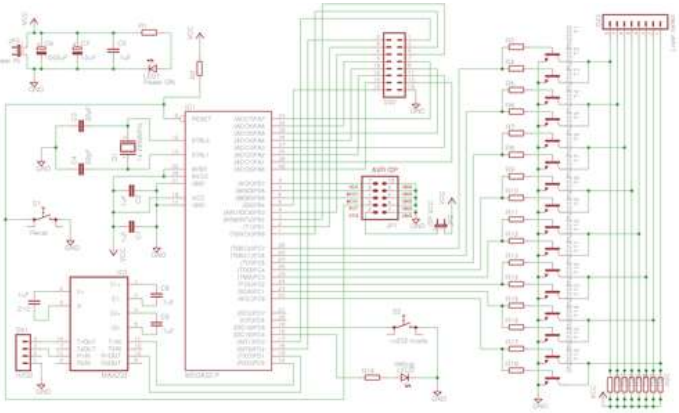
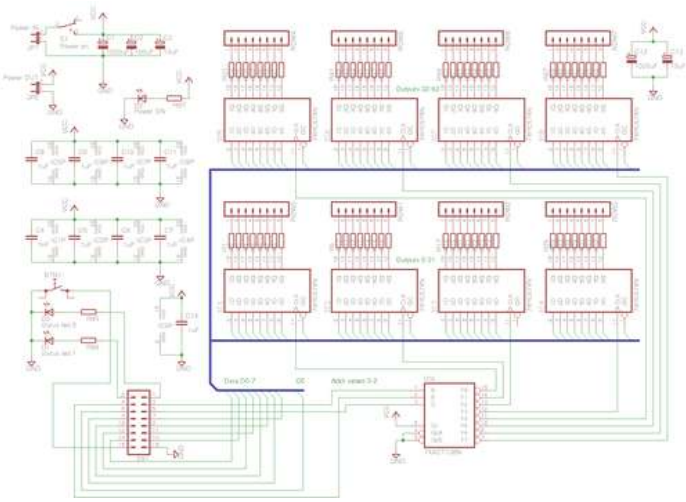


Image Notes
1. This is better.







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.

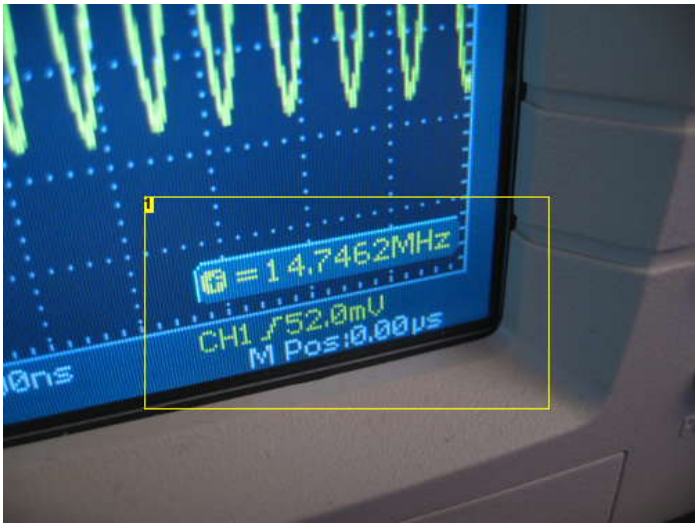
To get flawless serial communication, you have to use a clock frequency that can be divided by the serial frequency you want to use.

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!



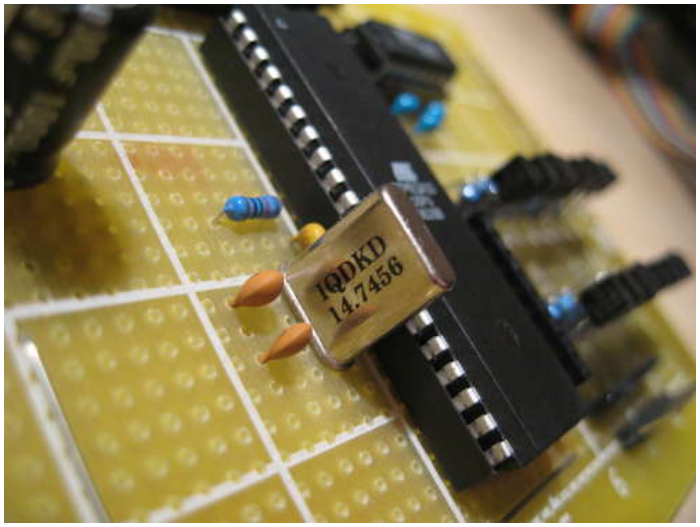


Image Notes
1. This is the frequency of the system clock

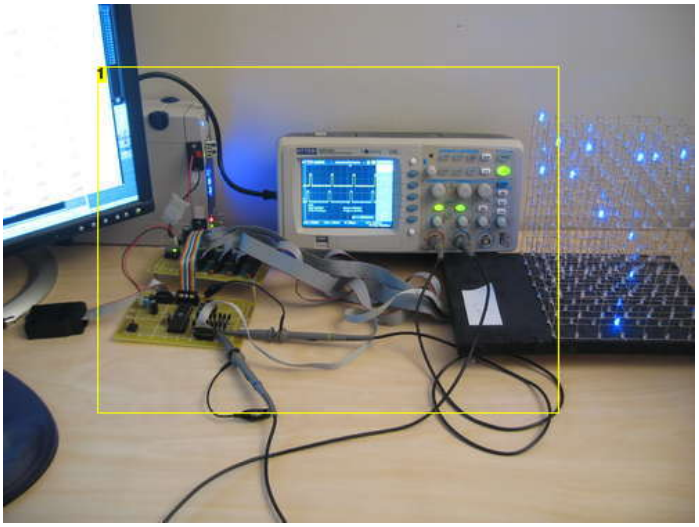


Image Notes
1. I got an oscilloscope for Christmas :D we used it to visualize some of the signals in the LED cube.

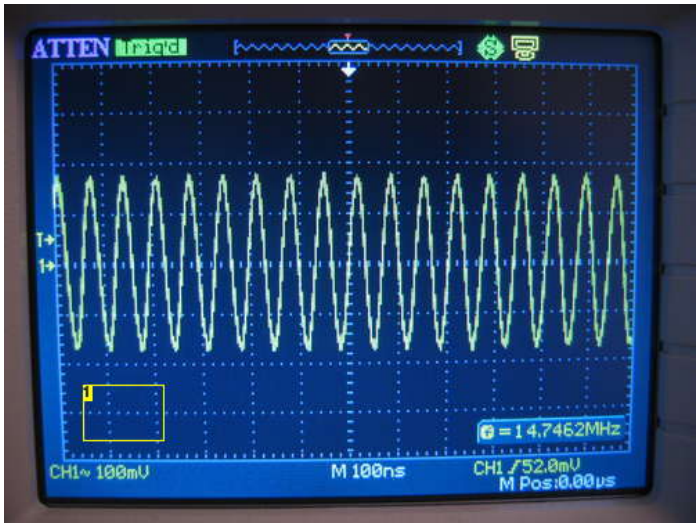


Image Notes
1. This is what the clock signal from a crystal looks like

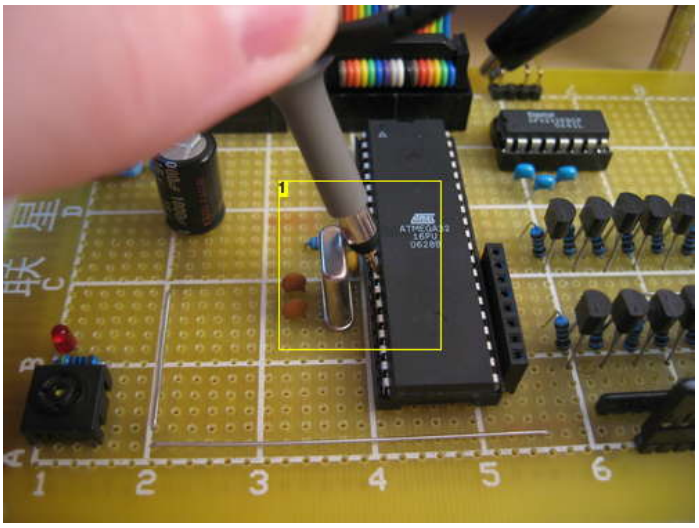


Image Notes
1. Probing the crystal

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 ;)

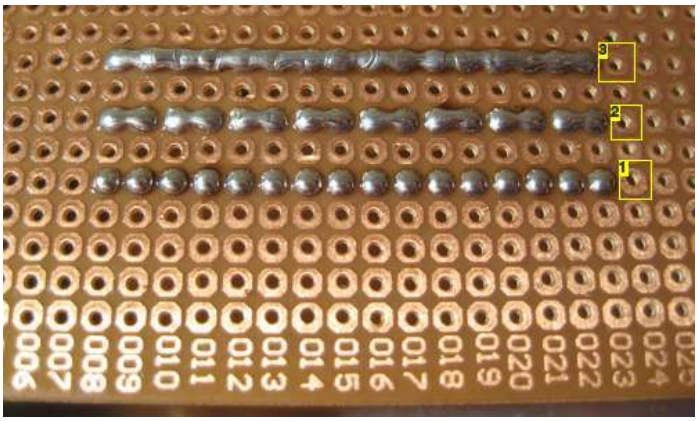


Image Notes
1. 1
2. 2
3. 3

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.

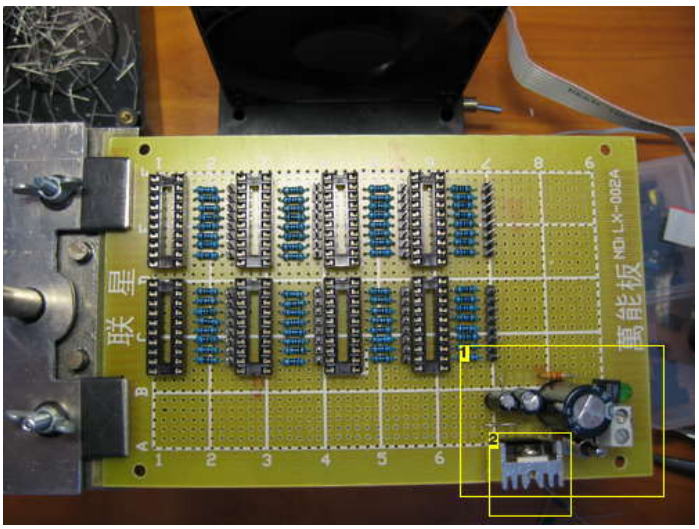


Image Notes
1. Power supply

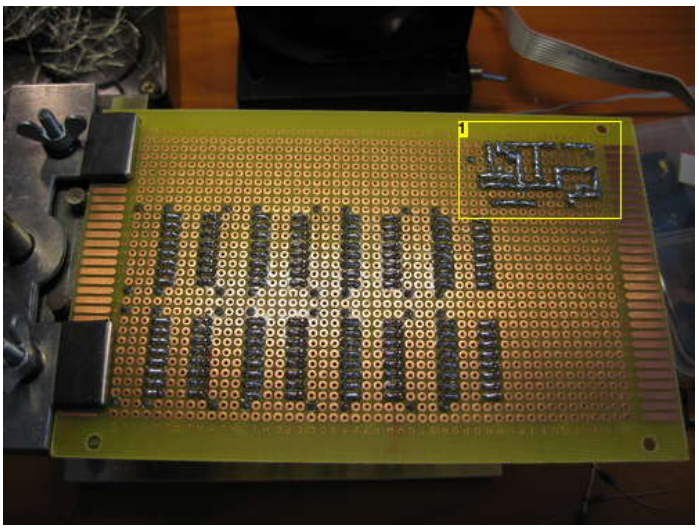


Image Notes
1. Bottom side of power supply. See, only solder traces. No wires.

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

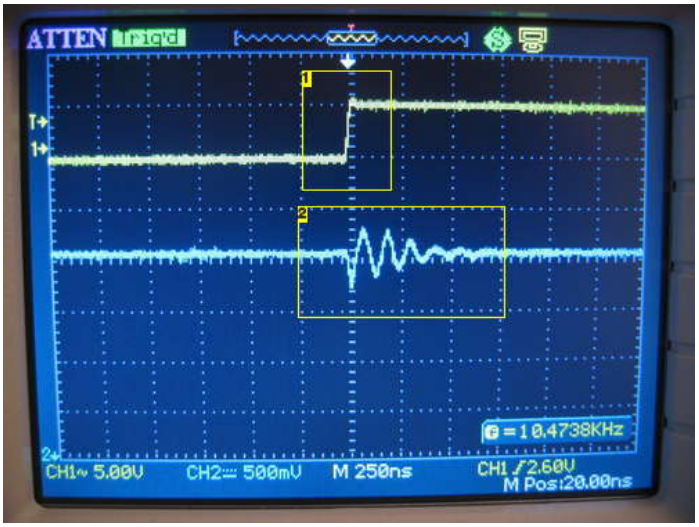


Image Notes
1. A layer in the led cube is switched on.
2. The resulting rise in current draw makes VCC fluctuate a little

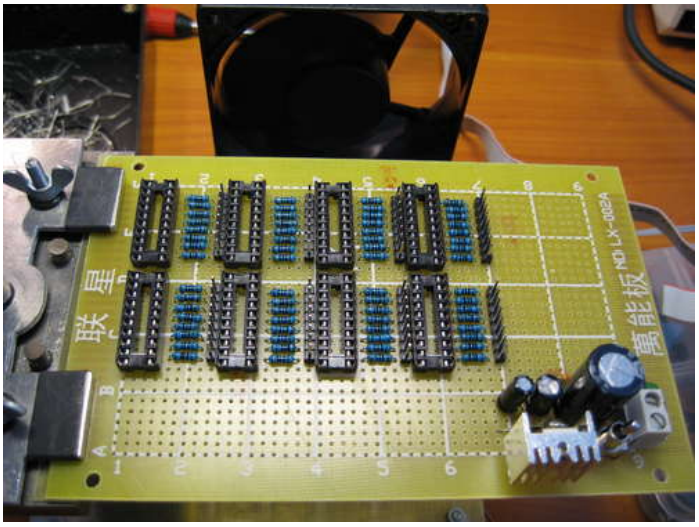
Step 34: Build the controller: IC sockets, resistors and connectors

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.



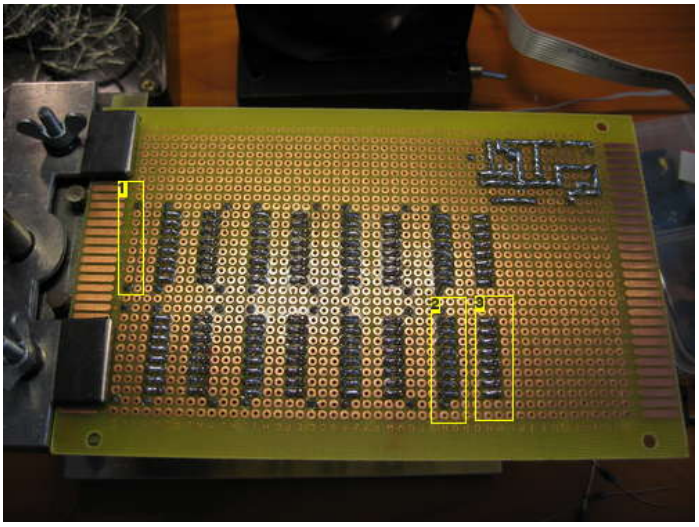


Image Notes
1. Input side not soldered yet.
2. Resistor soldered to IC
3. Resistor soldered to connector

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.

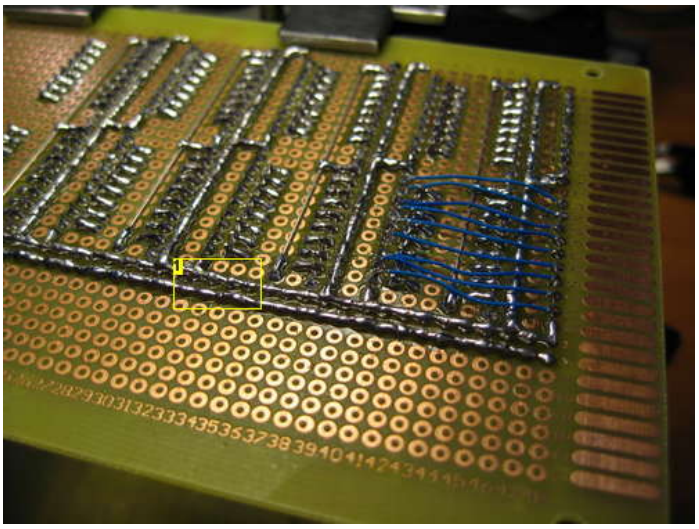


Image Notes
1. GND and VCC runs along the length of the board.

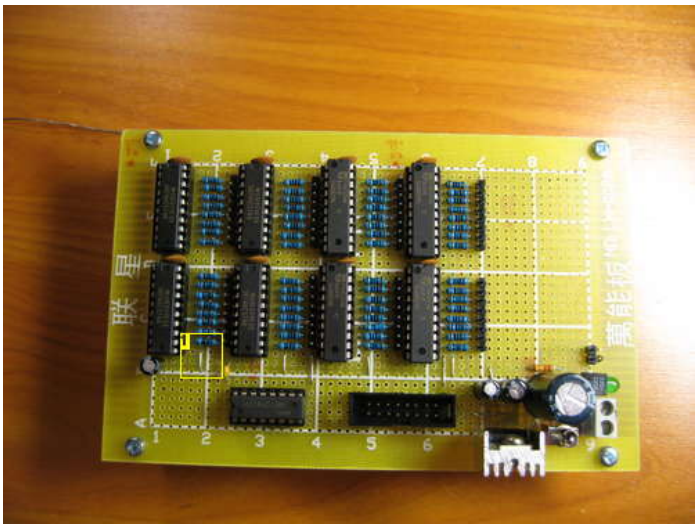


Image Notes
1. VCC crosses GND once for each row of ICs

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.

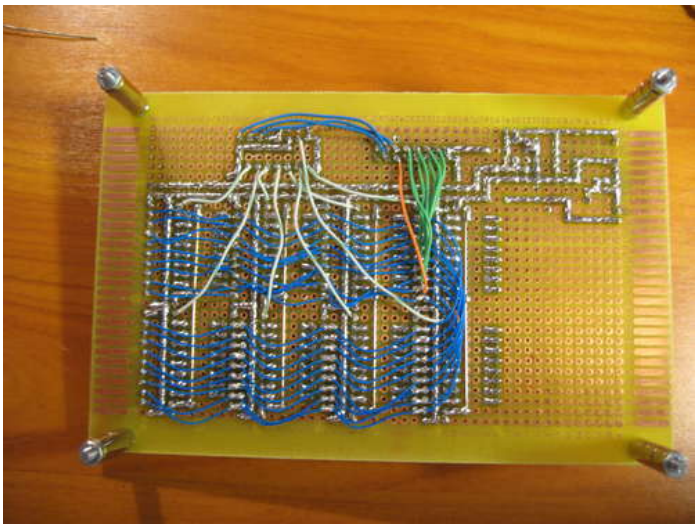
At the top of the board, we have added a 16 pin connector. This connects the latch board to the micro controller board. Next to that, you see the 74HC138.

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.



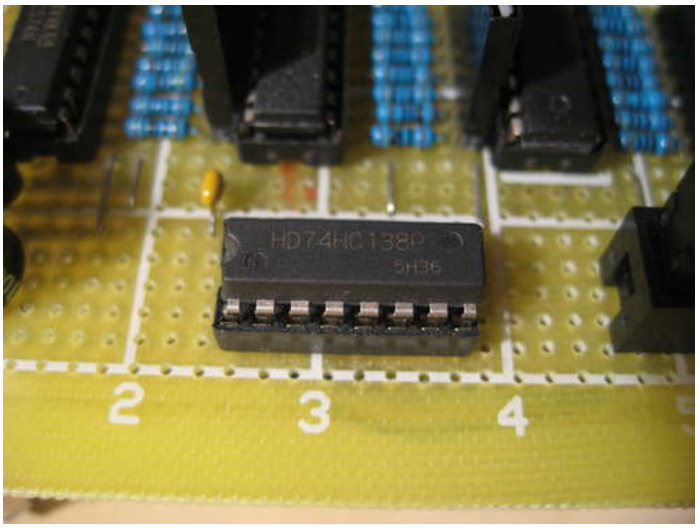
Step 37: Build the controller: Address selector

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.



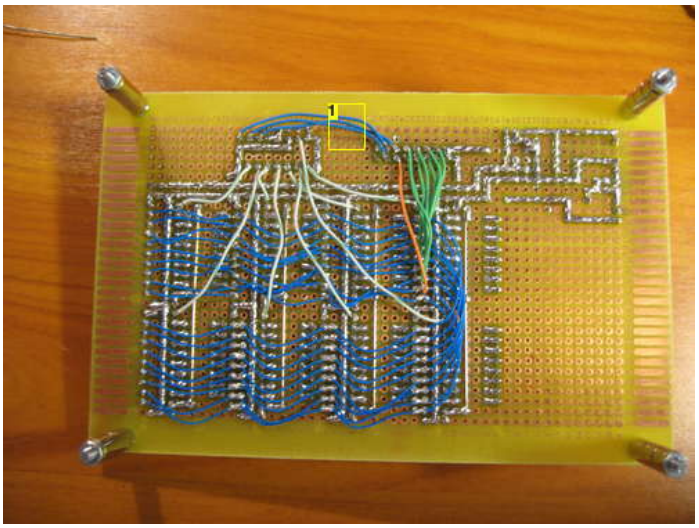


Image Notes
1. 3 bit address select bus

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!

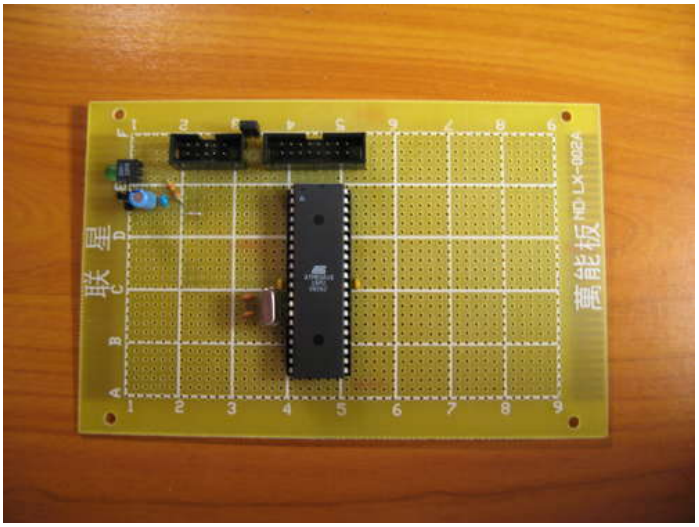
The second image shows the underside. Again all GND and VCC lines are soldered as traces on the protoboard or bare wire. We had some more left over straight metal wire, so we used this.

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.



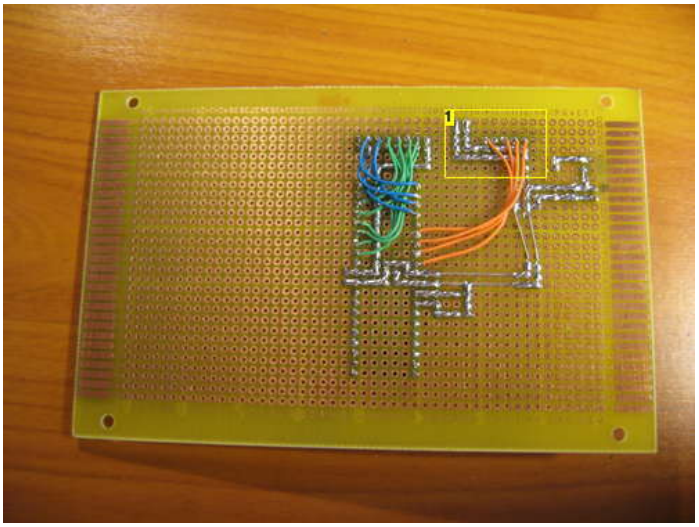
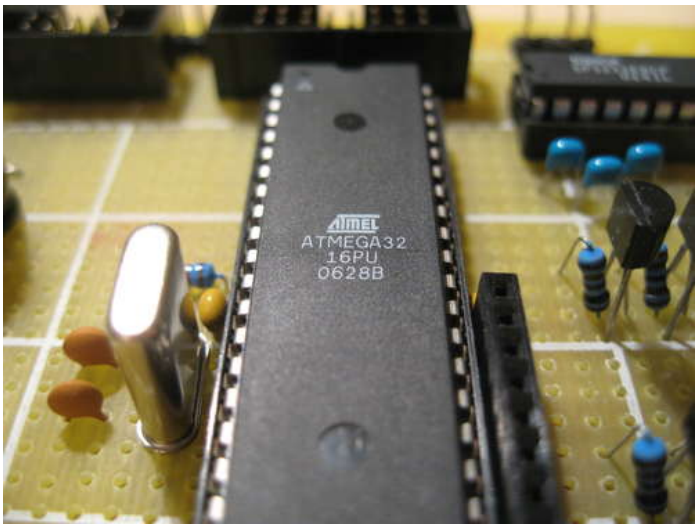
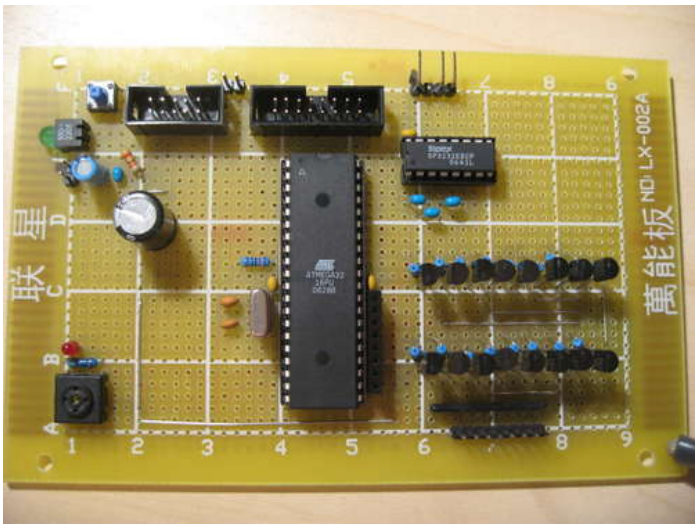


Image Notes
1. In circuit serial programming header.





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.

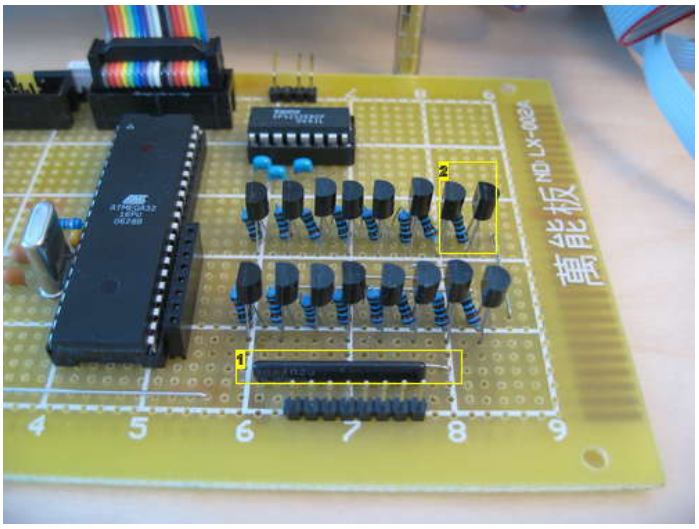


Image Notes
1. Pull up resistors. This type of resistor is called a resistor network. It just has a bunch of resistors connected to a common pin.
2. Two and two resistors work together.

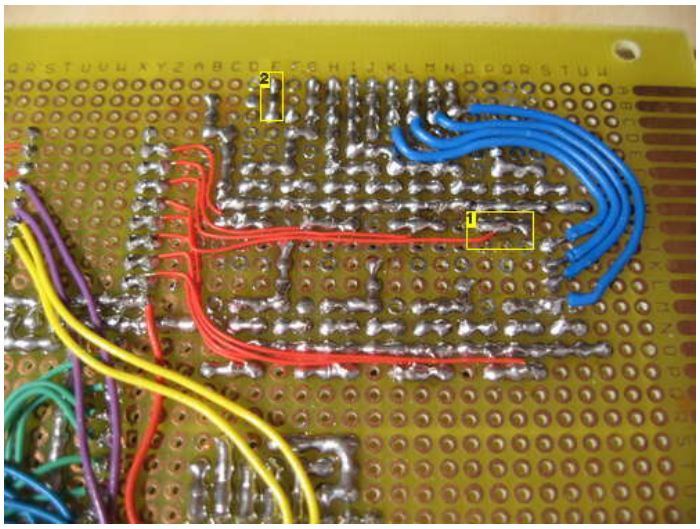
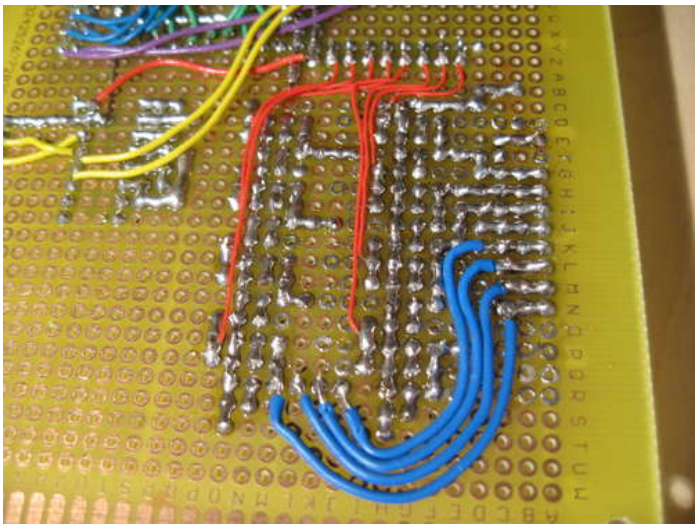


Image Notes
1. Signal goes to two transistors.
2. This point was connected to VCC after this picture was taken.



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:

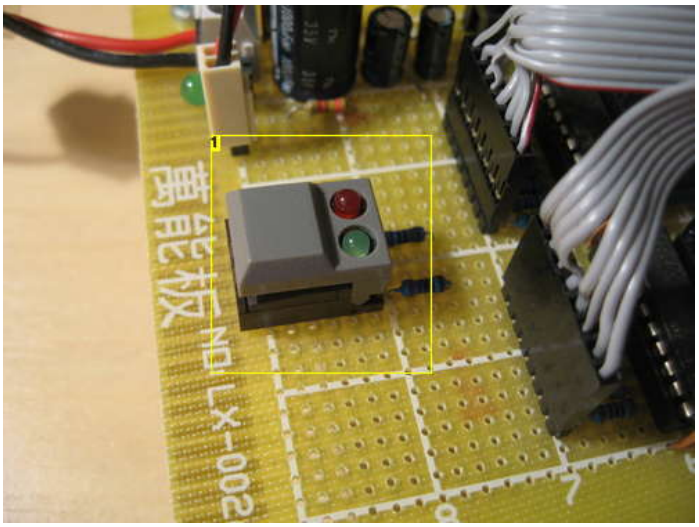


Image Notes
1. Start the cube in autonomous mode

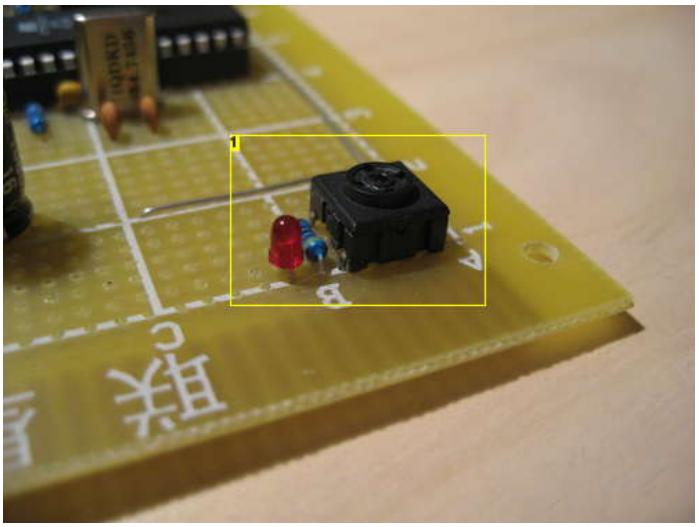


Image Notes
1. Start the cube in rs232-mode

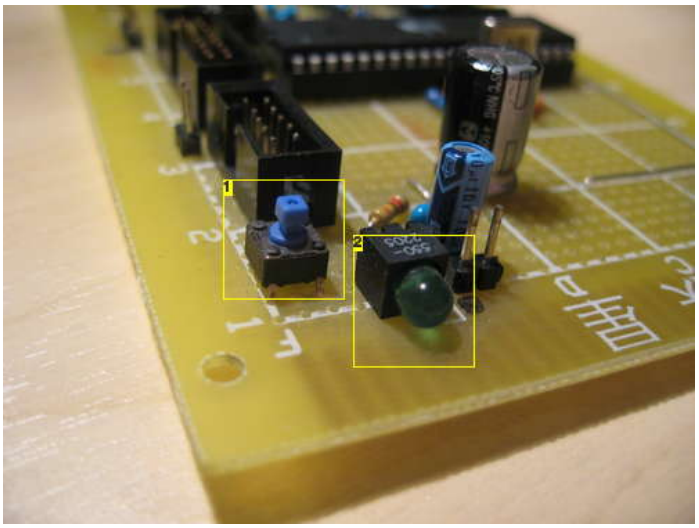


Image Notes
1. Reset
2. Power on

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.

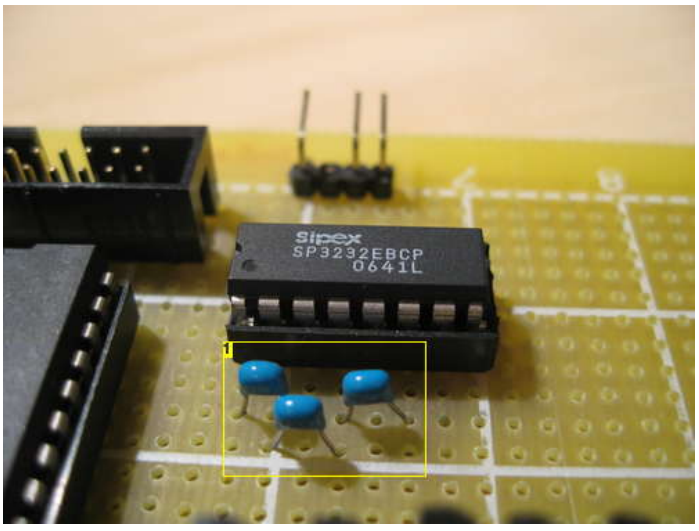


Image Notes
1. These capacitors helps the max232 bump the voltage up to rs232 levels.

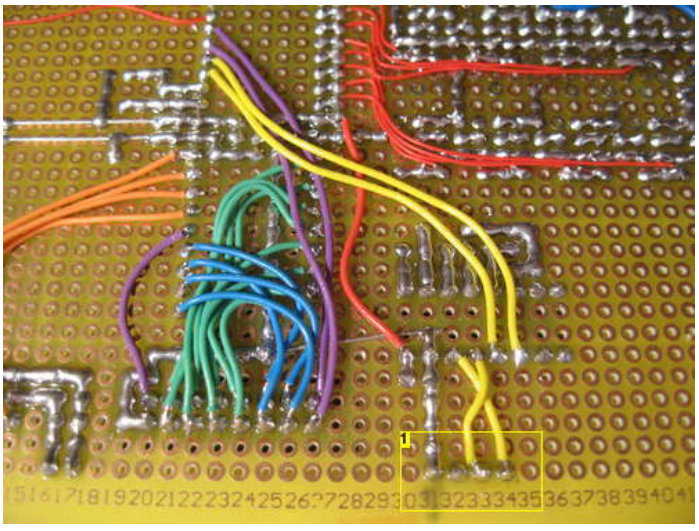


Image Notes
1. RS232 connector

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.

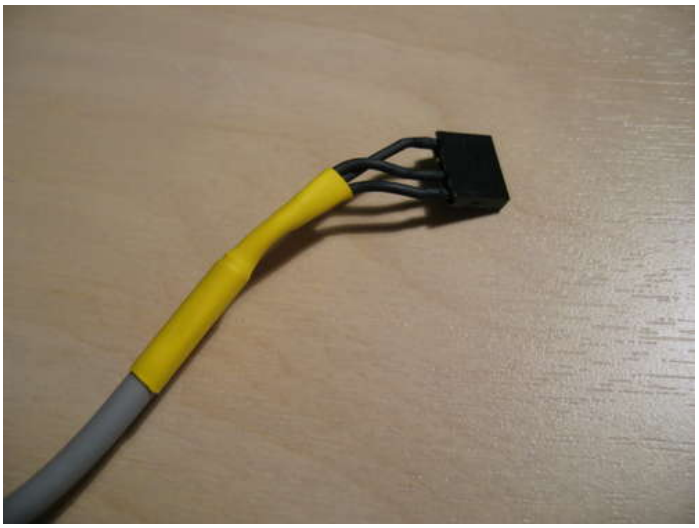
8) Pre-tin the wires to make soldering easier. Just apply heat and solder to the stripped wires.

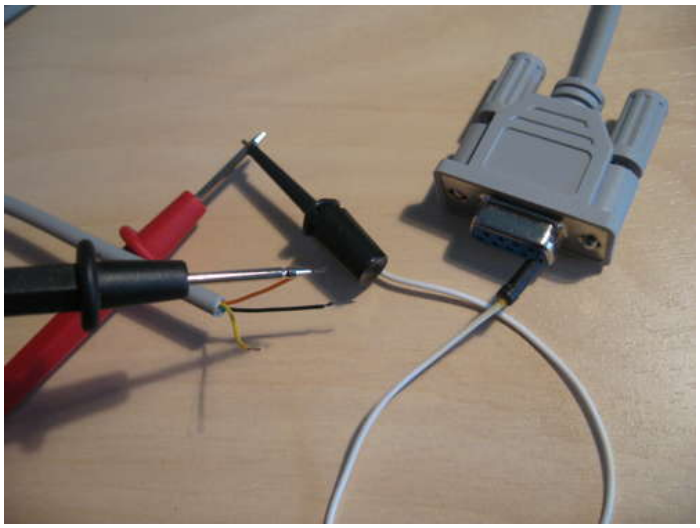
9) Slide a shrink tube over the cable. Slide three smaller shrink tubes over the individual wires.

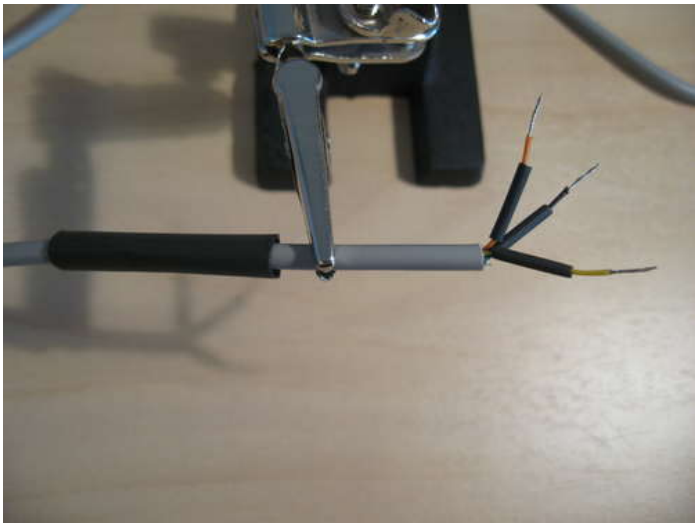
10) Solder the wires to the connector.

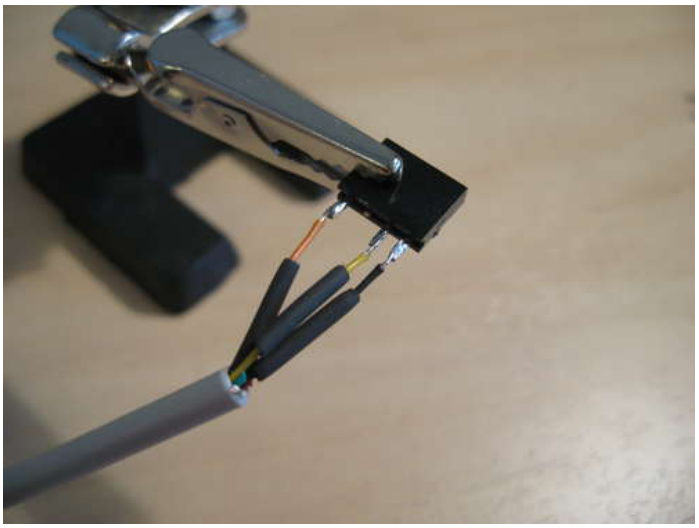
11) Shrink the smaller tubes first, then the large one. If you use a lighter, don't hold the shrink tube above the flame, just hold it close to the side of the flame.

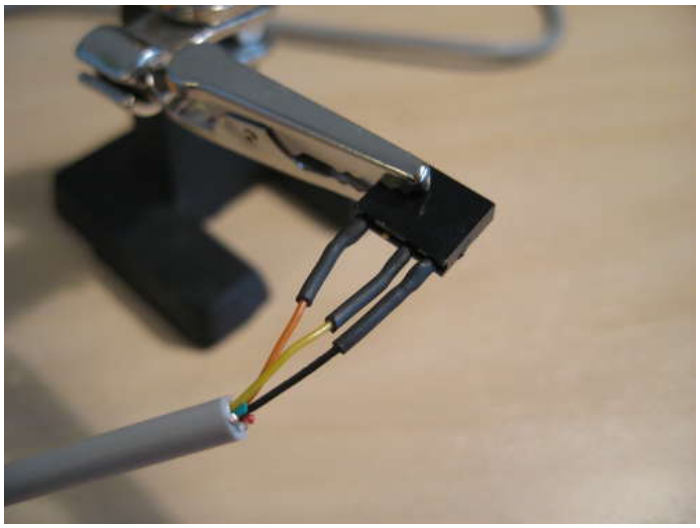
Don't make your cable based on the colors we used. Test the cable to find the correct colors.











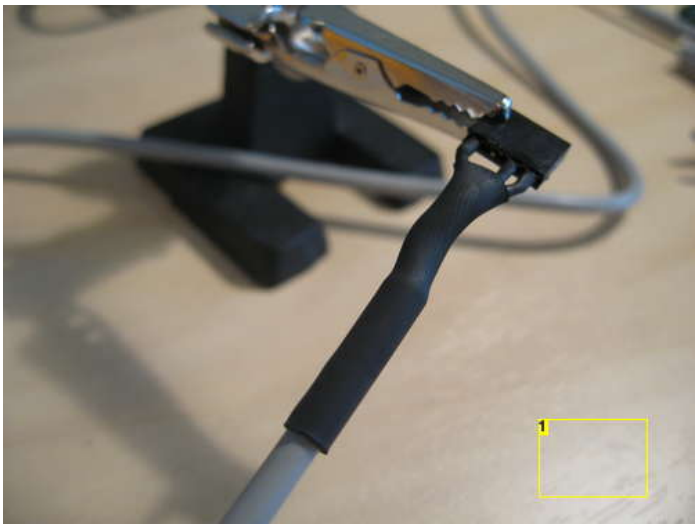


Image Notes
1. We managed to get the colors wrong on the first try. That's why the cable in the first picture has a yellow shrink tube ;)

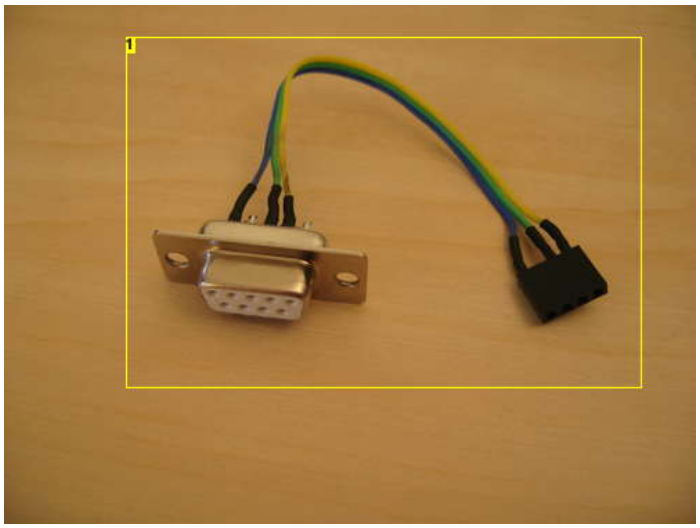


Image Notes
1. This is another way of doing it..

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.

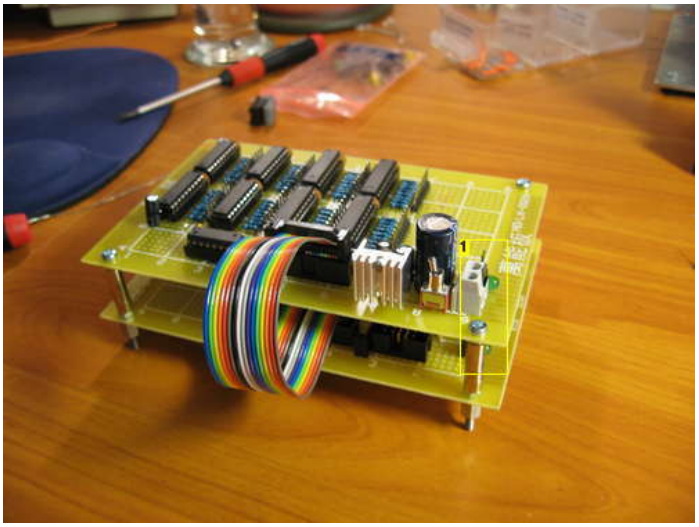
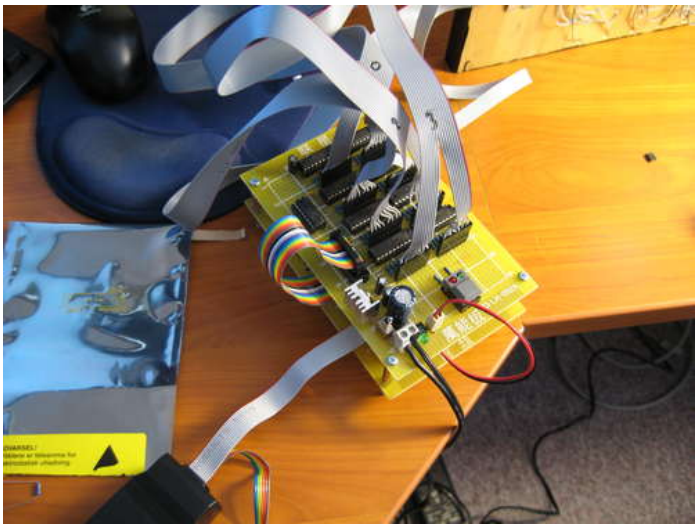
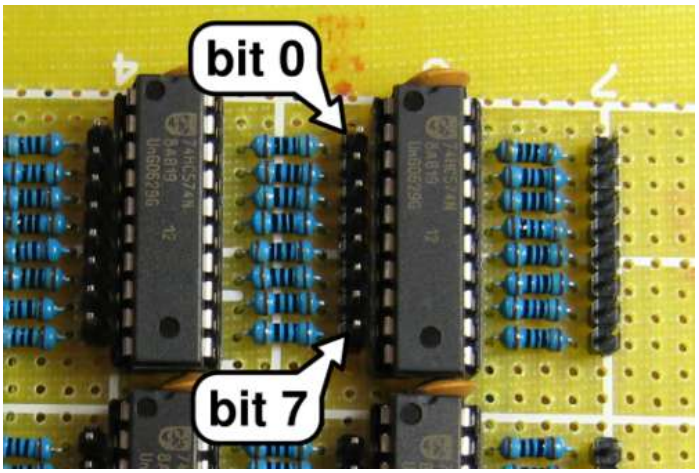


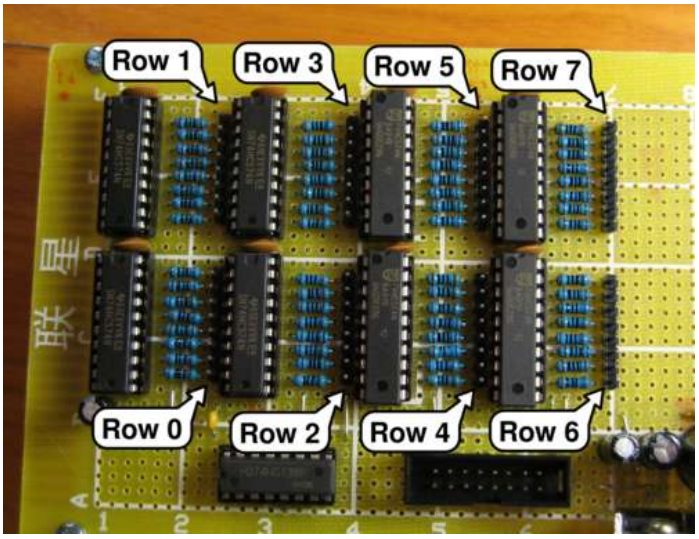
Image Notes
1. The GND/VCC cable connects between the two 2pin headers here.

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.







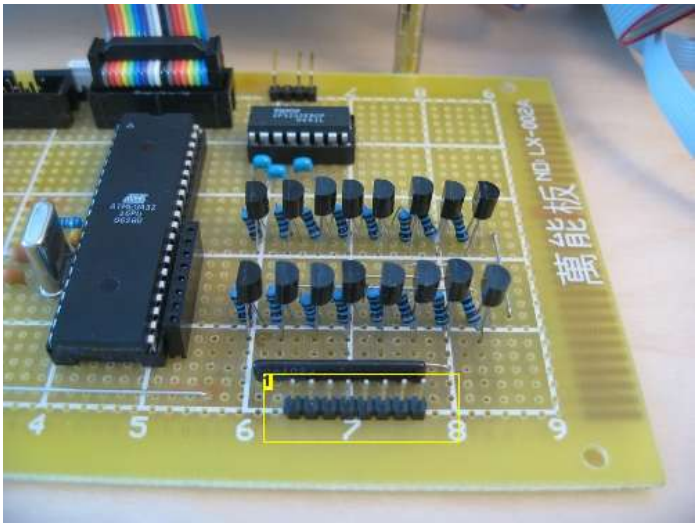


Image Notes
1. The ground layer ribbon cable connects here. Just connect it the other way if your LED cube is upside-down ;)

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!

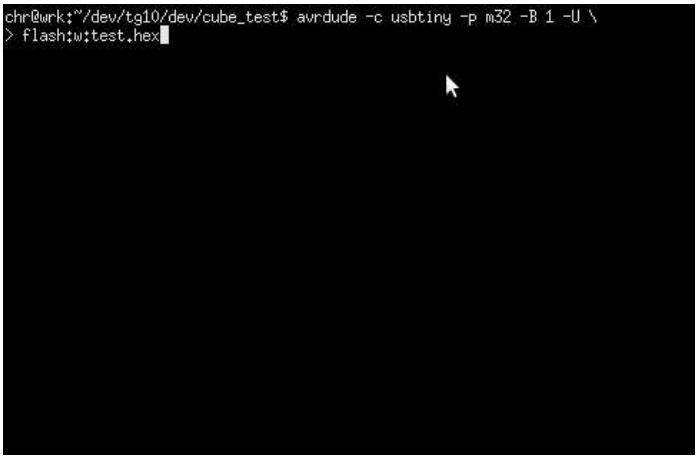
We have prepared a simple test program to check if all the LEDs work and if they are wired correctly.

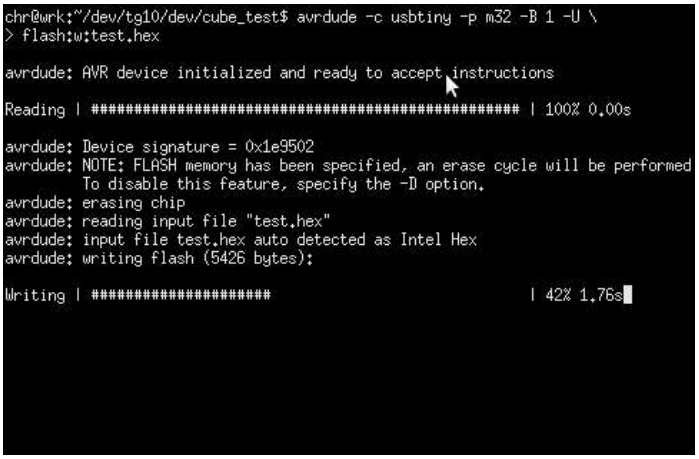
You can download the firmware test.hex in this step, or download the source code and compile it yourself.

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.





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.

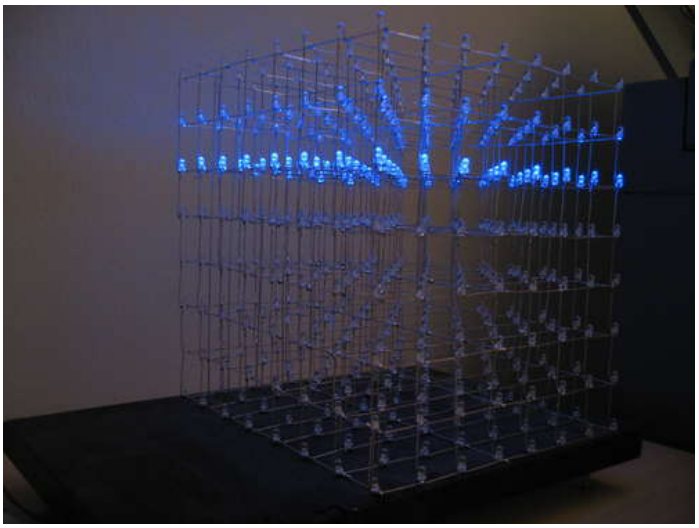
It will start by drawing a plane along one axis, then moving it along all 8 positions of that axis. (by plane we mean a flat surface, not an airplane :p) The test code will traverse a plane through all three axis.

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];

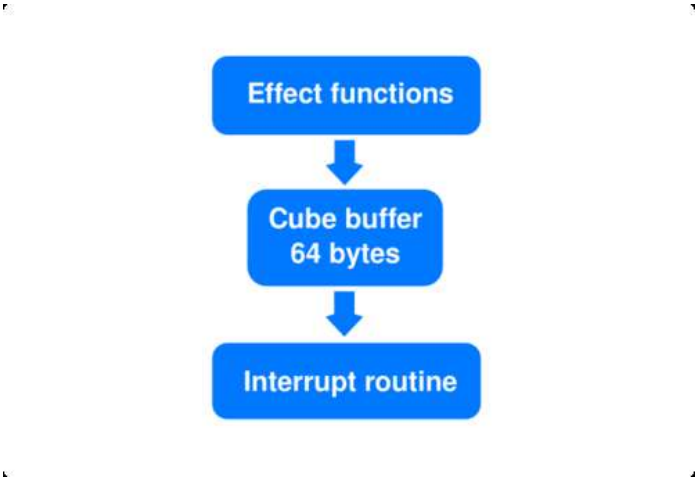
The interrupt routine reads from the cube array at given intervals and displays the information on the LED cube.

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.

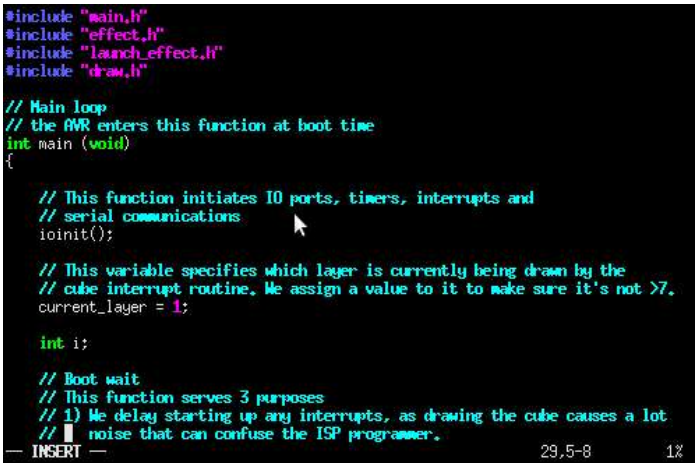
1) DDRx sets the data direction of the IO pins. (Data Direction Register). 1 means output, 0 means input.

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.



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.

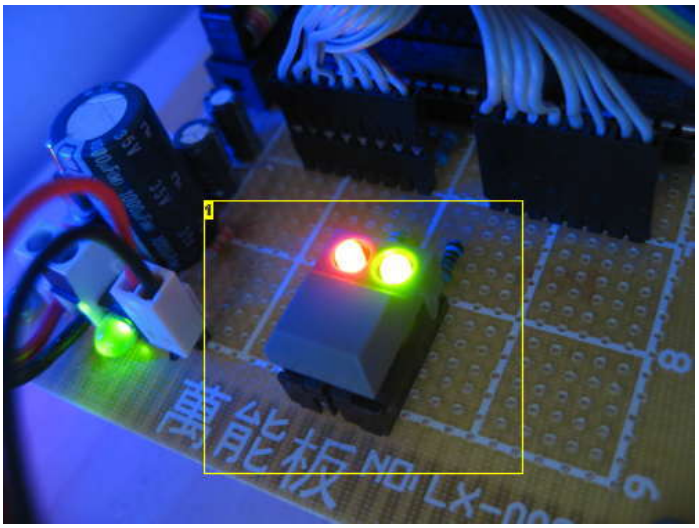


Image Notes
1. blink blink blink

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.

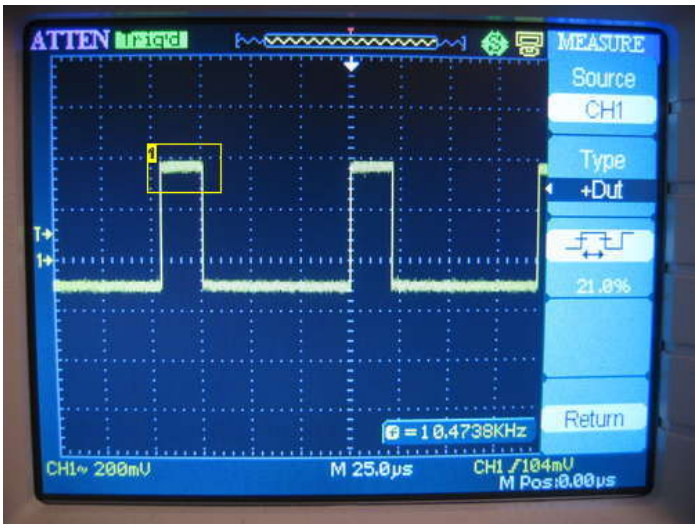


Image Notes
1. The interrupt routine pulls Output Enable high while running to disable the output of the latch array.

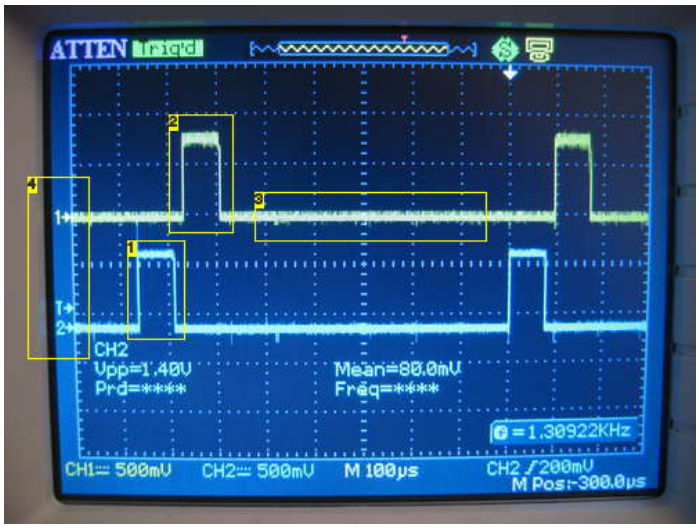


Image Notes
1. Layer 0 is on
2. Layer 1 is on
3. My oscilloscope doesn't have 8 channels, so I can only show the first two layers.
4. Output from the layer transistor lines.

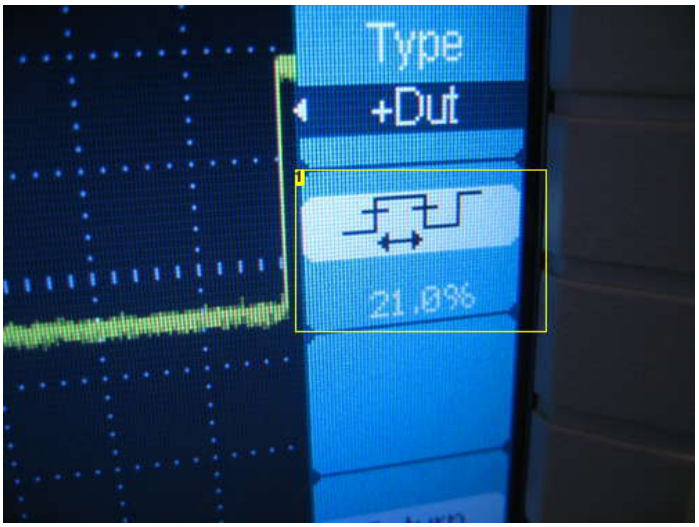


Image Notes
1. The interrupt routine runs roughly 21% of the time. This leaves the remaining 79% for effect code!

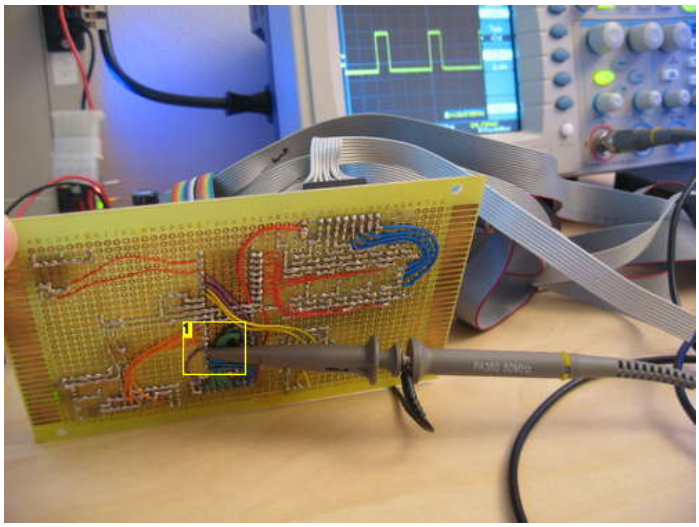
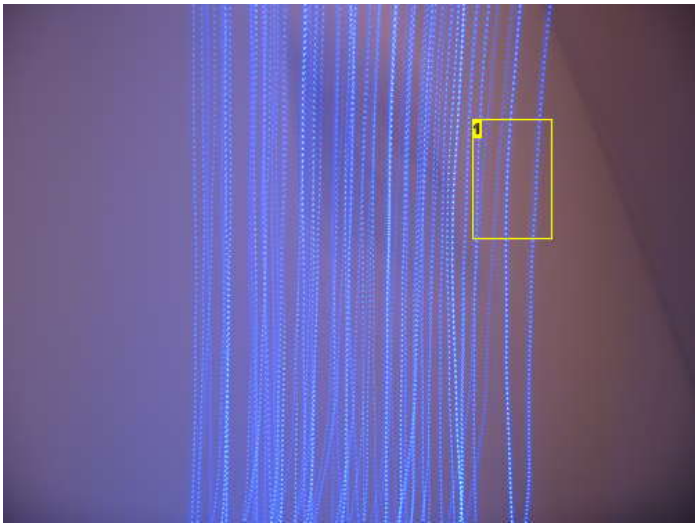


Image Notes
1. Probing output enable



```
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 ;)

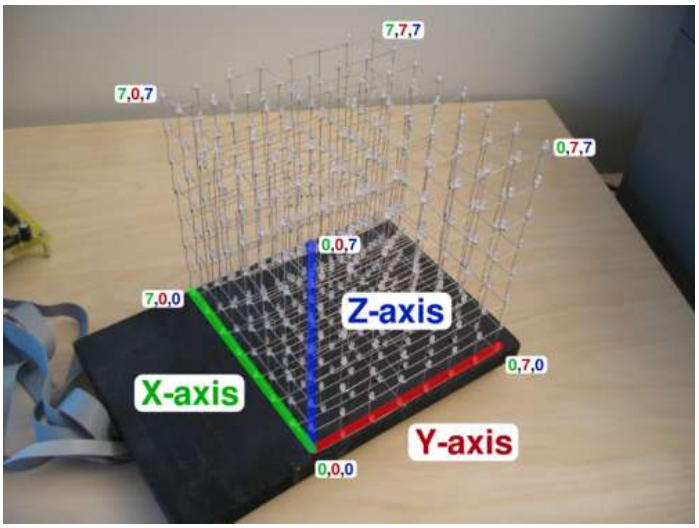
From now on, the native position of the LED cube will be with the cables coming out to the left.

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.



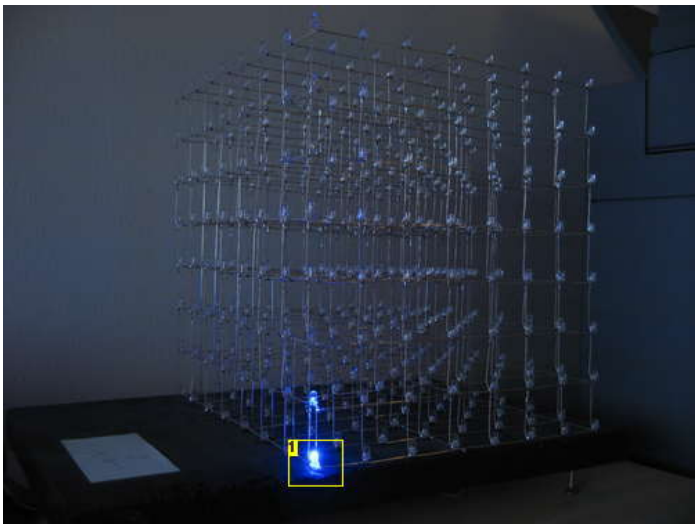


Image Notes
1. 0,0,0

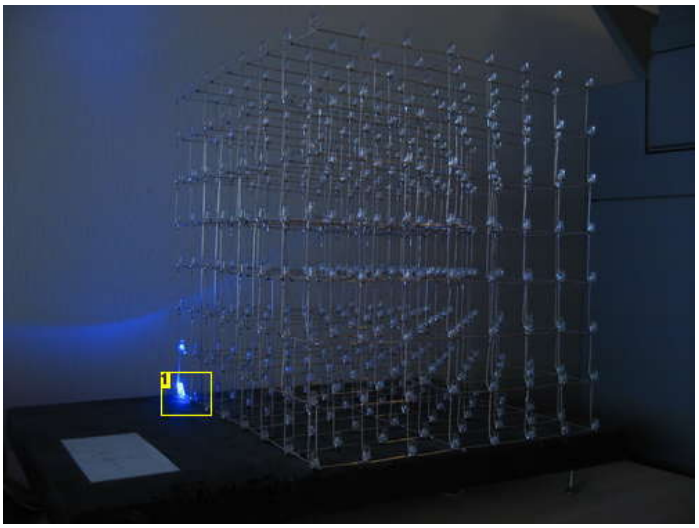


Image Notes
1. 7,0,0

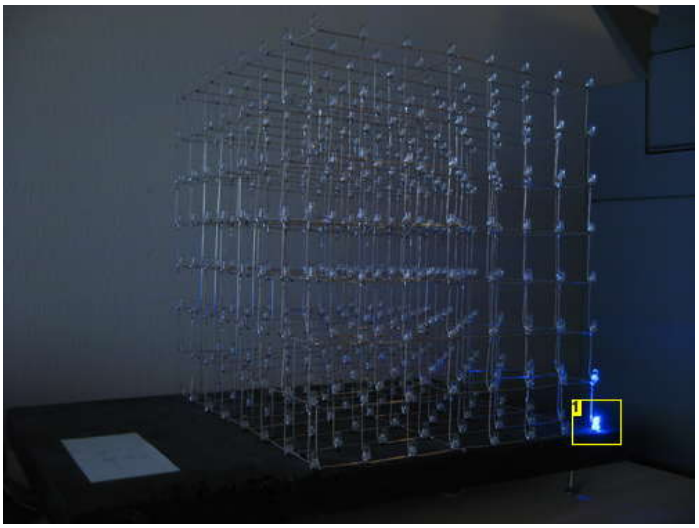


Image Notes
1. 0,7,0

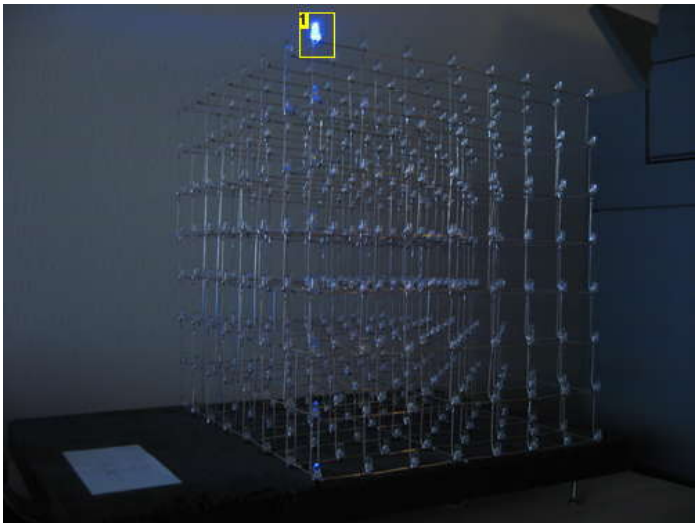


Image Notes
1. 0,0,7

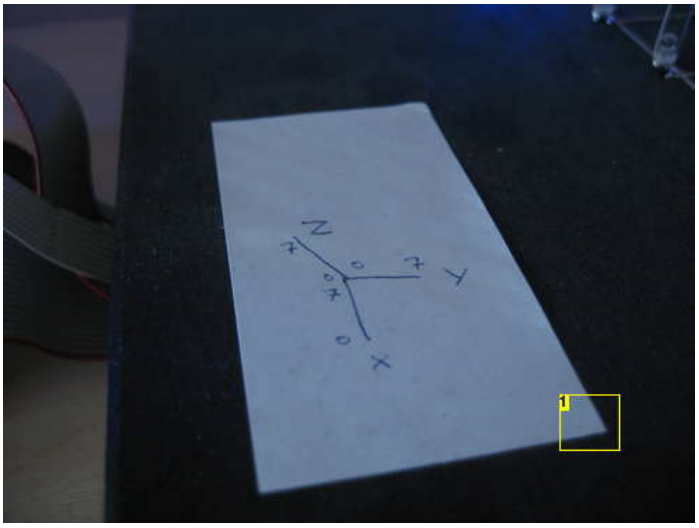


Image Notes
1. Made a little cheat note while programming ;)

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.

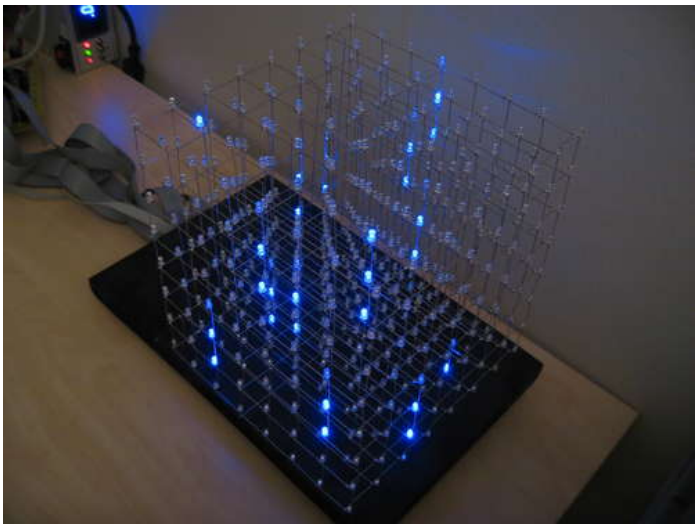
2) Loop a for() loop n number of times.

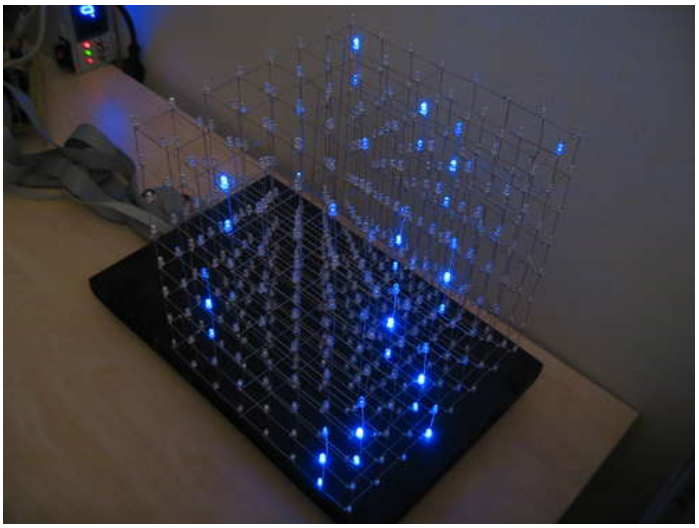
3) For each iteration of this loop, place a pixel on layer 7 (z=7) at random x and y coordinates.

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.

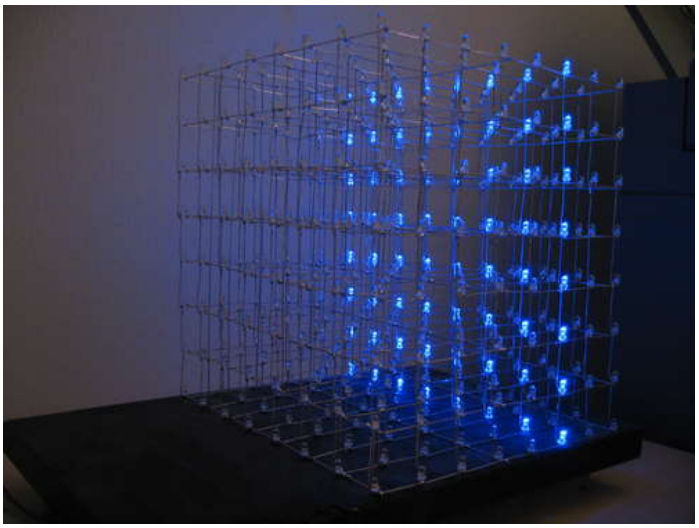
2) Clear the cube with fill(0x00);

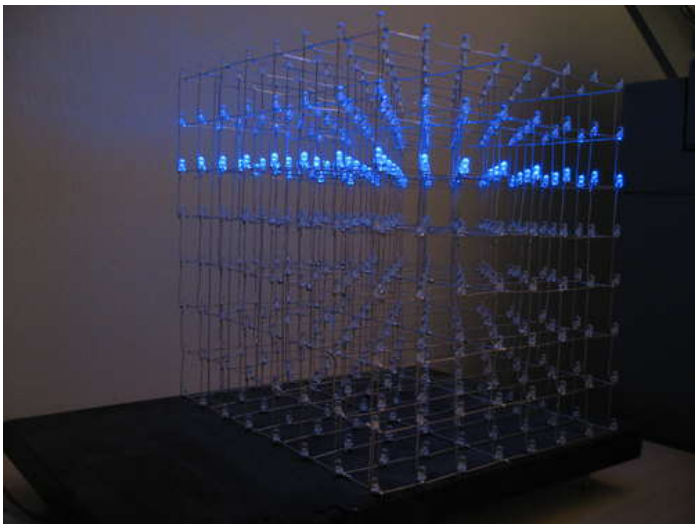
3) Call setplane() to draw a plane along the desired axis at position i. The plane isn't actually drawn on the axis specified, it is drawn on the other two axis. If you specify AXIS_Z, a plane is drawn on axis X and Y. It's just easier to think of it that way. Instead of having constants named PLANE_XY,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

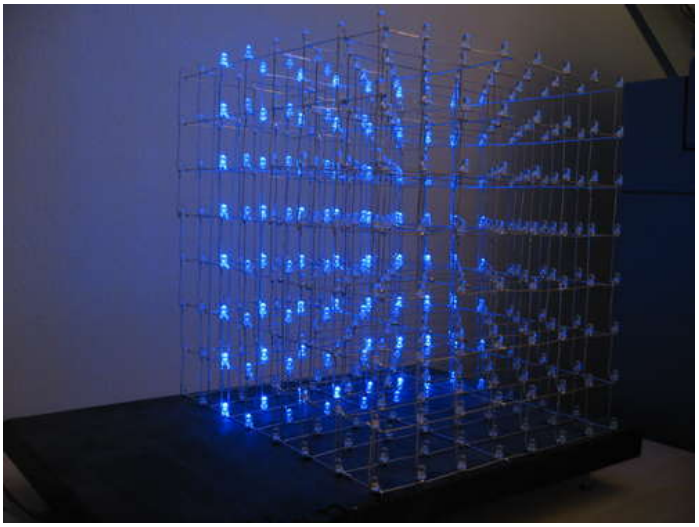
4) Delay for a while.

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:

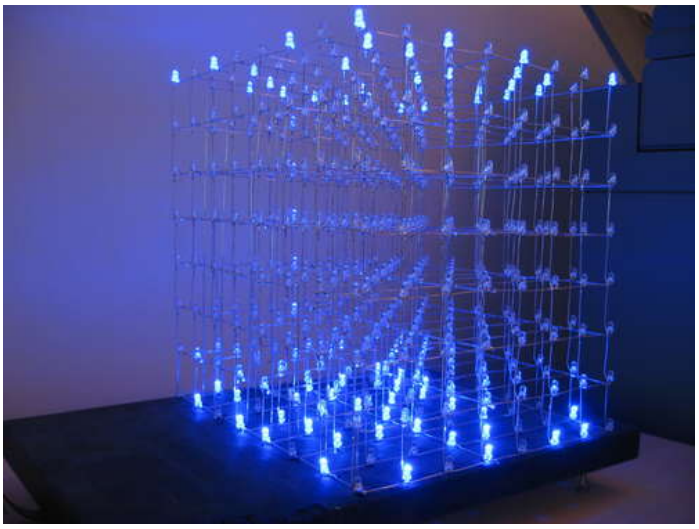
1) For()-loop i from 0 to 7.

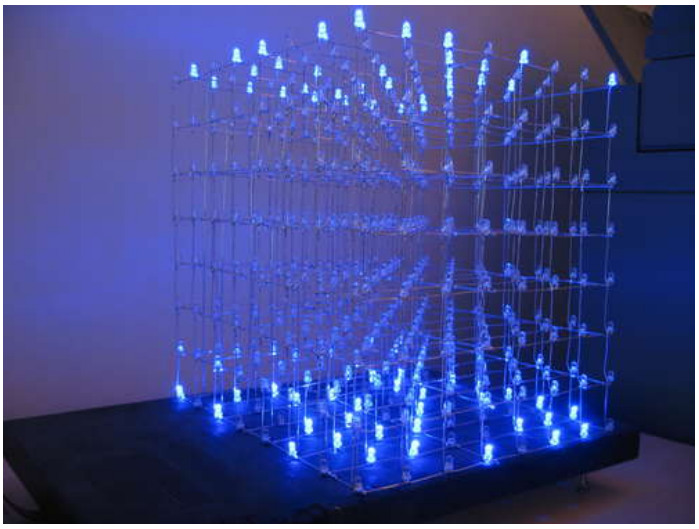
2) If the destination is 7, we set ii to 7-1, thus making ii the reverse of i. Clear the voxel at Z = ii+1. When moving down, ii+1 is the previous voxel.

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.

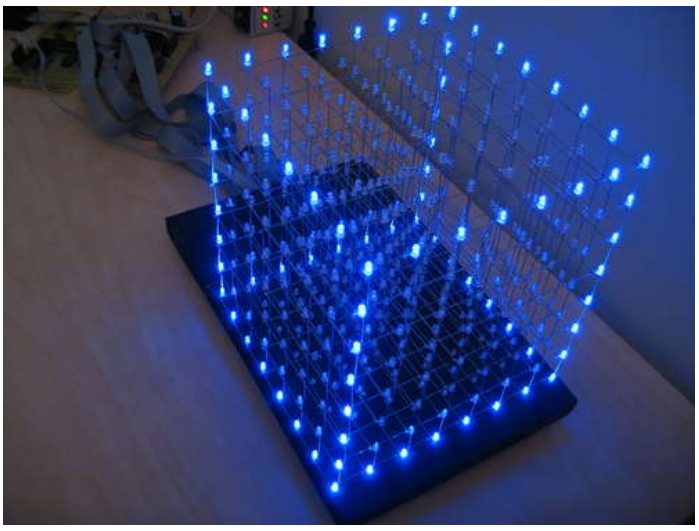
These two functions are used as one single effect in the effect launcher. First the shrink grow effect is called 8 times, one for each corner, then woopwoop is called four times, two shrink and grow cycles.

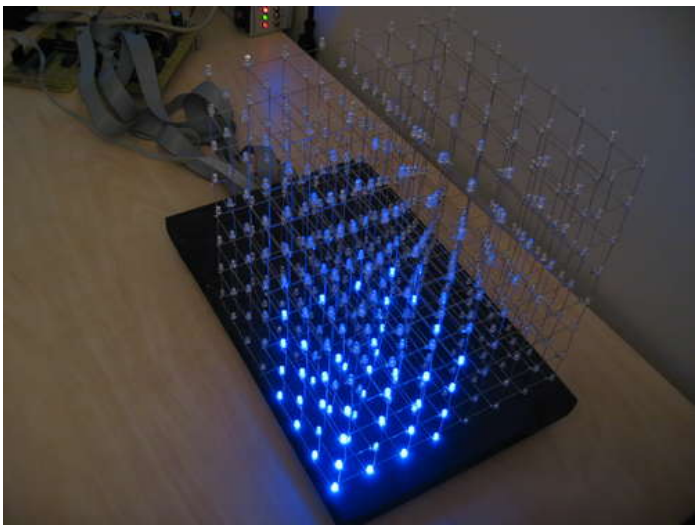
To launch the shrink grow function, we used a for loop with some neat bit manipulation tricks inside to avoid writing 8 lines of code.

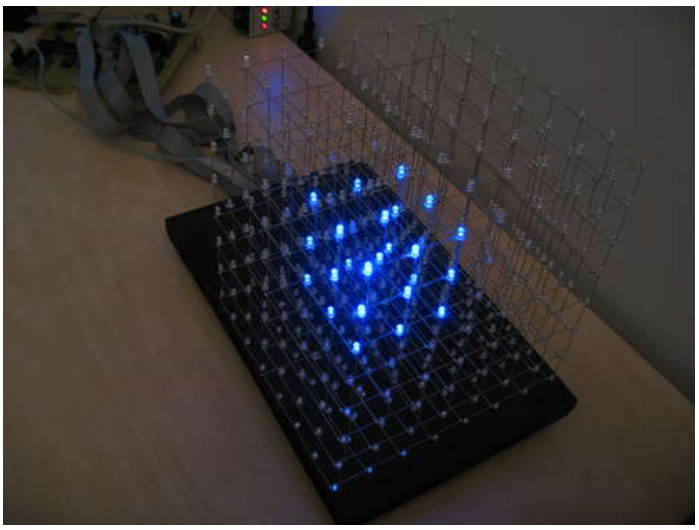
The second argument of the shrink grow functions is the rotation, in 4 steps. We are counting from 0 to 7, so we can't simply feed i into the function. We use the modulo operator % to keep the number inside a range of 0-4. The modulo operator divides by the number specifies and returns the remainder.

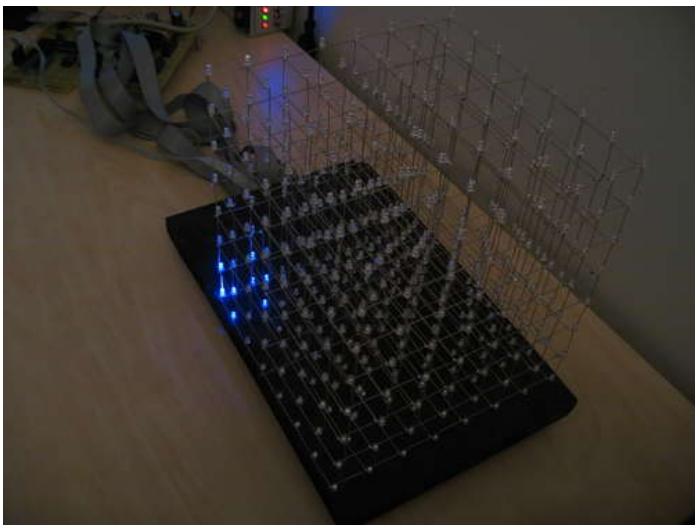
The third argument is the flip. When flip = 0, the cube is not flipped. > 0 flips. We use the bitwise AND operator to only read bit 3 of i.

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

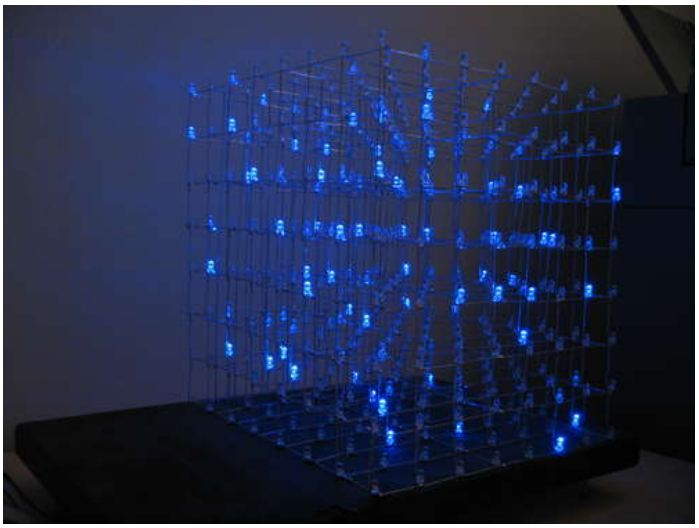
This is one of our favorite effects. The voxels randomly suspended in the cube gives a nice 3d depth, especially if you move your head while viewing the effect.

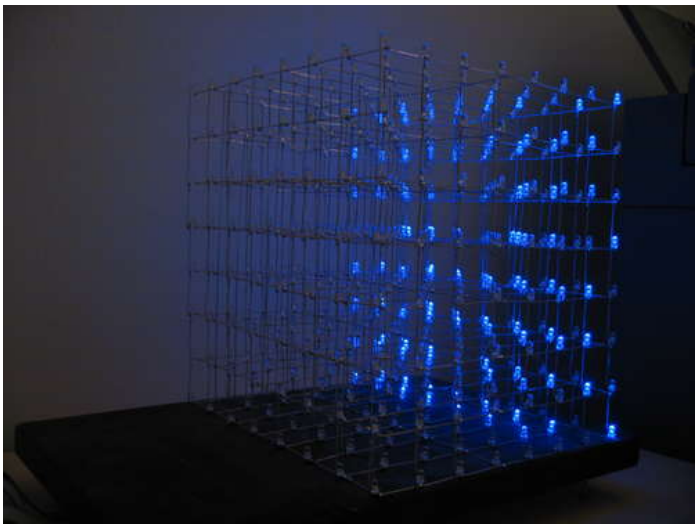
64 voxels start out on one of the side walls. Then they all get assigned a random midway destination between the side wall they started at and the wall on the opposite side.

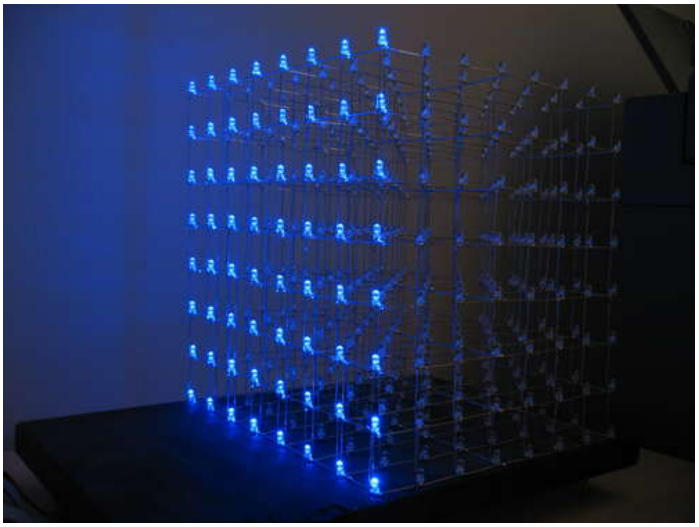
The function then loops 8 times moving each voxel closer to its midway destination. After 8 iterations, the voxels are suspended at different distances from where they started. The function then pauses for a while, thus the name axis_updown_randsuspend ;). It then loops 8 times again moving the voxels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

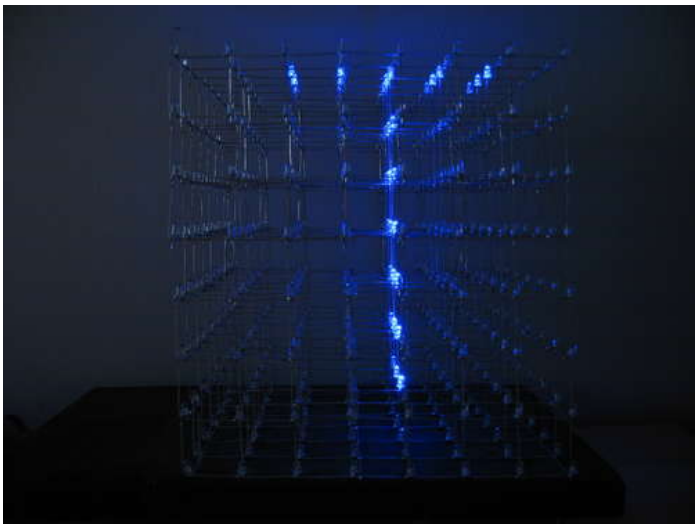
Step 62: Software: Effect 6, stringfly

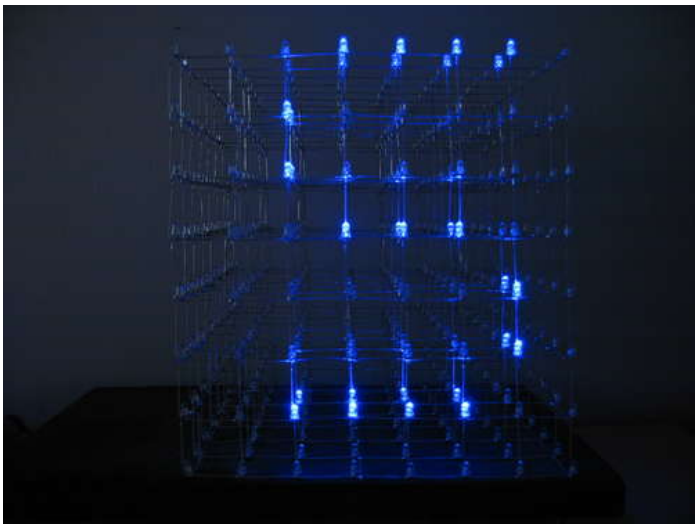
8x8 is about the smallest size required to render a meaningful text font, so we just had to do just that!

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 = {
    0x00,0x00,0x00,0x00,0x00,0x5f,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;

        // Uncomment 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:

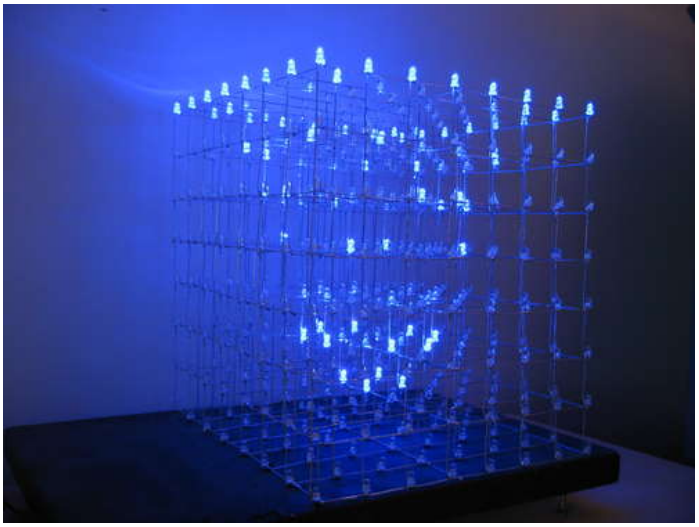
1) Loop through the iteration counter.

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!

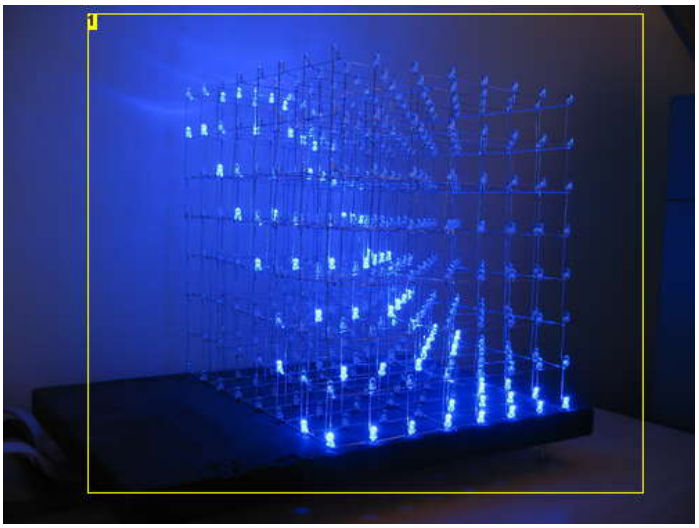


Image Notes
1. Beautiful math!

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.

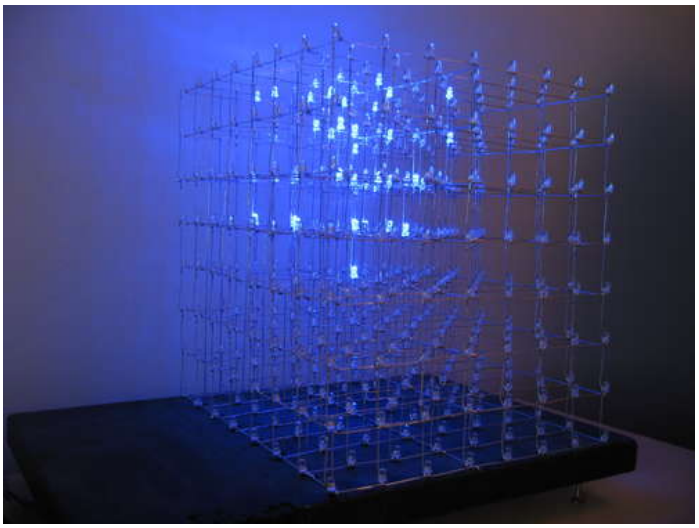
7) For each particle, the x y and z coordinates are incremented by their dx, dy and dz velocities divided by the slowrate. This will make the particles move slower and slower.

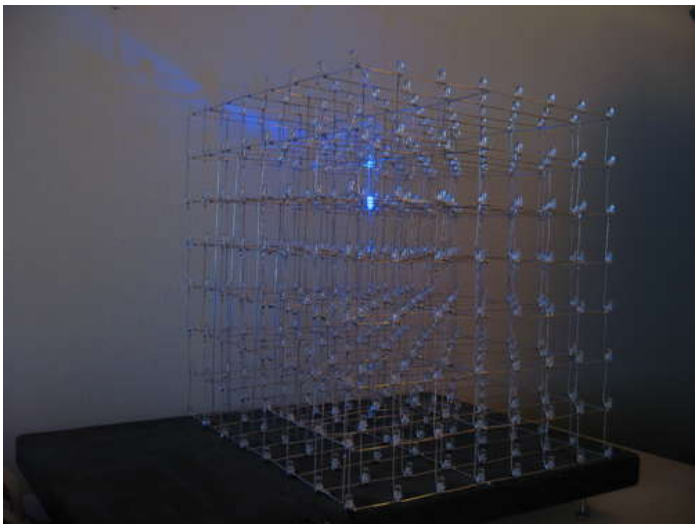
8) The z coordinate is decreased by the gravity variable.

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

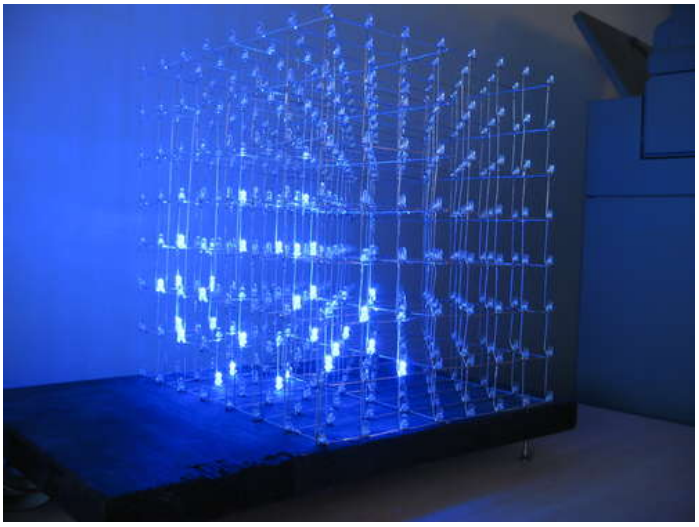
The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway. You can read more about this on Wikipedia , if you haven't heard about it before.

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.

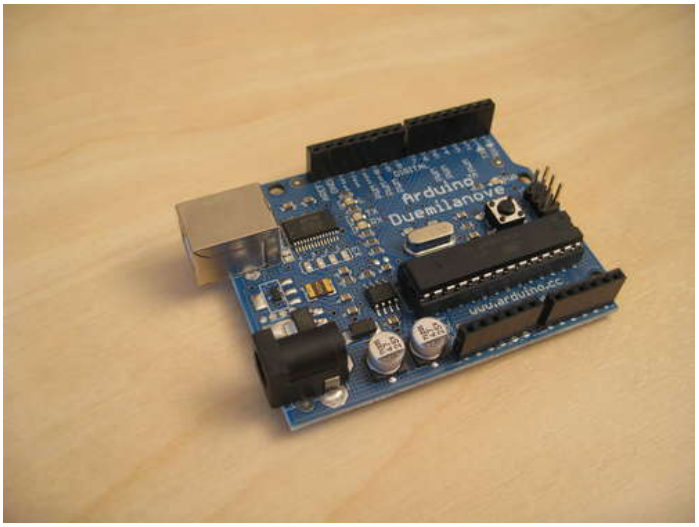
Since the multiplexer array and AVR board are separated by a ribbon cable, connecting the IO lines to an Arduino is a simple matter of connecting some breadboard wires. Luckily, we soldered in a female 0.1" pin header for the transistor lines when we were debugging the first set of transistors. Just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



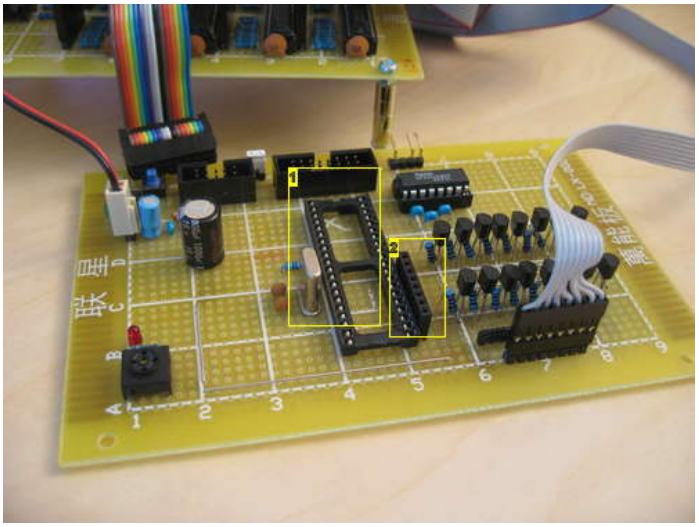
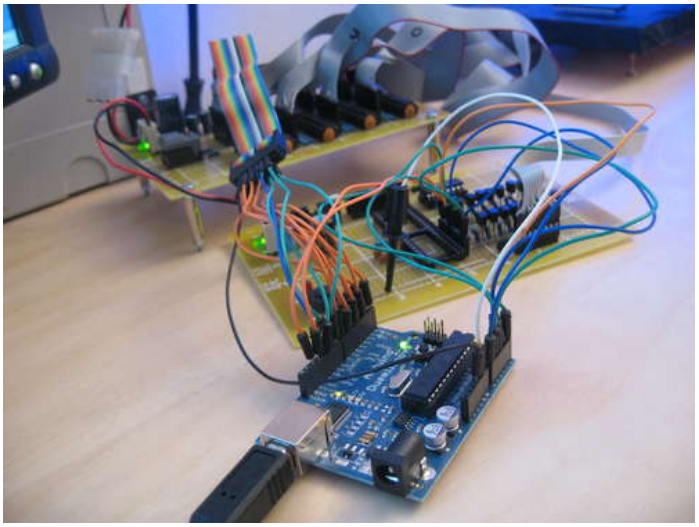
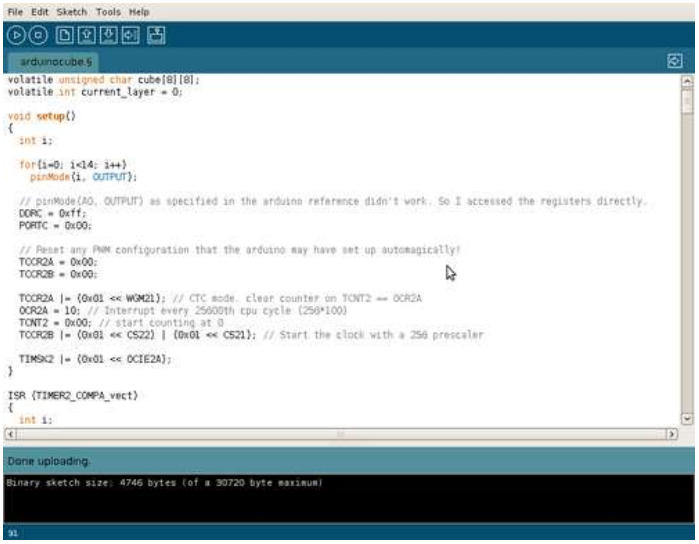


Image Notes
1. ATmega temporarily removed
2. Layer select lines can be connected to this header, without the ATmega interfering.





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.

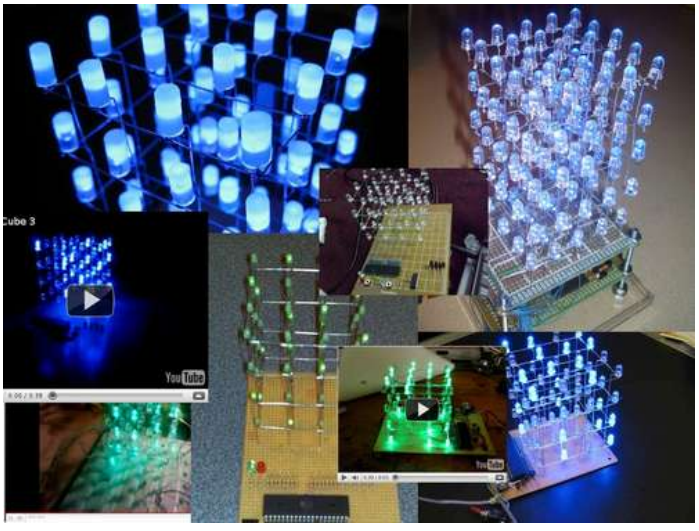
Step 72: Feedback

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

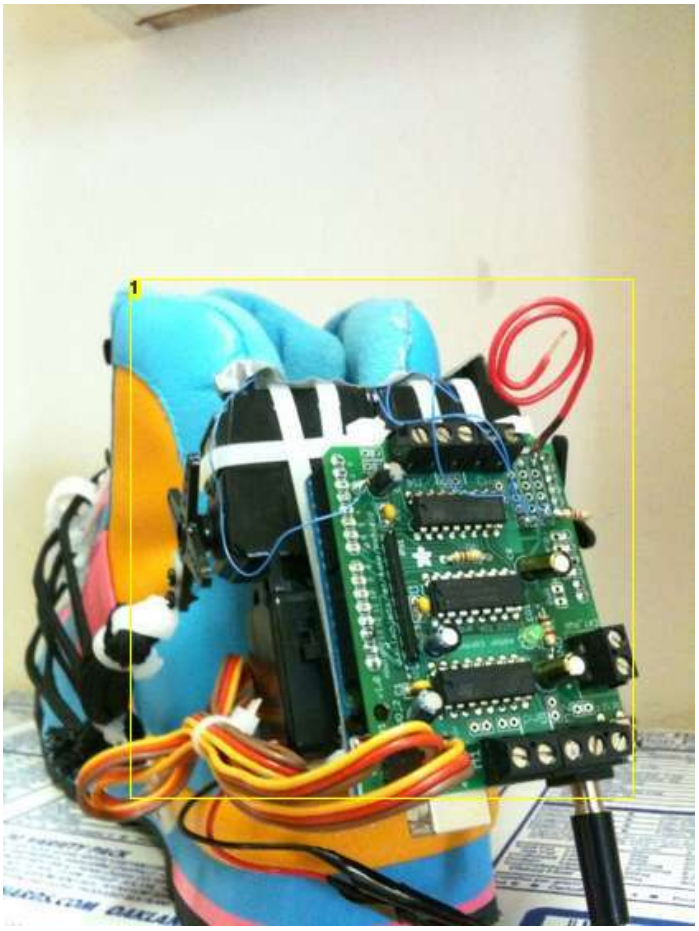


Image Notes
1. Just wait until everyone at the Airport sees your new kicks!

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

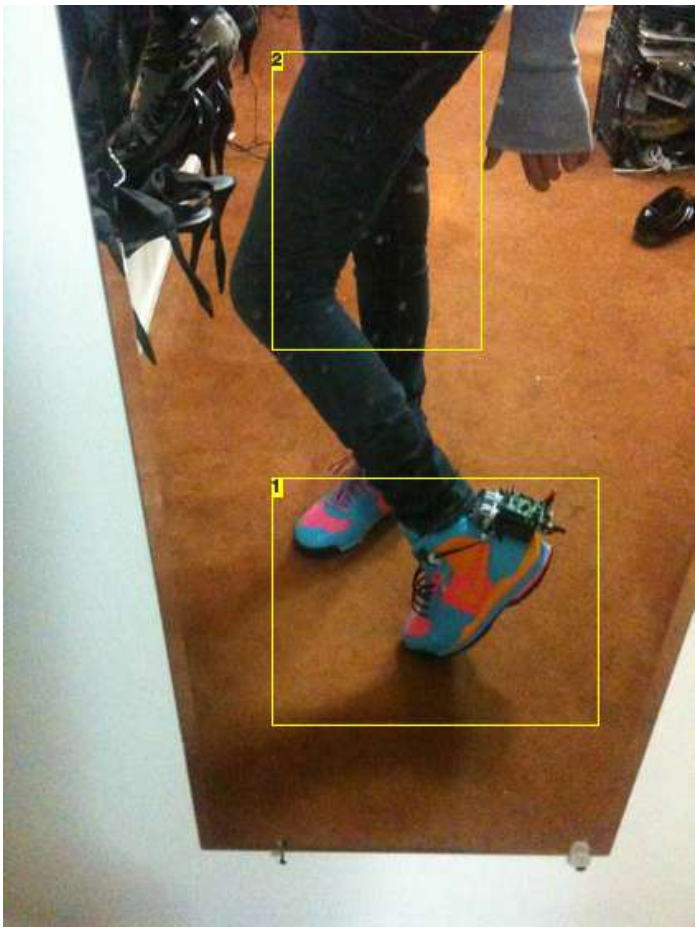


Image Notes
1. Surprisingly wearable!
2. I should get around to cleaning this mirror, eventually.

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.

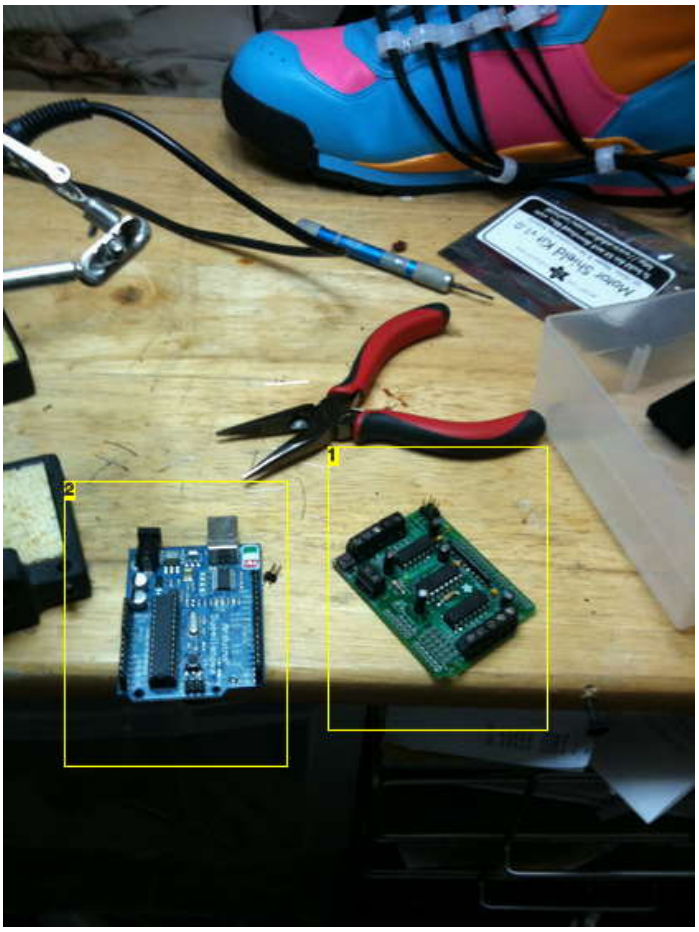


Image Notes
1. Completed Motor Shield
2. Arduino

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.

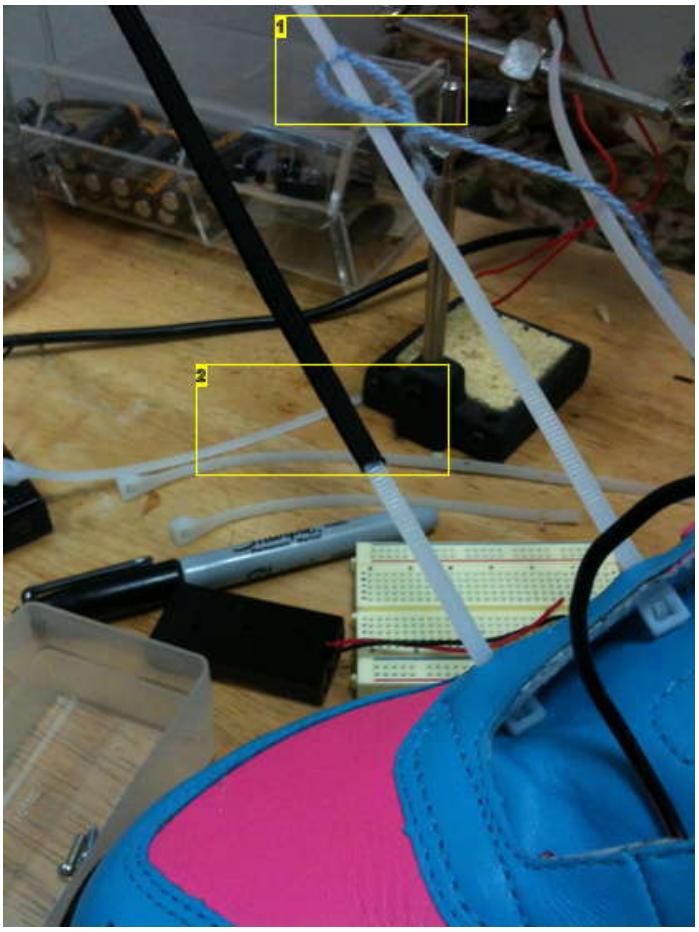


Image Notes
1. Ignore this- was just for protyping
2. Fit the shell onto the Zip tie.

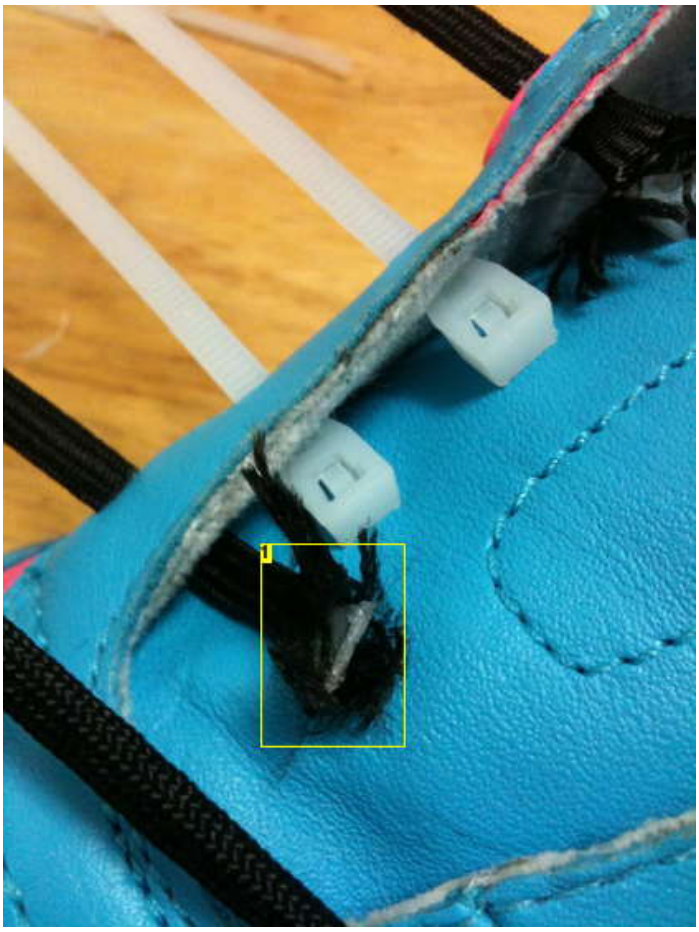


Image Notes
1. Trim right at the head, leaving a bit of a wedge.

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!

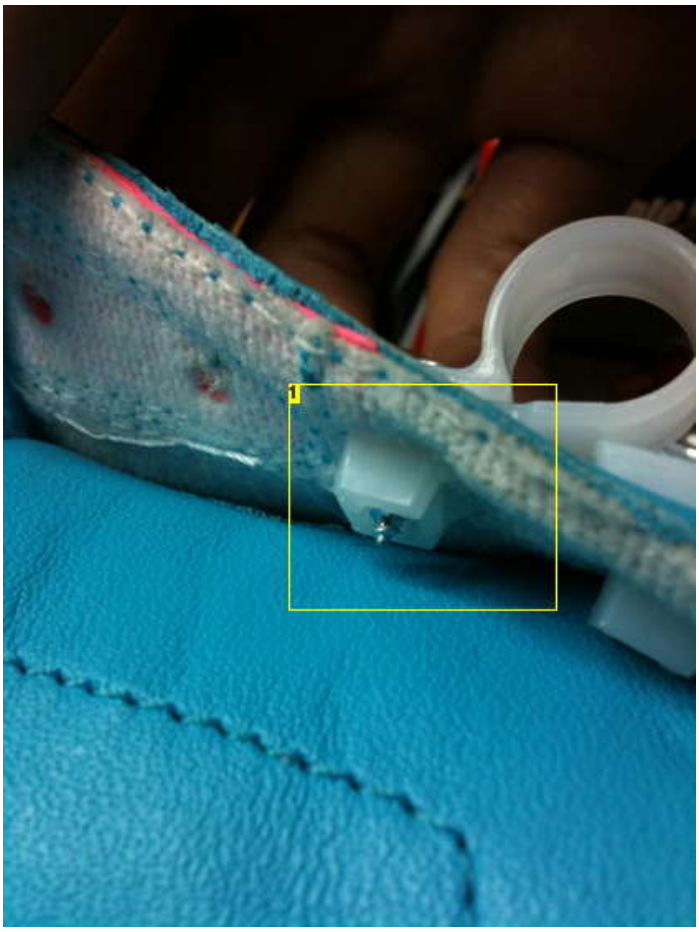


Image Notes
1. I ran out of screw nuts, so I reused the "heads" from the zip ties. I put a little hot glue on the screw tips afterward to prevent it from scratching anything.

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.

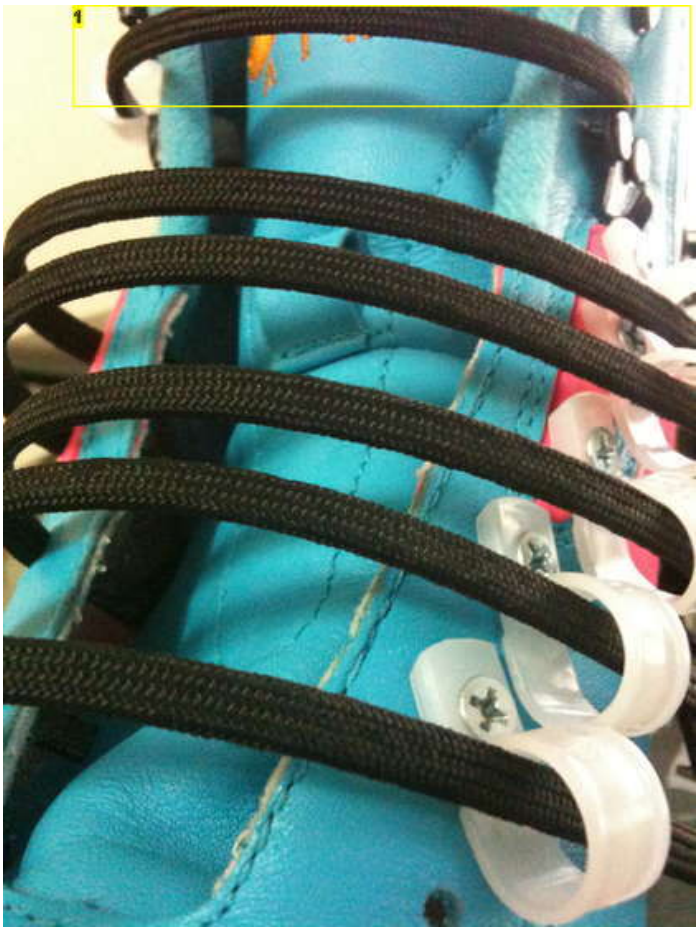


Image Notes
1. Ankle strap, added later in the project.

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.

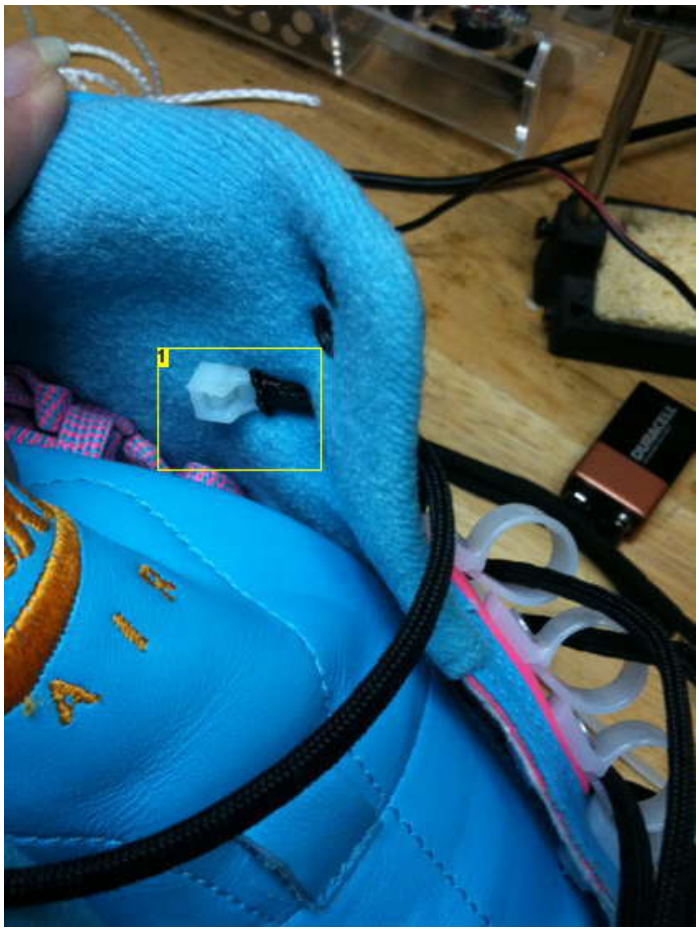


Image Notes
1. I still melted the nylon down a bit, but kept the head on this one attached. I then screwed it into place to secure it.

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.

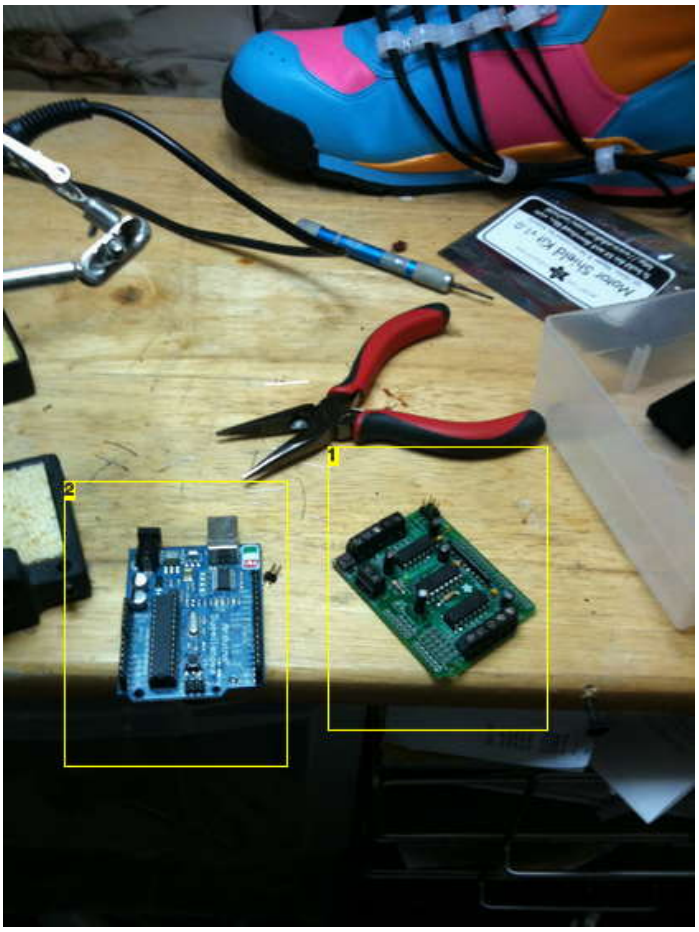


Image Notes
1. Completed Motor Shield
2. Arduino

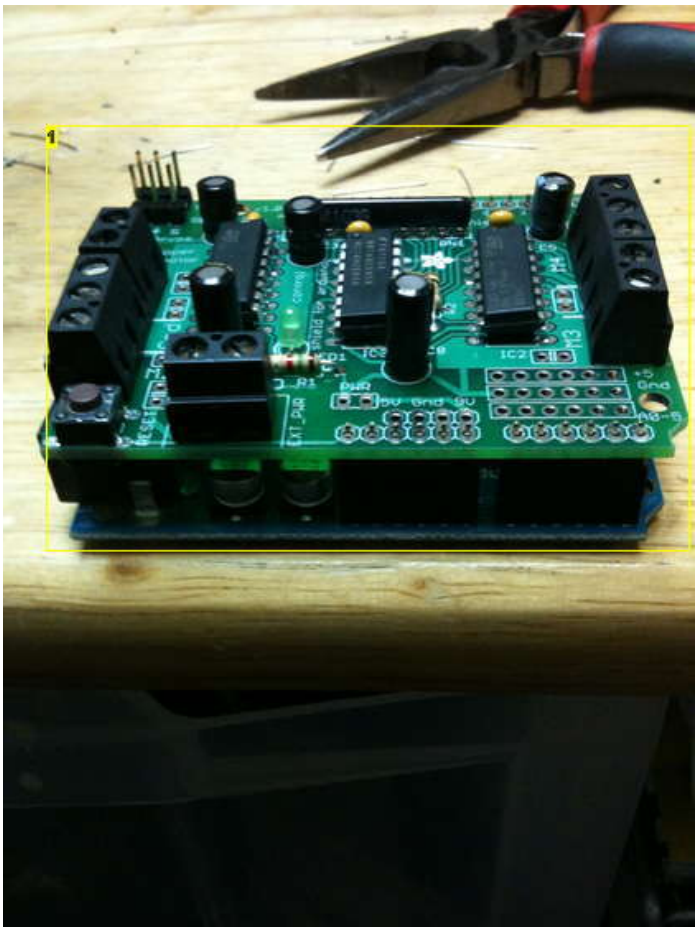


Image Notes
1. Motor shield mounted into place.

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.

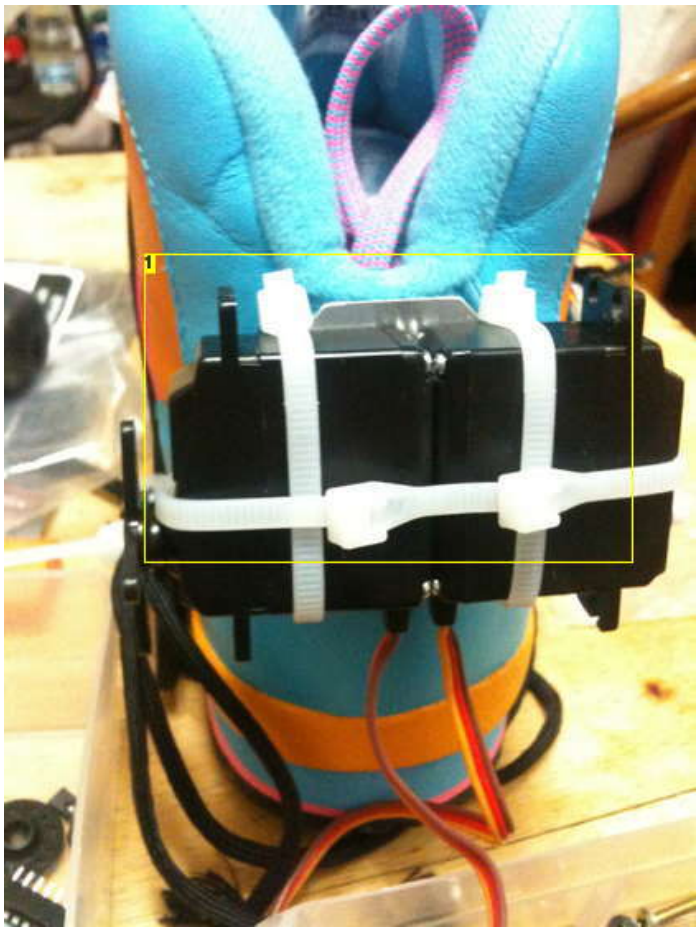


Image Notes
1. Clipped ties

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.

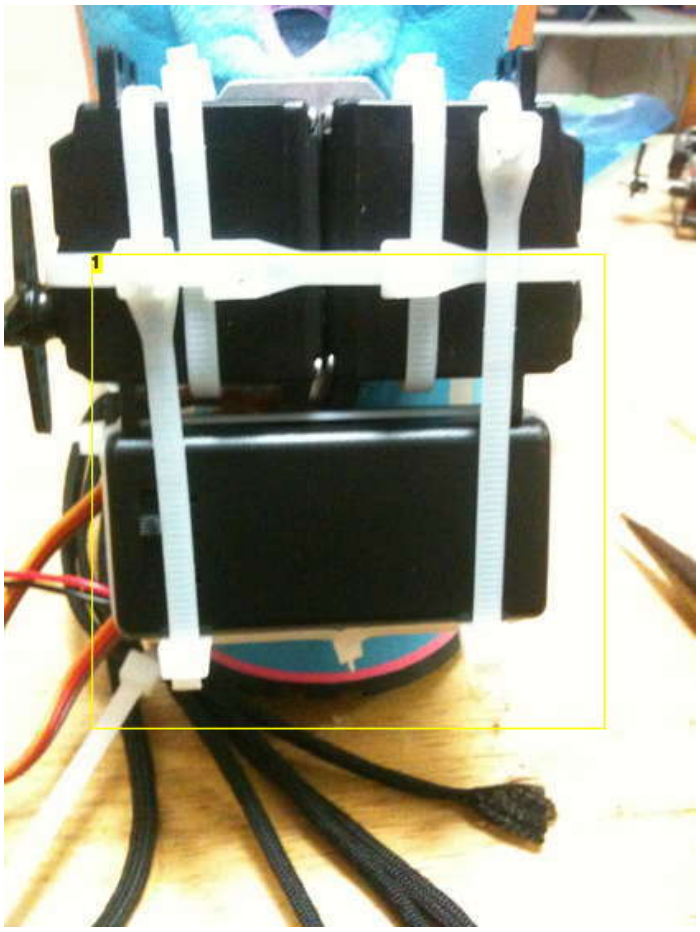


Image Notes
1. Just enough space here to mount the Arduino.

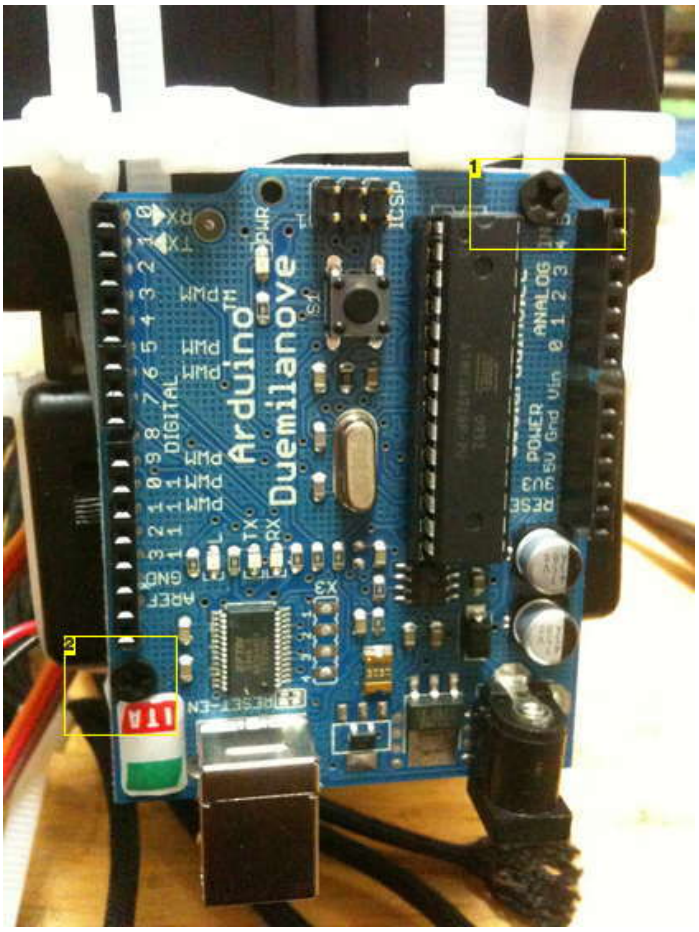


Image Notes
1. One screw
2. Two screw

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!

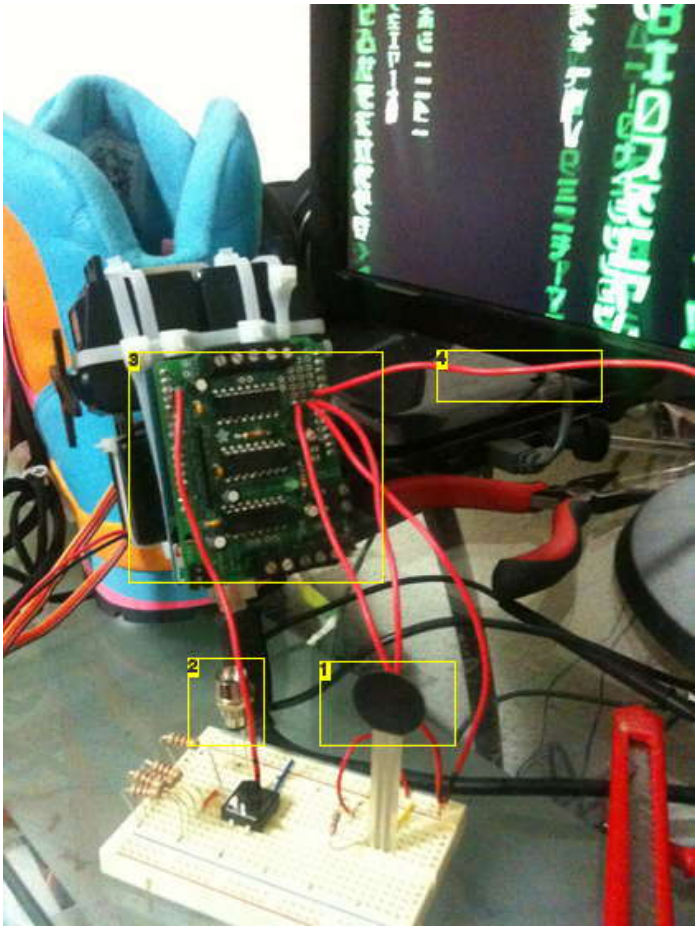


Image Notes
1. Force Sensor
2. LED (later removed the the metal shell)
3. Science!
4. Later wound up and turned into touch sensor

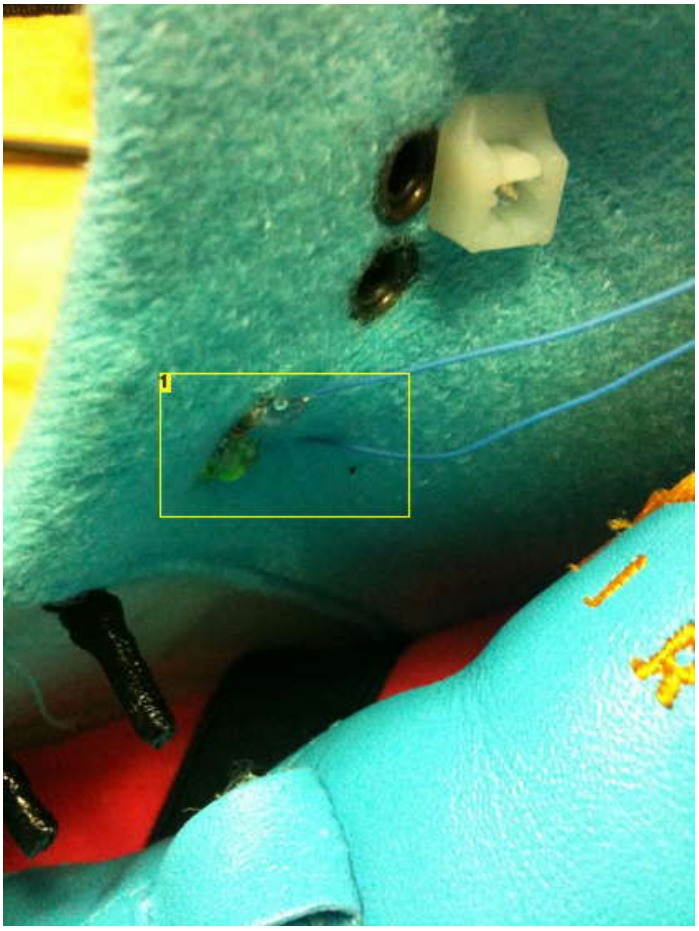


Image Notes
1. Added a little hot glue to protect the electronics.

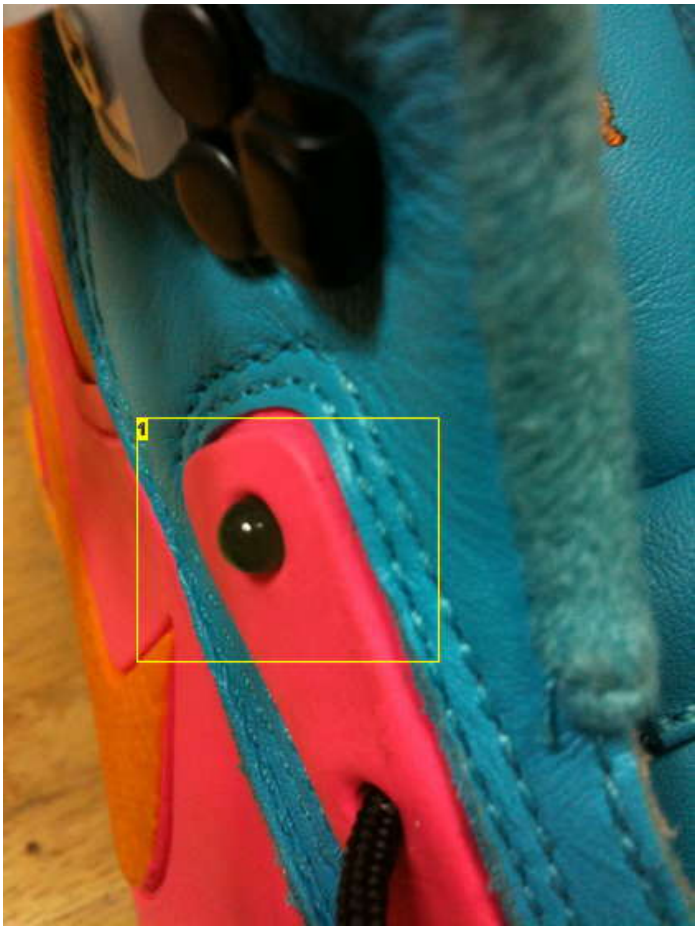


Image Notes
1. Lookin' subtle- nice contract to the rest of this contraption :)

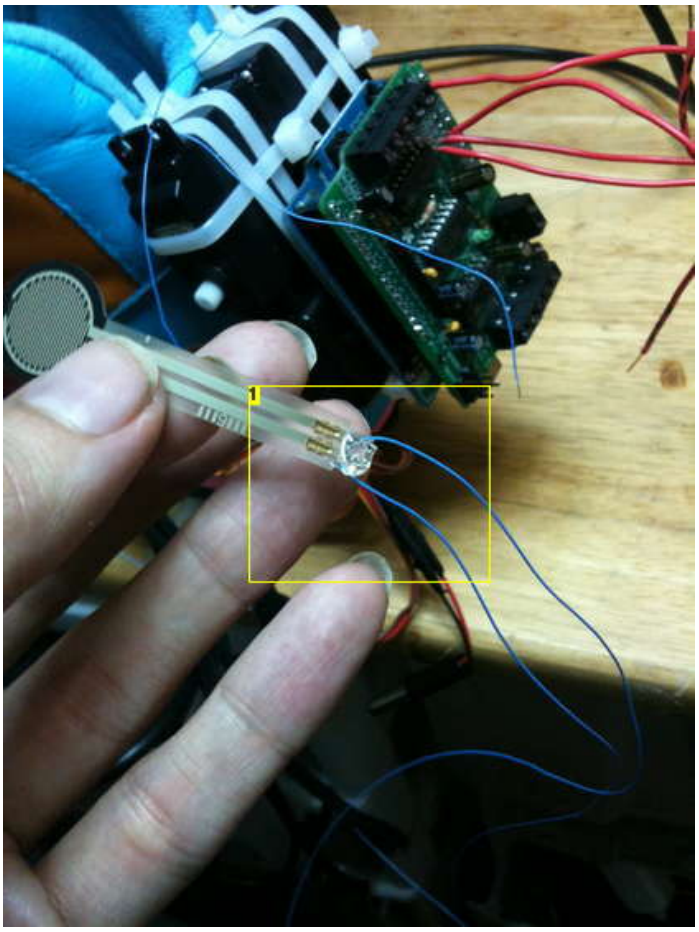


Image Notes
1. Force sensor with attached leads.

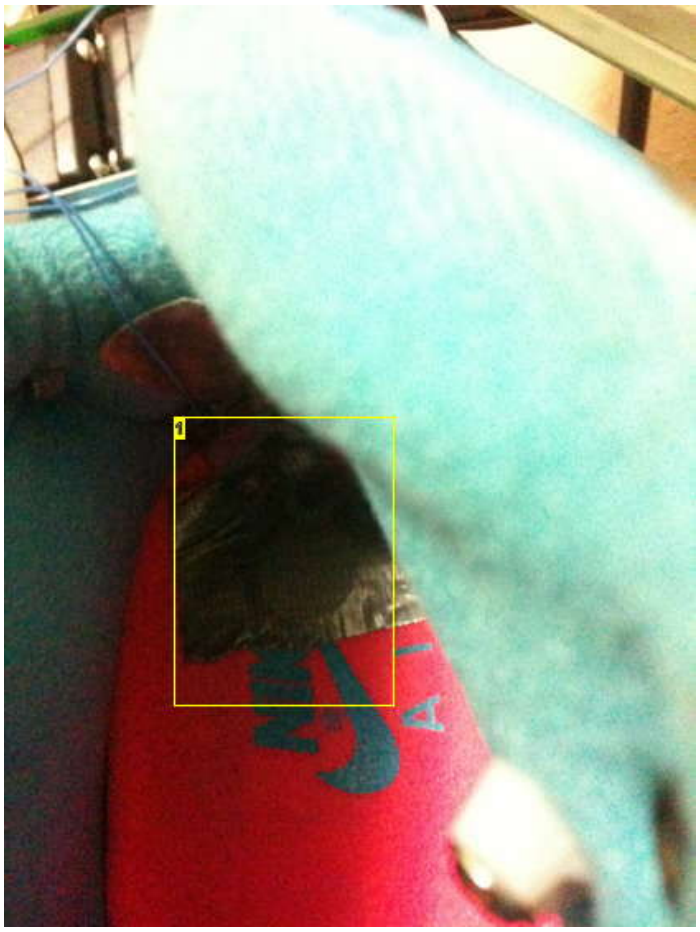


Image Notes
1. Duct Tape! (Covering force sensor)

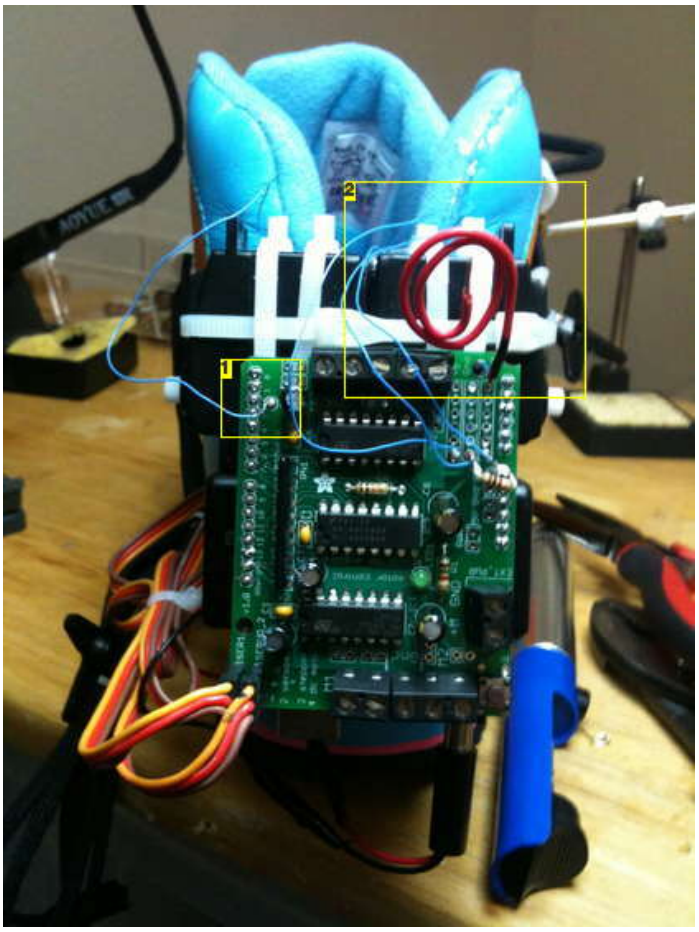


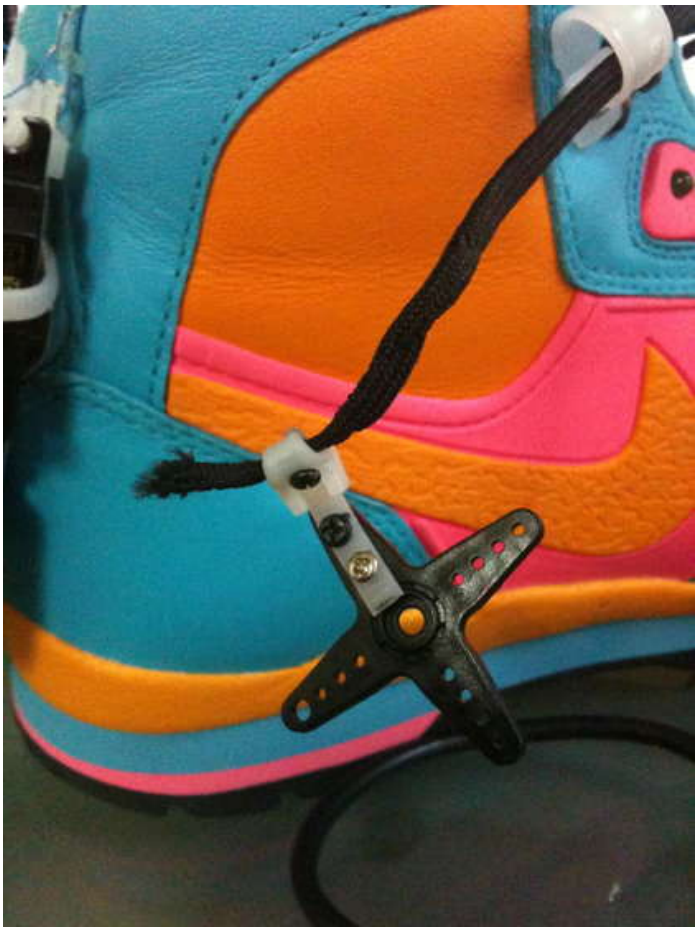
Image Notes
1. Positive LED wire to digital pin 2
2. I didn't have any working push-button switches, but I found that just touching the wire was enough to change the flow of current and could act as a switch by itself.

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 CO2 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 6CO2 +6H2O --> sunlight--> C6H12O6 +6O2. 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.

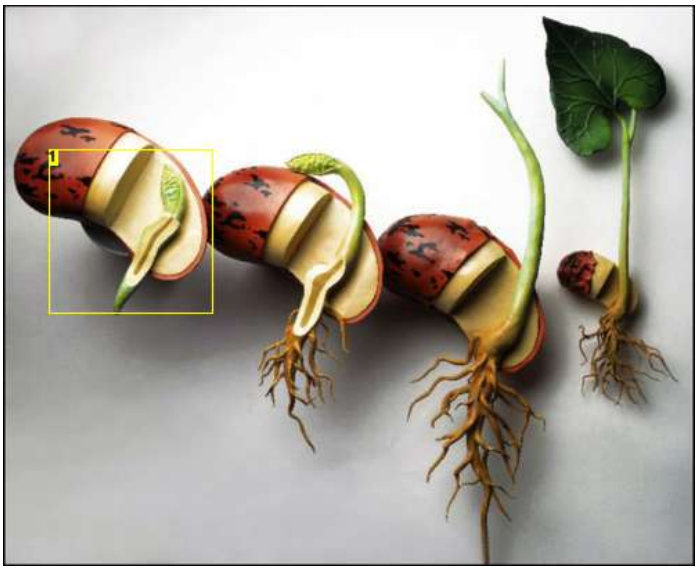


Image Notes
1. first thing to develop is the roots

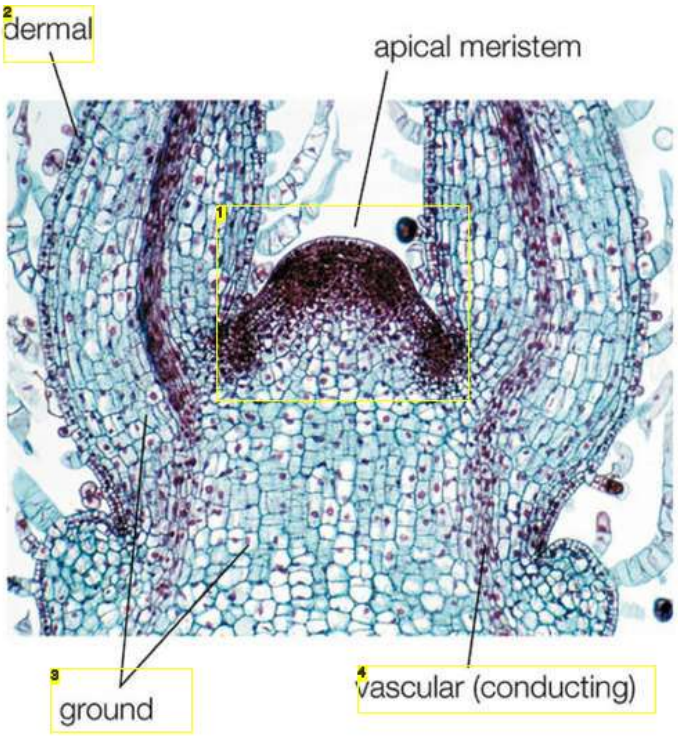


Image Notes
1. Where the growth happens
2. or epidermal tissue
3. where photosynthesis takes place
4. Where the water and nutrients are stored and moved around throughout the plant

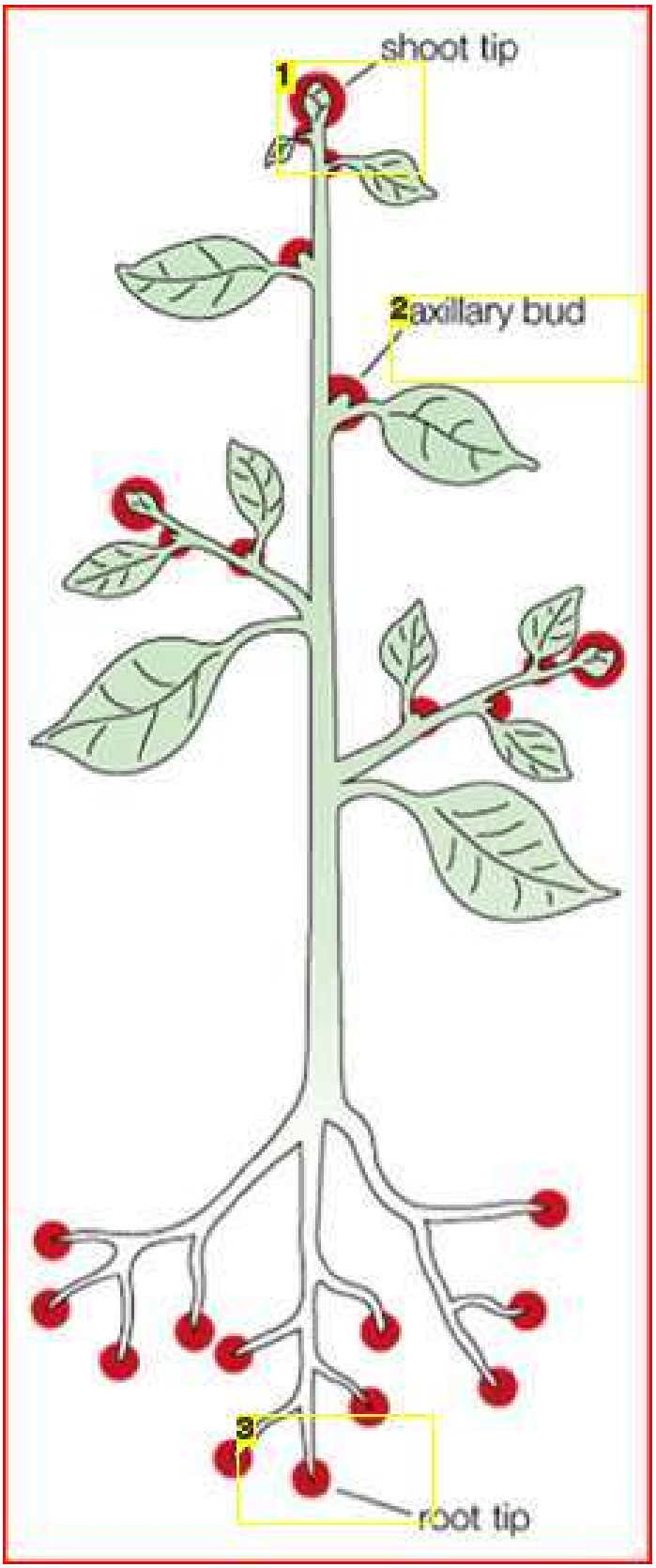
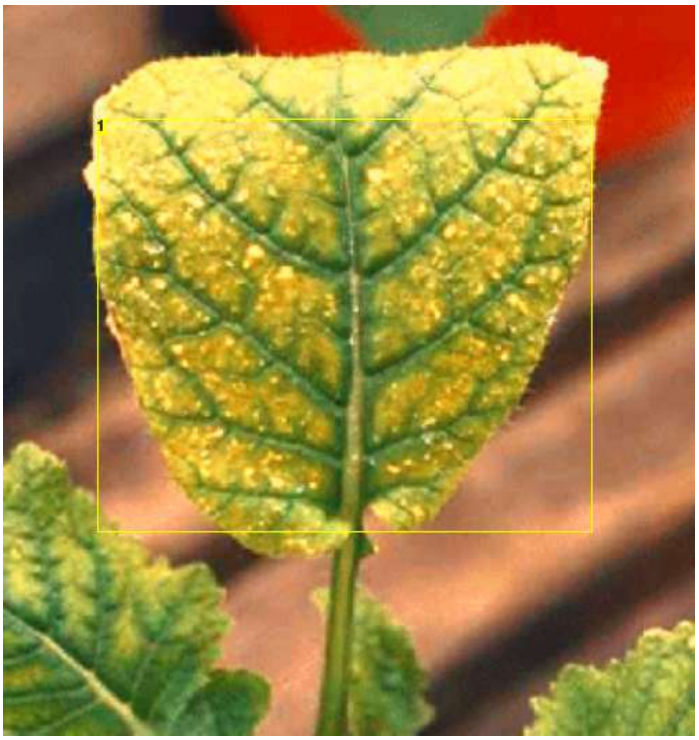


Image Notes
1. Apical Bud: Main longitudal growth
2. main latitudinal growth
3. Growth in the roots



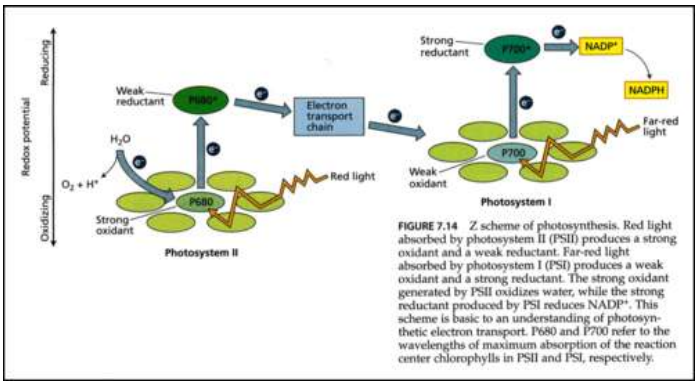
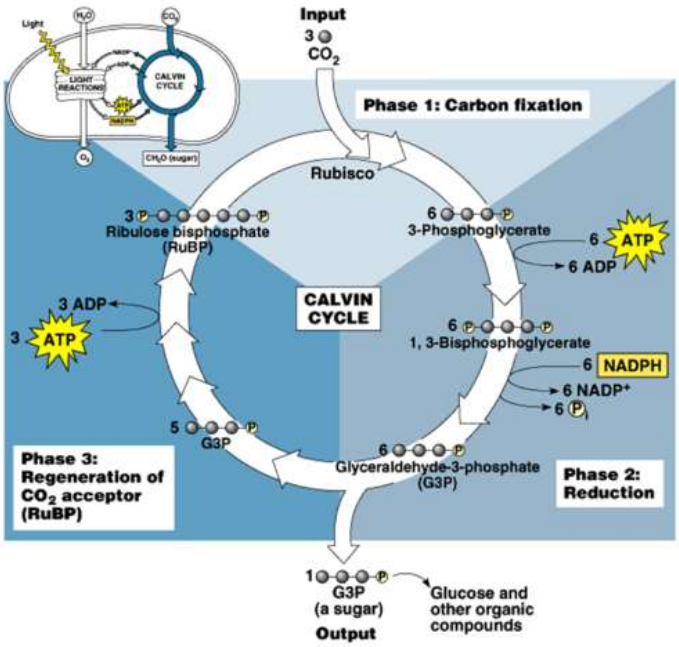


FIGURE 7.14 Z scheme of photosynthesis. Red light absorbed by photosystem II (PSII) produces a strong oxidant and a weak reductant. Far-red light absorbed by photosystem I (PSI) produces a weak oxidant and a strong reductant. The strong oxidant generated by PSII oxidizes water, while the strong reductant 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

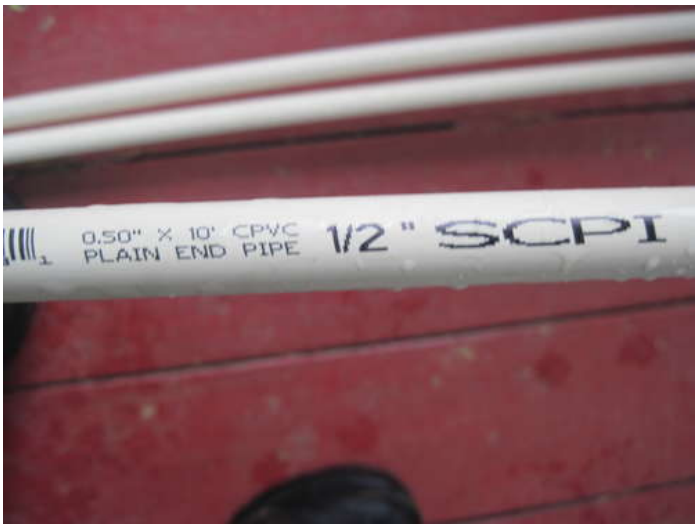
Tools:

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.

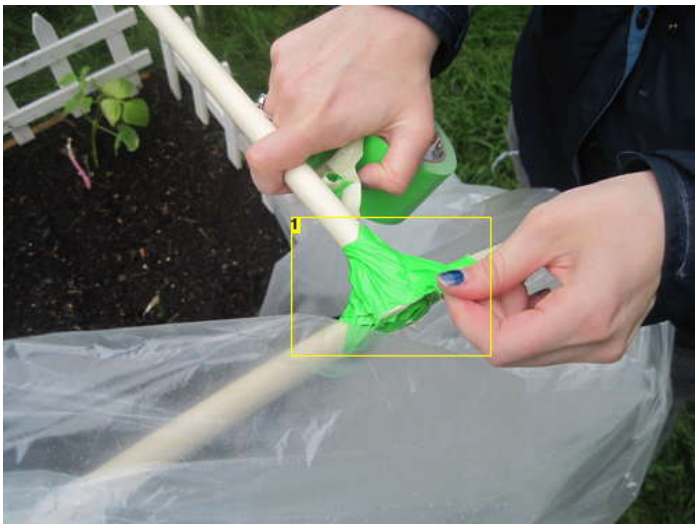


Image Notes
1. duct tape the joints of the PVC bar

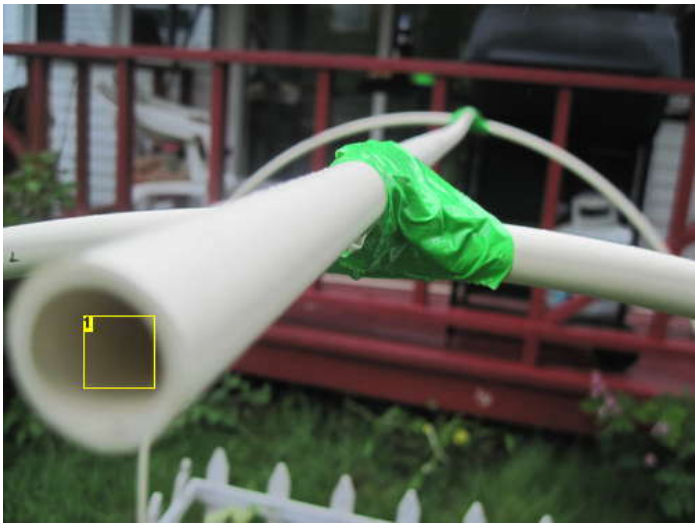


Image Notes
1. This overhang is not essential. It was just a cosmetic choice.

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

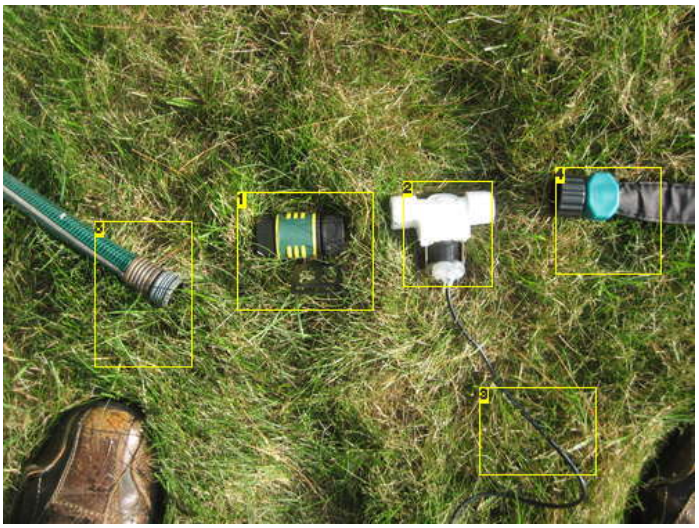


Image Notes
1. Hose connector
2. Valve- you may need to add plumbers tape to the threads so that the nozzles fit on as tight as possible.
3. cord to the Wall Wart
4. Sprinkler Connection
5. Hose connected to water source

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.

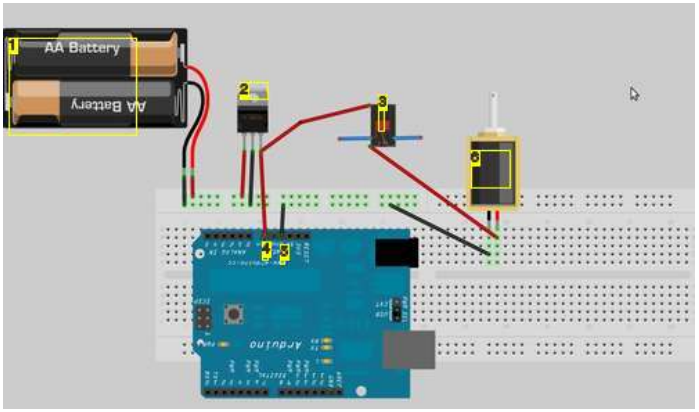


Image Notes
1. Battery over 12 volts
2. 12 Volt Regulator
3. relay
4. V in
5. Ground
6. Solenoid valve

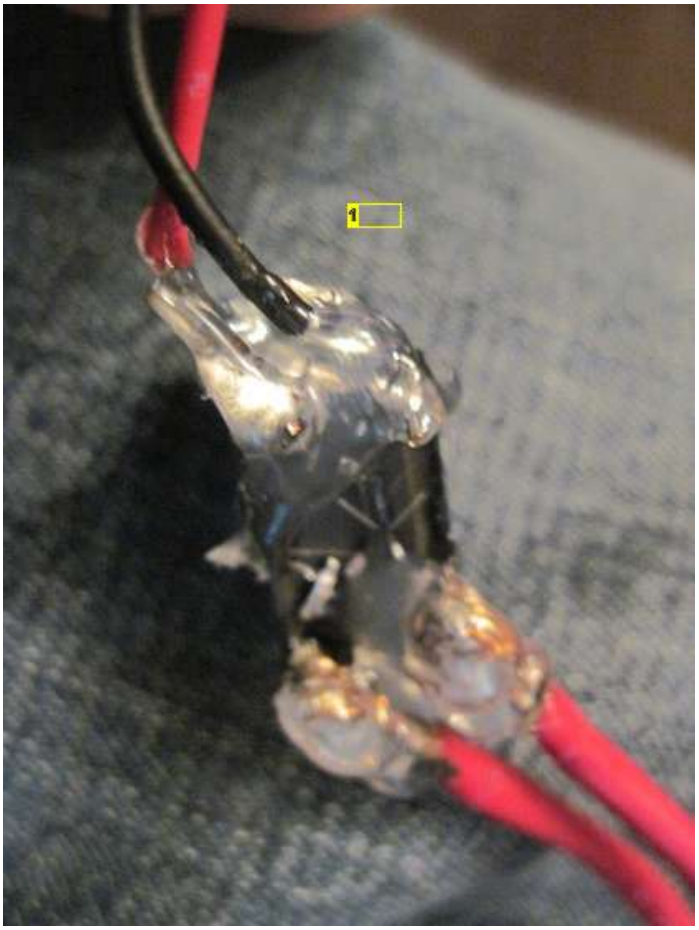


Image Notes
1. Hot glue every exposed connection. SERIOUSLY!

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.

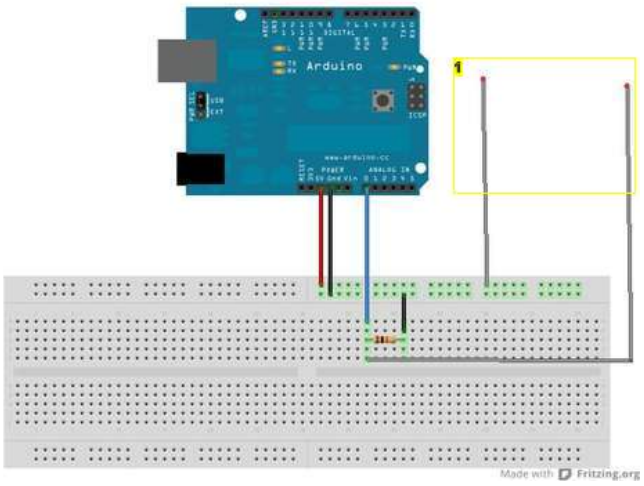
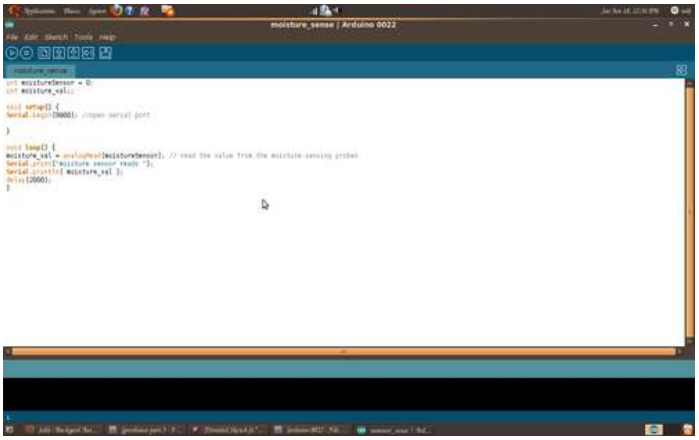


Image Notes
1. these are the nails

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.



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.

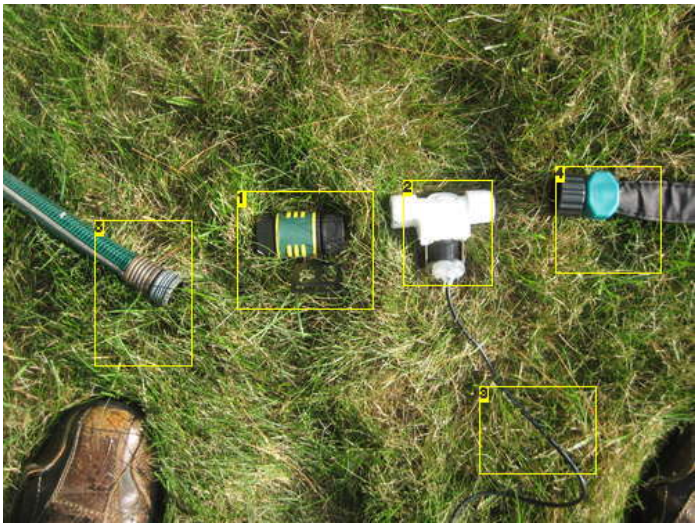


Image Notes
1. Hose connector
2. Valve- you may need to add plumbers tape to the threads so that the nozzles fit on as tight as possible.
3. cord to the Wall Wart
4. Sprinkler Connection
5. Hose connected to water source

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

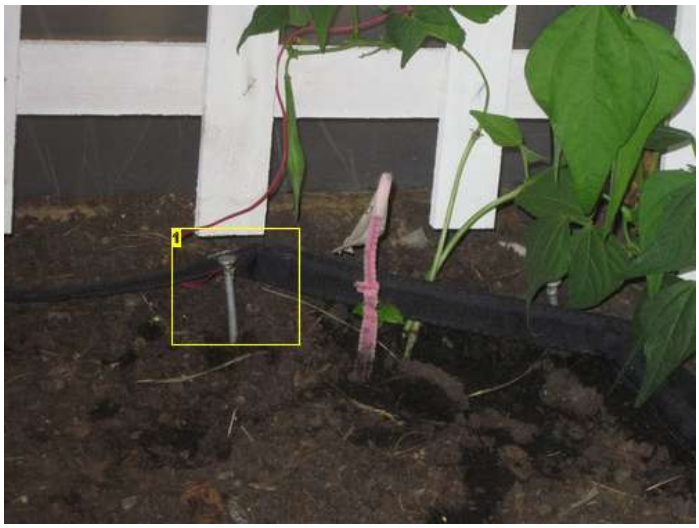


Image Notes
1. place the moisture sensors in a place that is representative for the entire garden

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

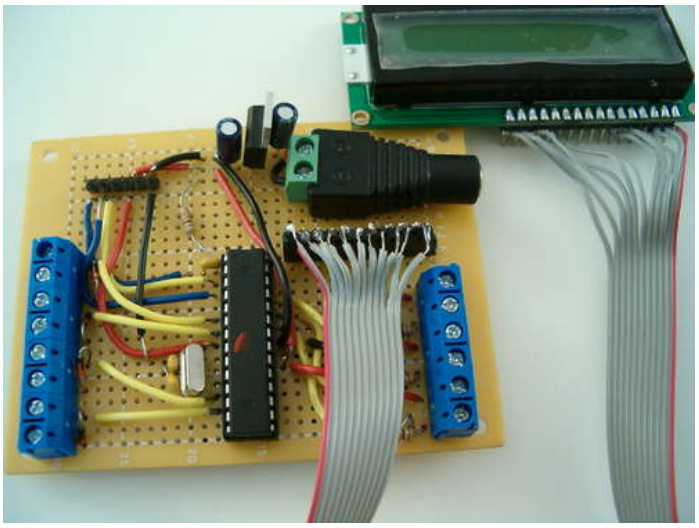
17. (4) 10k resistors

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.

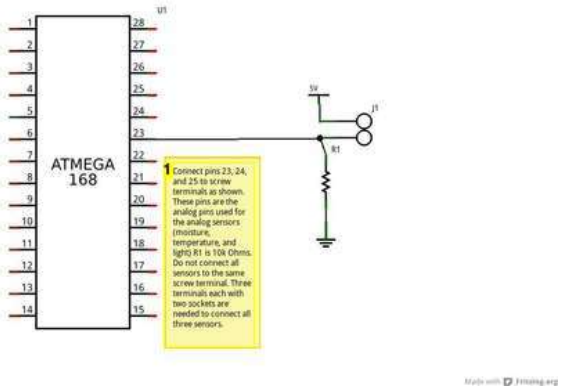


Image Notes

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.

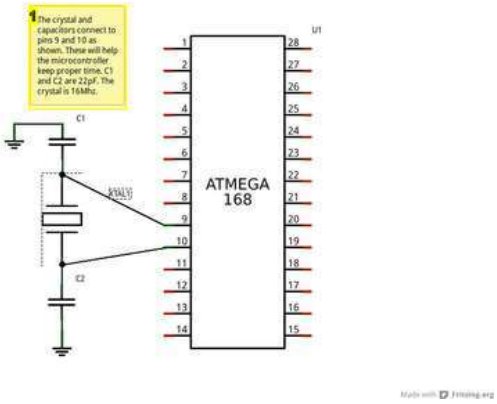


Image Notes

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.

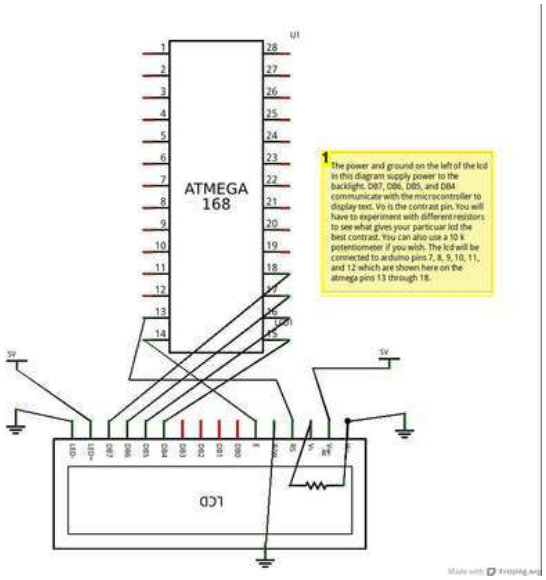


Image Notes

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

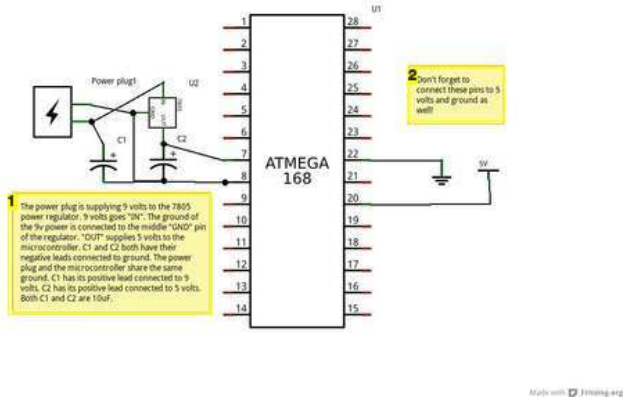


Image Notes

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.

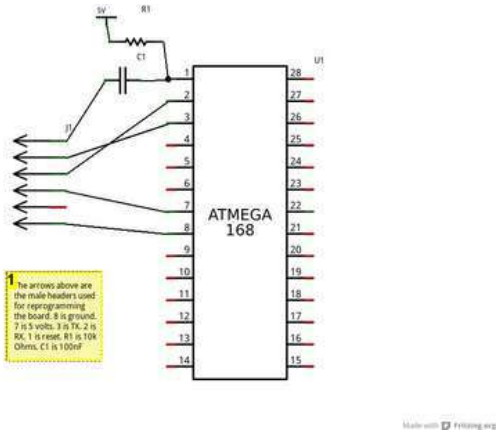


Image Notes
1. The arrows above are male headers used for reprogramming the board. 8 is ground. 7 is 5 volts. 3 is TX. 2 is RX. 1 is reset. R1 is 10k ohms. C1 is 100nF.

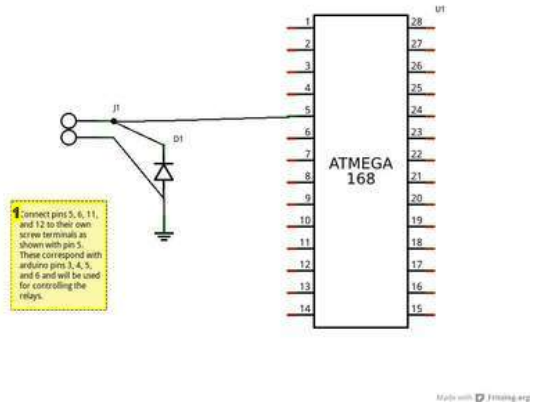


Image Notes
1. Connect pins 5, 6, 11, and 12 to their own screw terminals as shown with pin 5. These correspond with arduino pins 3, 4, 5, and 6 and will be used for controlling the relays.

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.

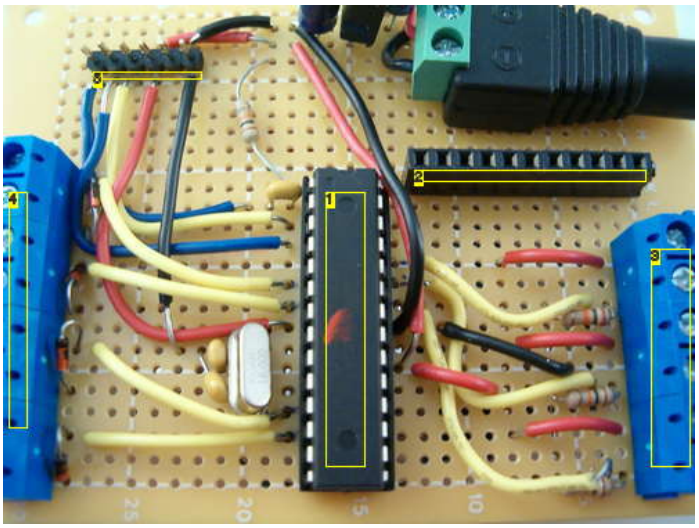


Image Notes
1. Atmega
2. LCD hookup pins
3. analog sensors
4. relay outputs
5. programming cables

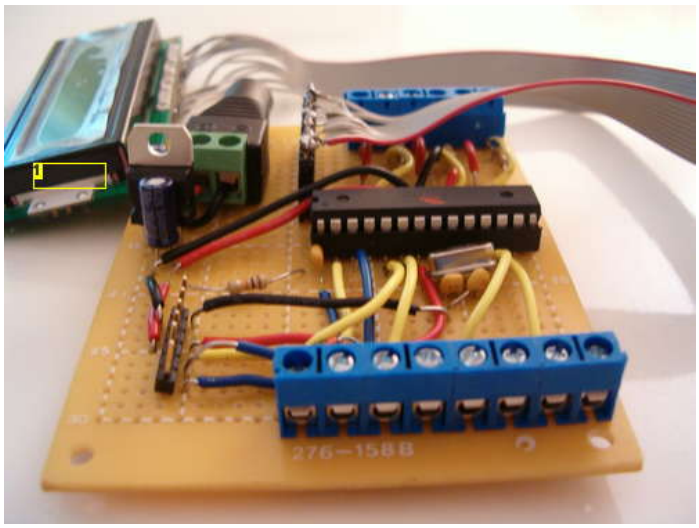
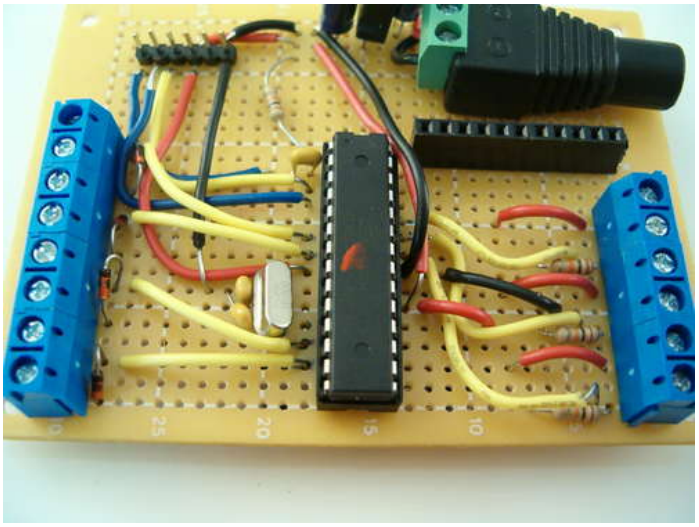


Image Notes
1. LCD



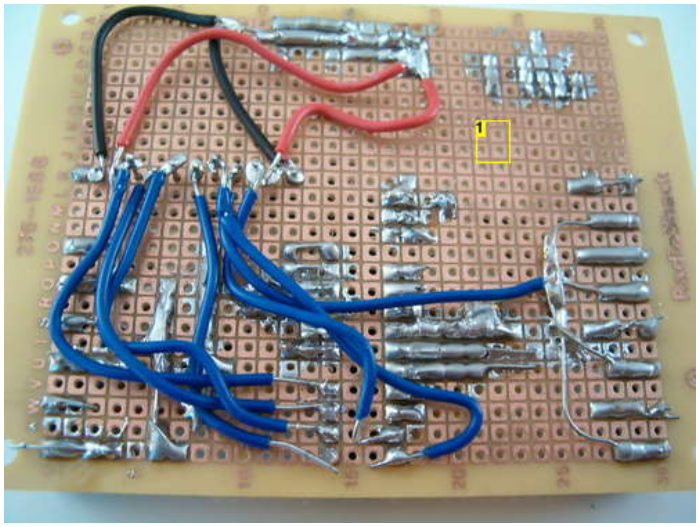
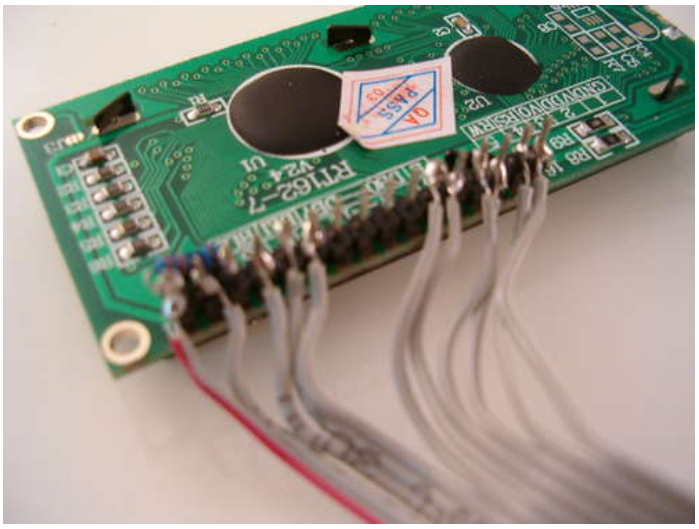


Image Notes
1. LCD wires are connected on the back of the board



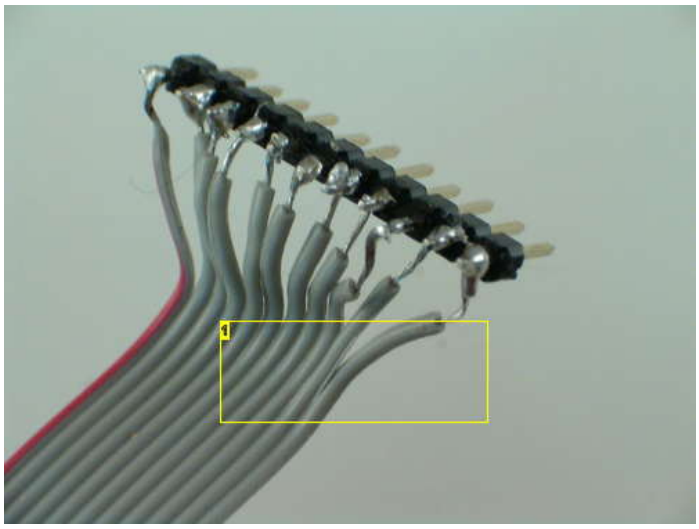


Image Notes
1. 11 wires come from the LCD

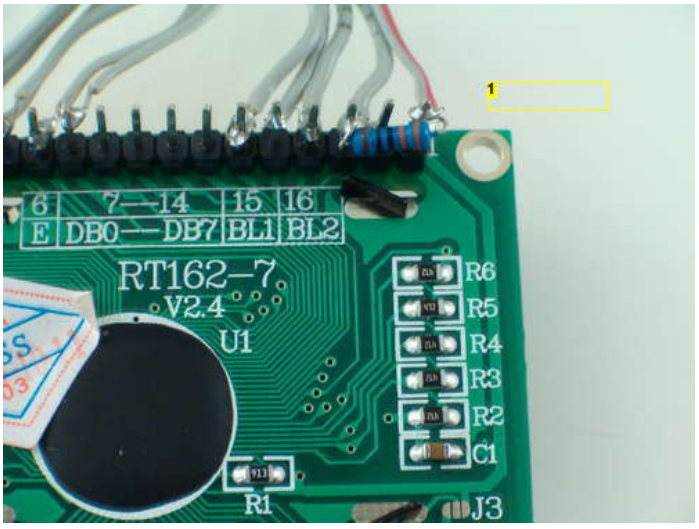


Image Notes
1. The resistor here connects to pins 1 and 3 of the LCD and controls the contrast. You can use a 10 k potentiometer instead if you don't want to bother testing resistors to find what value works with your screen

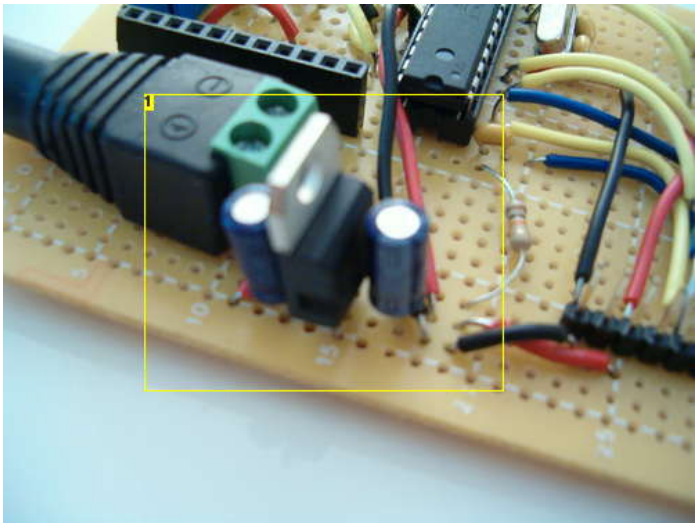


Image Notes
1. Power regulation.

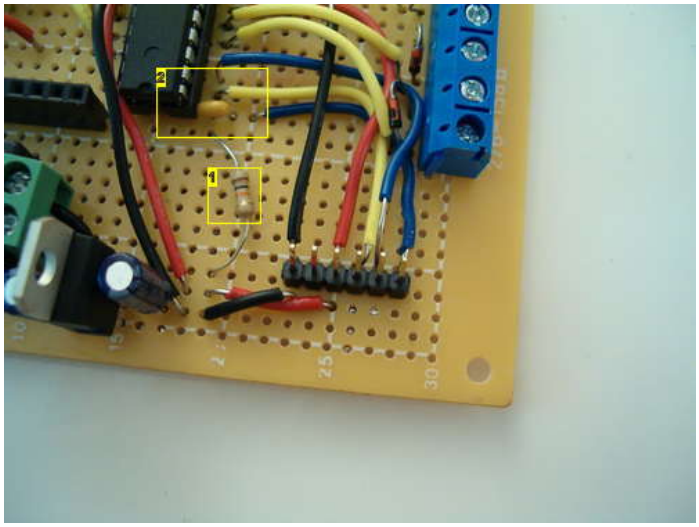


Image Notes
1. 10 k resistor goes from reset pin to 5 volts
2. DO not forget this 100nF capacitor! you need it to allow programming communication.

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

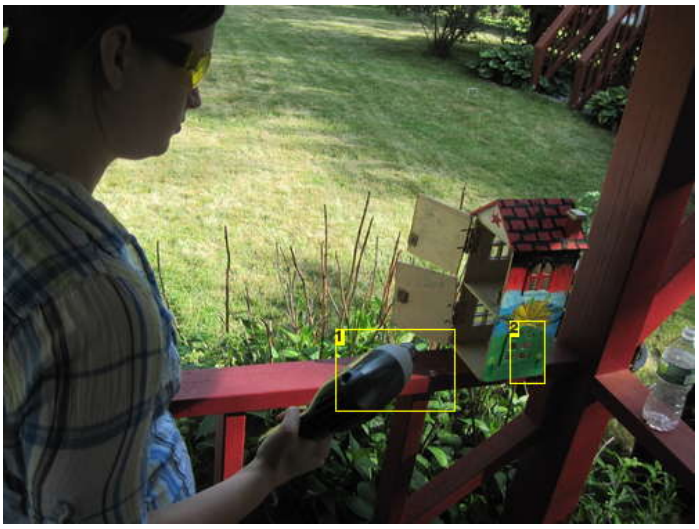


Image Notes
1. Securely Attach it to house
2. These windows were cut out so that the plug from the relay bow could reach the extension cord

Step 19: Creating the Birdhouse Motherboard

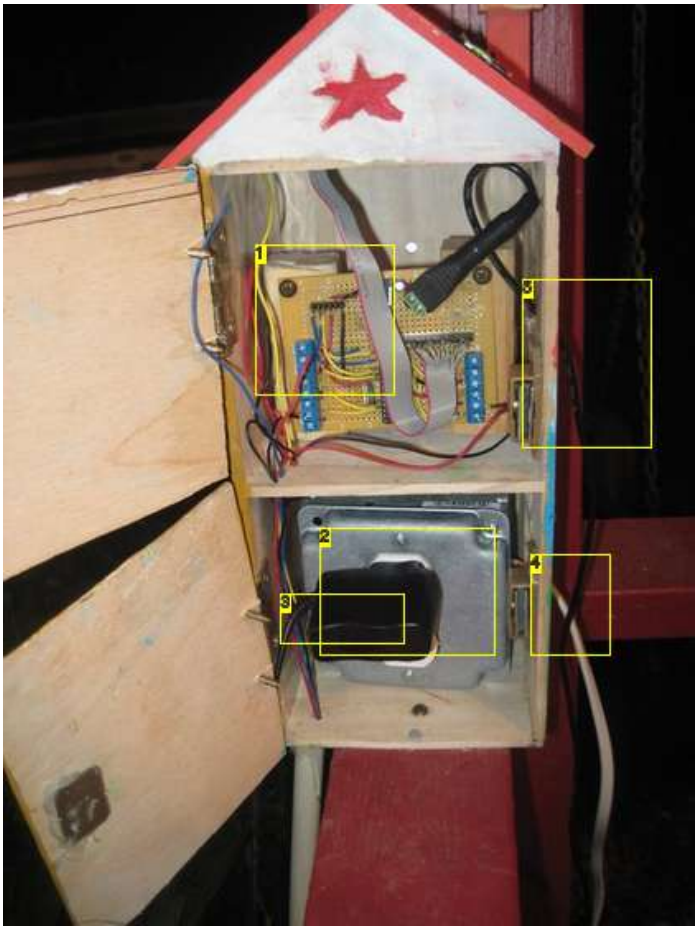


Image Notes
1. We nailed the board to pieces of wood that are offset from the back for easy access to the screw terminals
2. Relay box goes into the bottom shelf of the birdhouse
3. Wallwart that is connected to the solenoid valve (make sure it is plugged into the right outlet!)
4. cord from relay box 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!

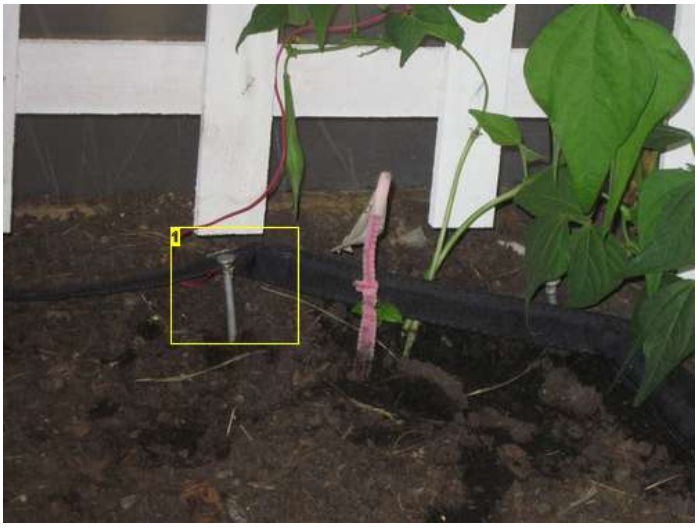


Image Notes
1. place the moisture sensors in a place that is representative for the entire garden

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:

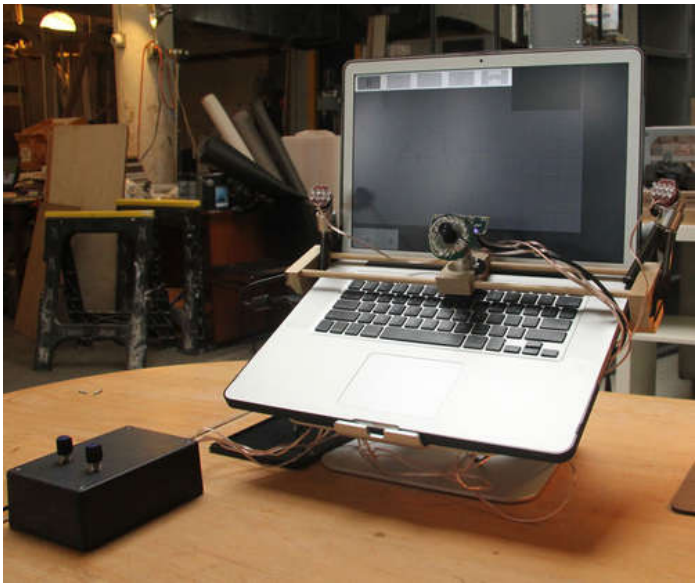
Since that first video, we've been hacking on and developing the project, and we have a new design, which we've called "eyewriter 2.0" which improves the accuracy of the device, and allow for people who's heads are moving slightly to also use an eye tracker. The original eyewriter, designed for a paralyzed 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.

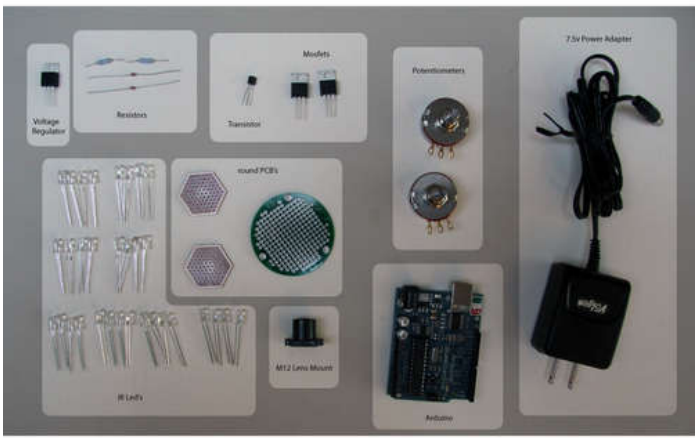
From a technical perspective, the 2.0 system works by strobing 3 IR illuminators every frame. On even frames, it uses the center illuminator (located around the camera lens) and on odd frames it uses the 2 side illuminators. On even frames, the pupil appears bright, since the IR light is actually bouncing 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)

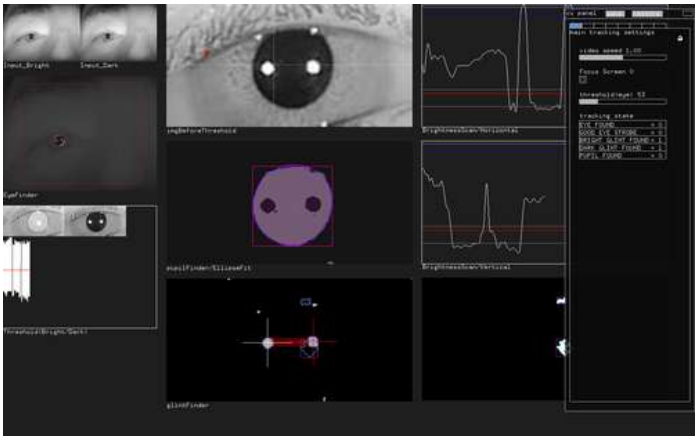
- An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.
- Download and install an Integrated Development Environment (IDE) to run openFrameworks if necessary.
  http://www.openframeworks.cc/setup

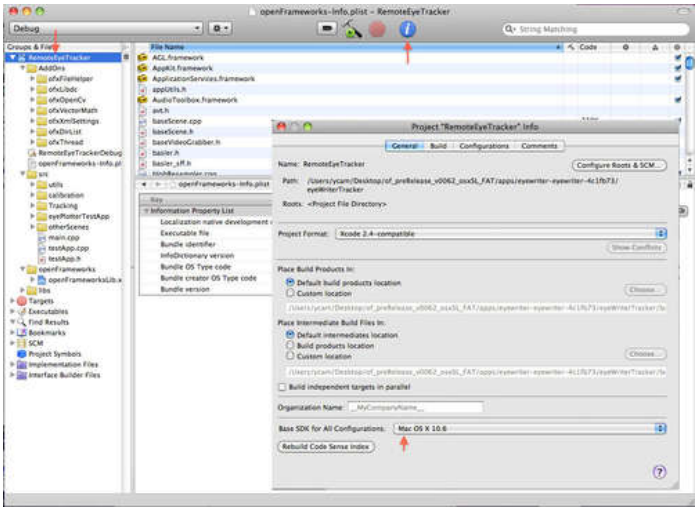
B. openFrameworks

- 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.





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.

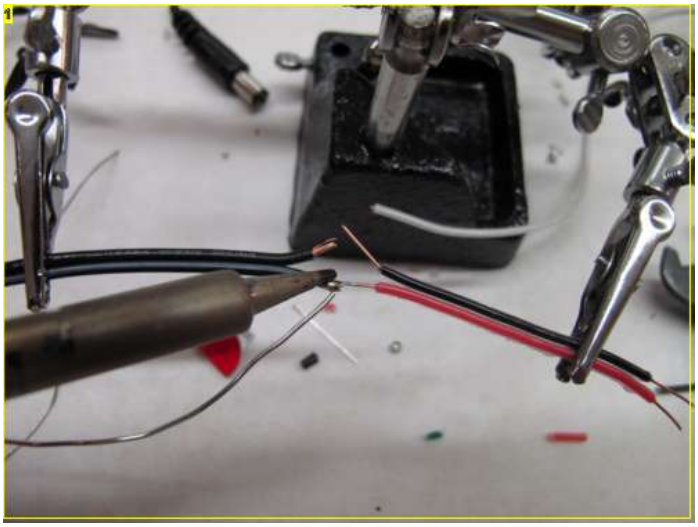


Image Notes

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.

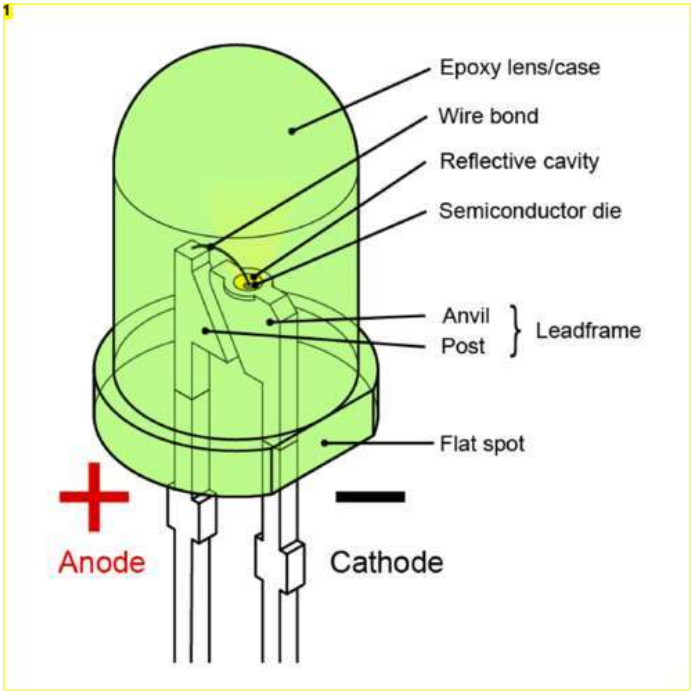


Image Notes
1. To build LED arrays on the PCBs you'll need to know the positive and negative ends of each LED. Generally speaking the longer leg of the LED is the anode (positive), and the shorter leg is the cathode (negative).

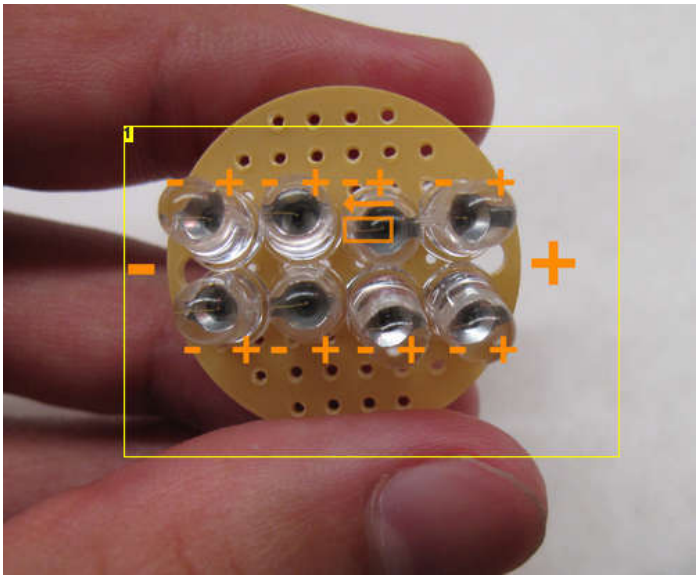


Image Notes
1. From overhead, take note of which direction the wire bond points relative to positive and negative. Setup a circuit of 4 LEDs in series, in parallel with another set of 4 LEDs in series.

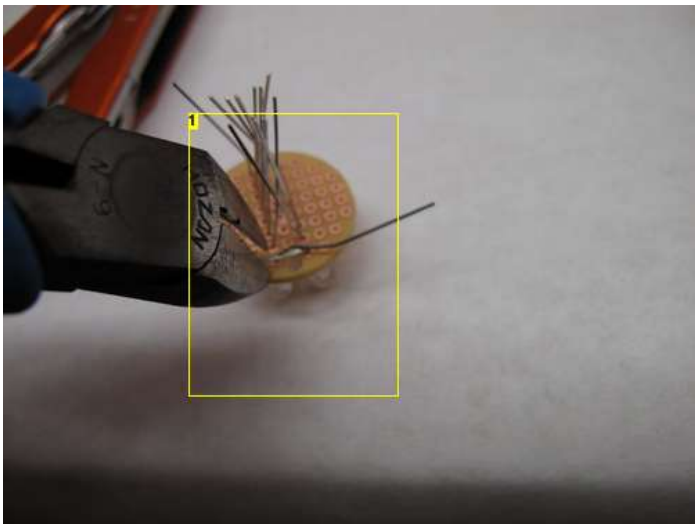


Image Notes
1. Clip the legs of the LEDs and solder them together.

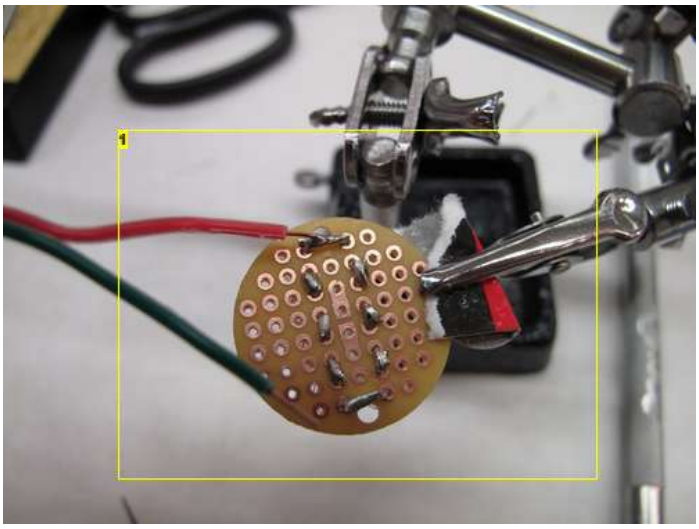


Image Notes
1. After soldering the LED legs together to form the circuit, solder about 2 feet (60 centimeters) of the red & green intercom wire to the LED circuit s positive & negative ends.

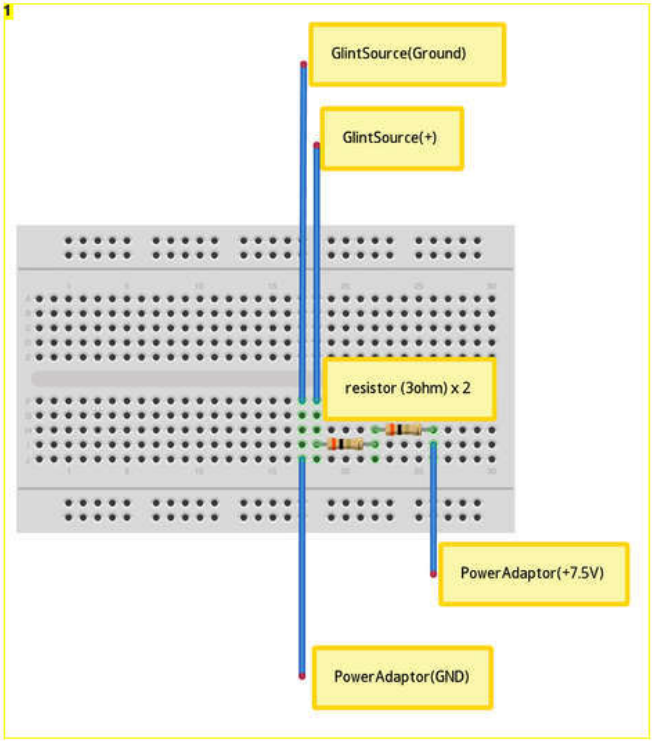


Image Notes
1. To test the LED PCB panel, build the circuit below. Look carefully to see if your IR LEDs are glowing a faint red.

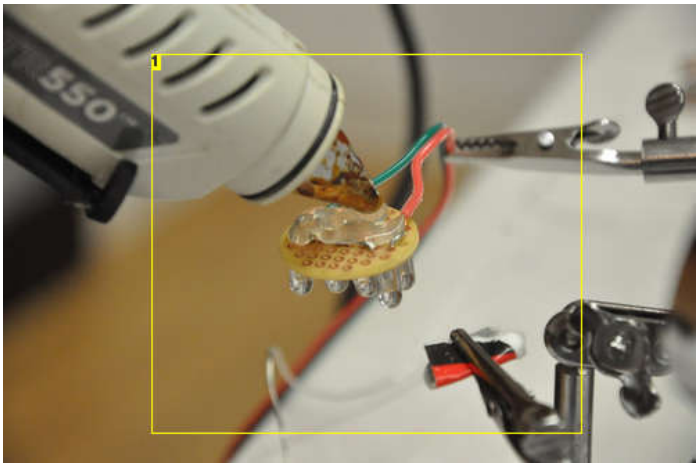


Image Notes
1. After confirming your IR LEDs are working, cover the back of the LED PCB panel with hot glue to keep all connections in place.

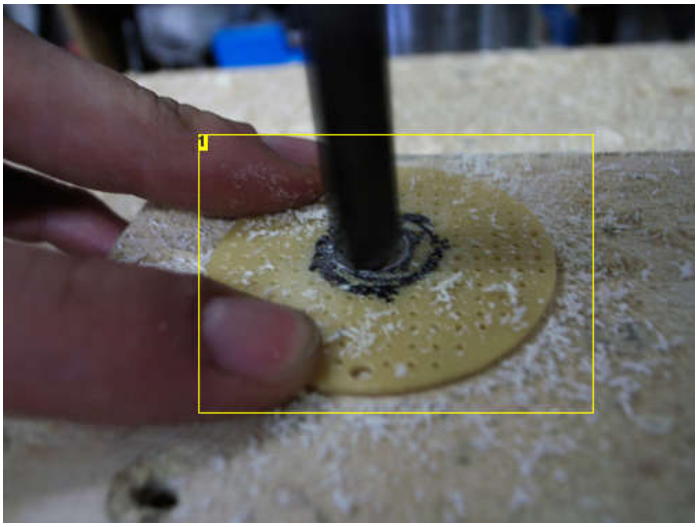


Image Notes
1. Using a larger round PCB, carefully drill press a hole into the center of the board.

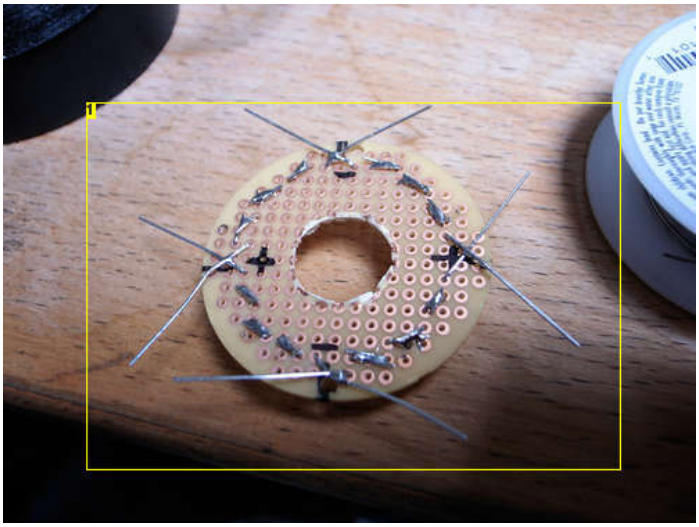


Image Notes
1. On the outer rim of the PCB, build a circuit of 4 parallel sets of 4 LEDs in series. The placement of the LEDs should allow the PS Eye camera to fit through snugly, without the camera blocking the LEDs.

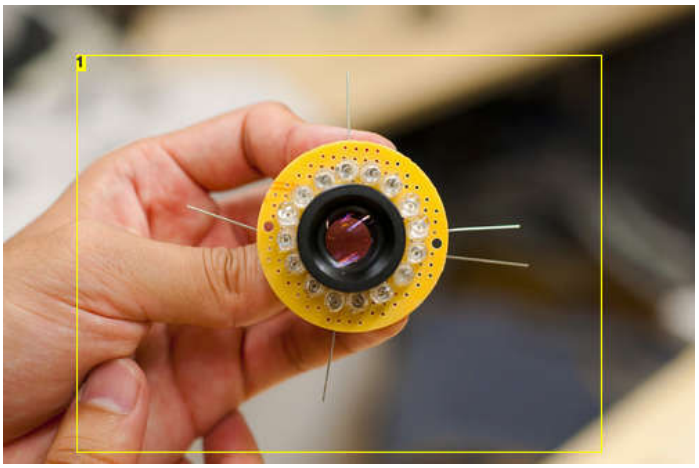


Image Notes
1. After soldering the LED legs together to form the circuit, solder wiring to connect all 4 positive ends together and all 4 negative ends together, putting all 4 LED sets in parallel.

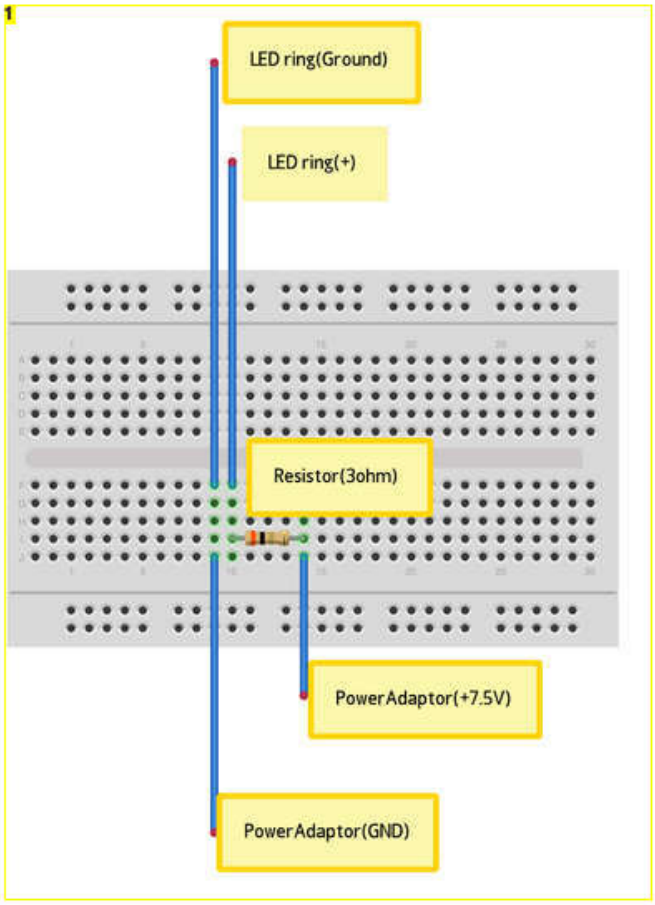


Image Notes
1. To test the larger LED PCB panel, build the circuit below. Look carefully to see if your IR LEDs are glowing a faint red.

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.

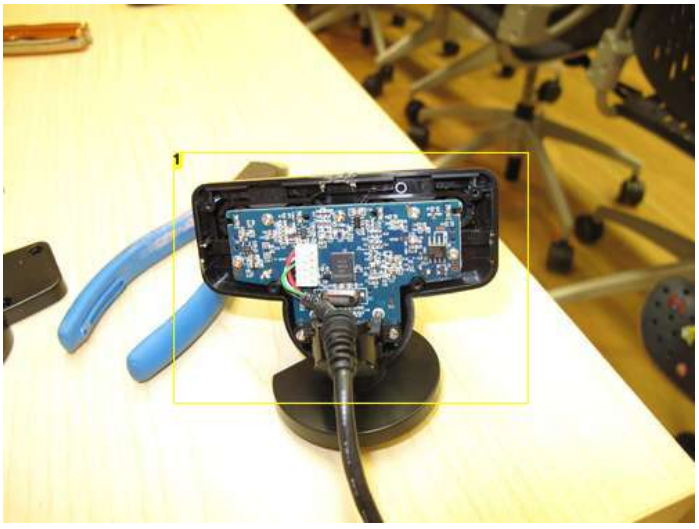


Image Notes
1. Pull the cord aside and unscrew the two bottom screws beside the plastic

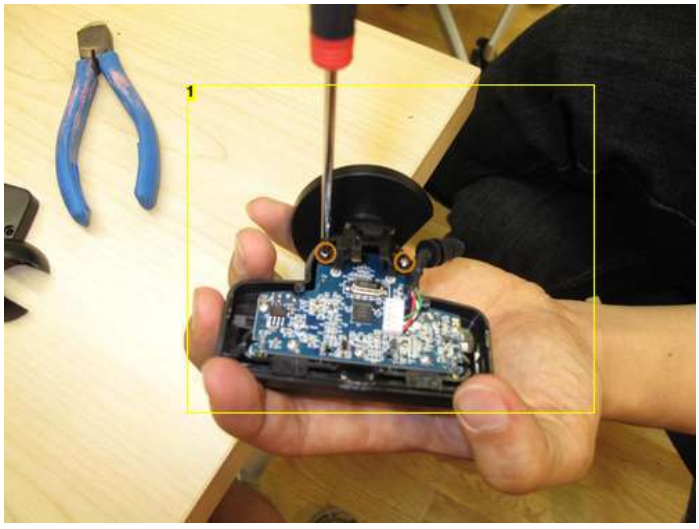


Image Notes
1. Pull the cord aside and unscrew the two bottom screws beside the plastic

holder. Keep these screws also.

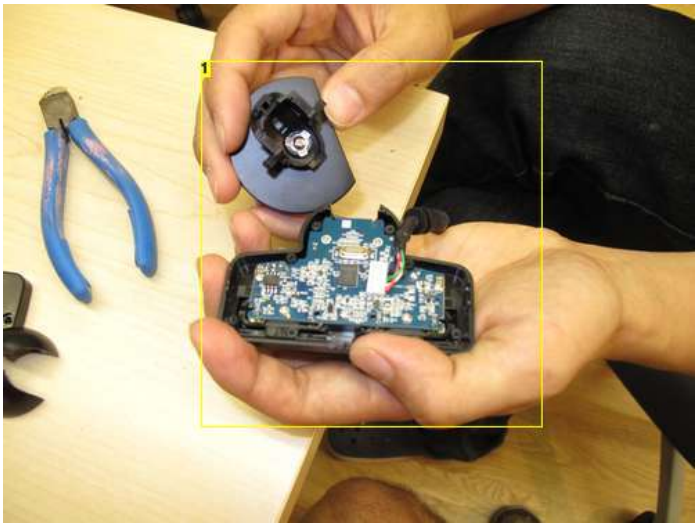


Image Notes
1. Remove the stand piece.

holder. Keep these screws also.

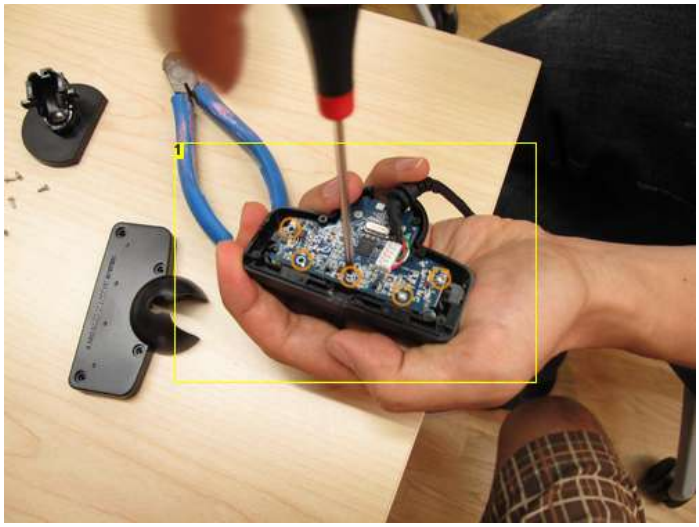


Image Notes
1. Unscrew the five screws around the board (two screws on the side, three screws on top). Keep these screws also.

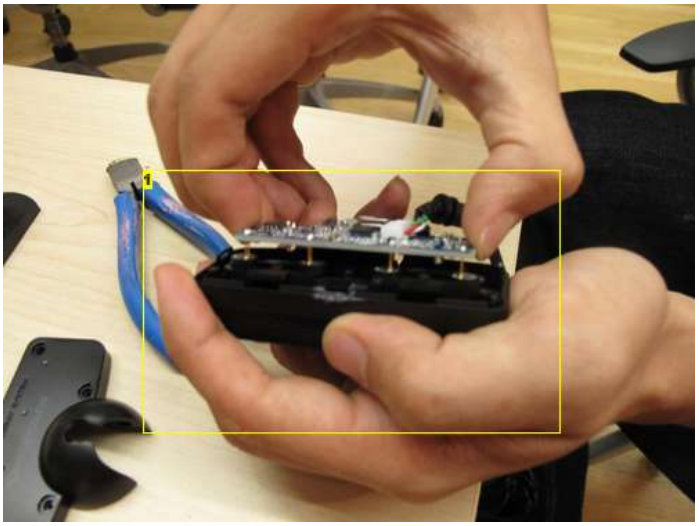


Image Notes
1. With all five screws removed, lift the board out of the front casing.

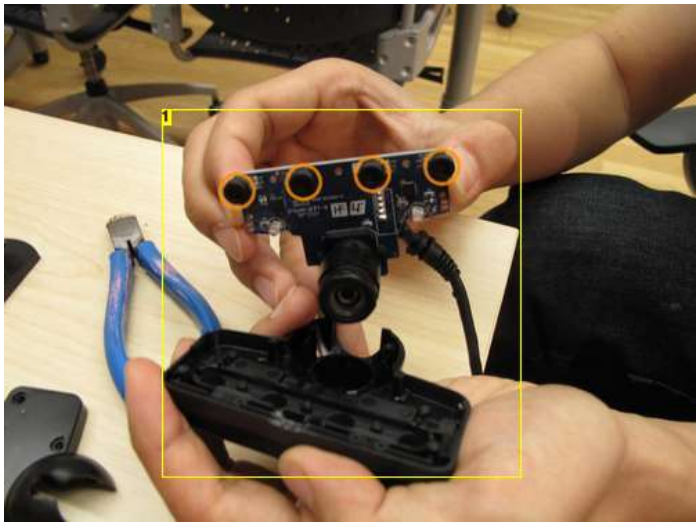


Image Notes
1. There are four microphones across the top of the board. Using wire cutters, clip off the microphones because they won't be used.

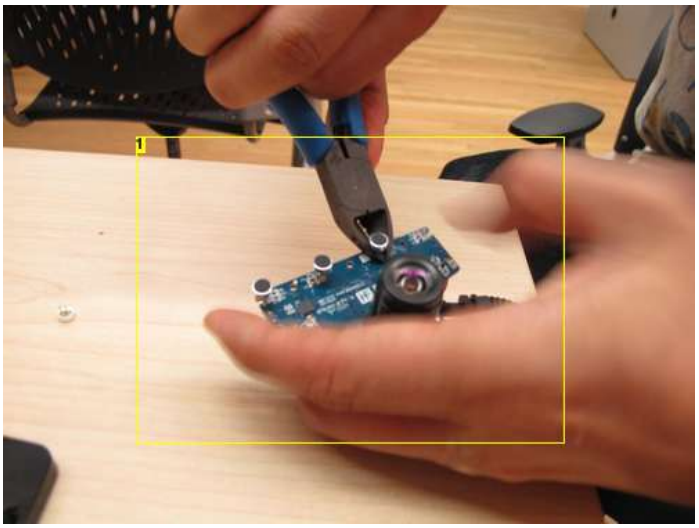


Image Notes
1. There are four microphones across the top of the board. Using wire cutters, clip off the microphones because they won't be used.

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.

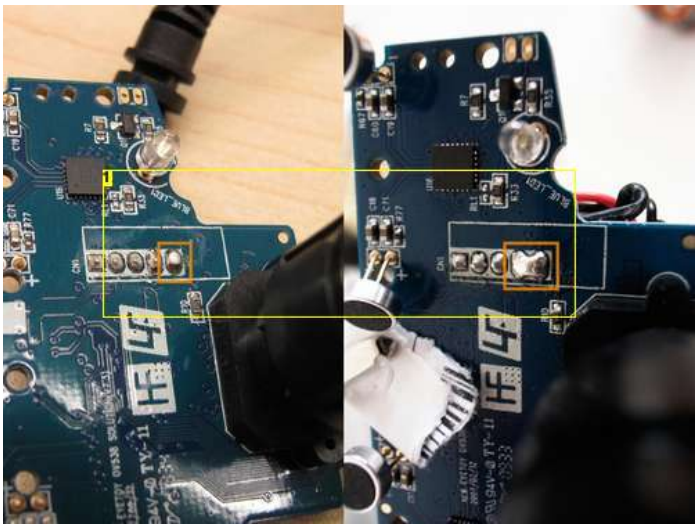


Image Notes
1. Locate the Ground joint on your PS Eye board. Some PS Eye models have 5 joints near the lens mount (left image below), while some have 4 joints (right image below). If your model has 5 joints, the Ground joint is at the end closest to the lens mount. If your model has 4 joints, the Ground joint is also at the end closest to the lens mount, and twice as wide as the other joints.

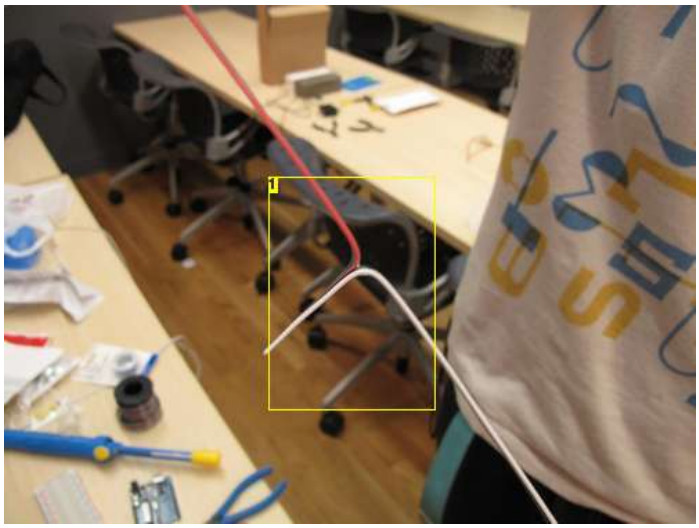


Image Notes
1. Cut about 2 feet (60 centimeters) of your 4-color intercom wire, and split the red and green from the black and white.

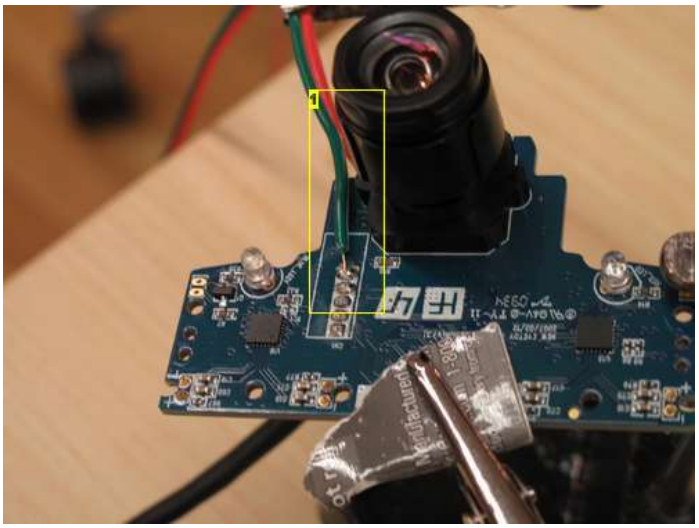


Image Notes
1. Split the red and green wire about 2 inches (5 centimeters) from one end, and strip off a small section of insulation at the end of the green wire. The green wire will be soldered to the PS Eye s Ground joint. Clip the PS Eye board and green wire to a stand, and prepare to solder the green 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.

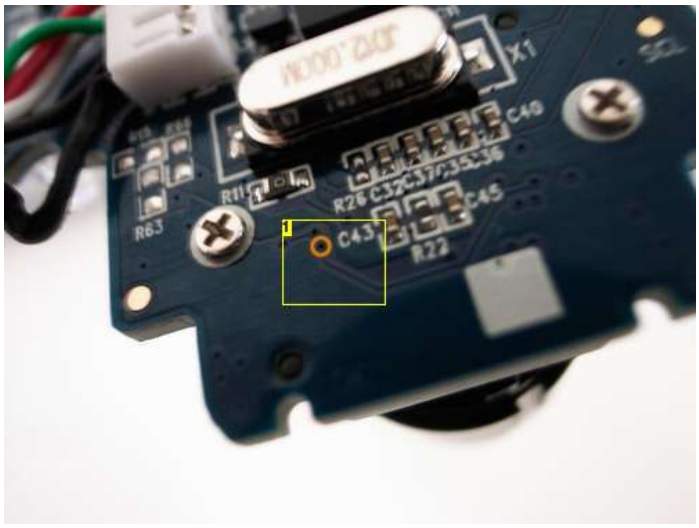


Image Notes
1. Locate the V-Sync via on the board. It s the via circled in the image below.

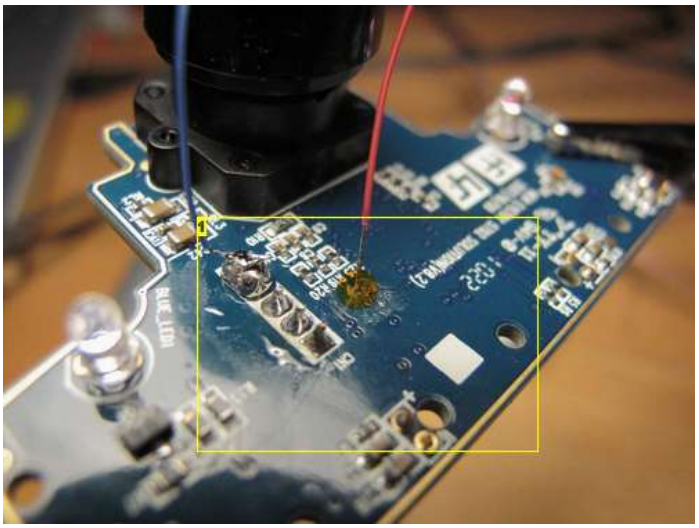


Image Notes
1. for more recent models of the PSEye camera (identified by the golden rim around the board) the VSync hotspot can be found on the front of the PCB, directly above the R19 resistor

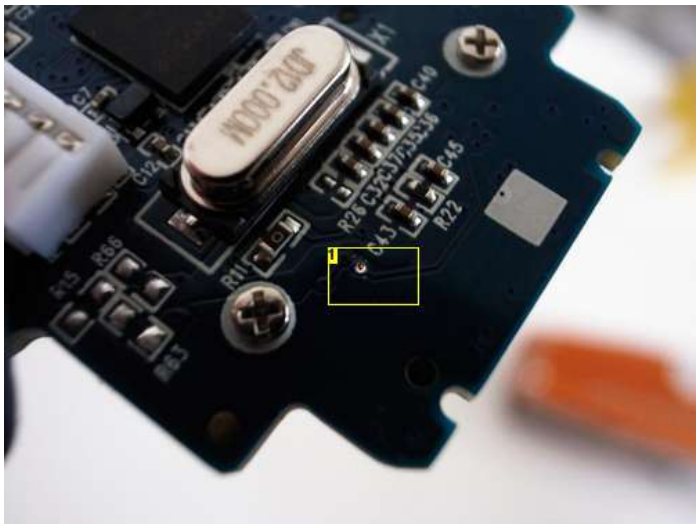


Image Notes
1. Using a sharp knife, carefully pivot the knife tip on the via, and scrape off enough insulation coating to expose the metal contact below.

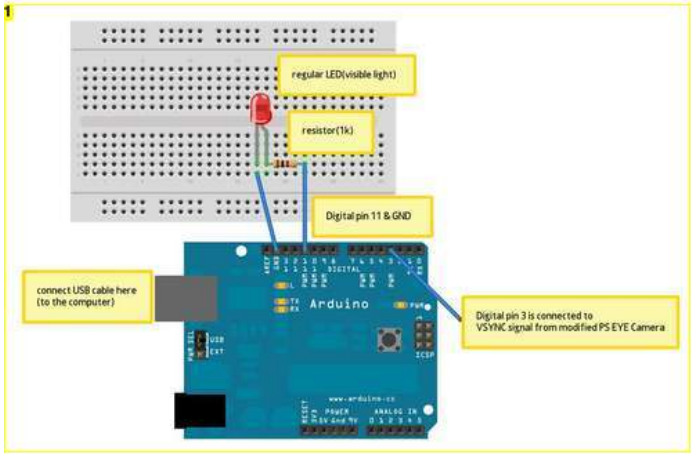


Image Notes

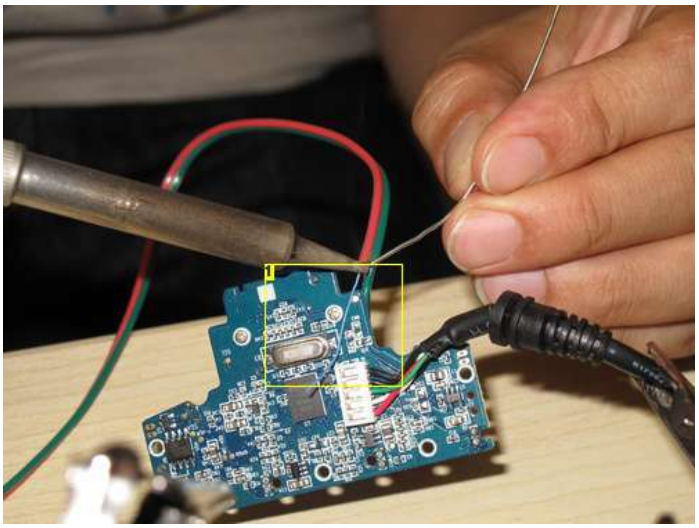
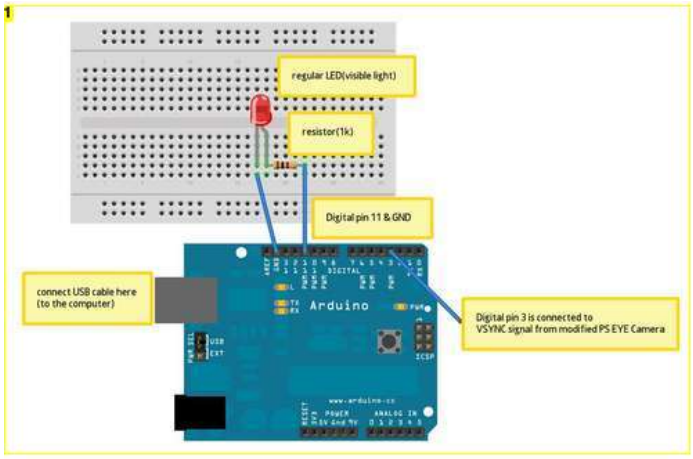


Image Notes
1. The red wire needs to connect to the exposed V-Sync via, but the wire is too thick to be soldered neatly to the small via, so a 30 gauge wire will be used in between. Strip the ends of a 2 piece of 30 gauge wire. Shorten the red wire, then solder one end of the 30 gauge wire to the end of the red wire.



1. 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

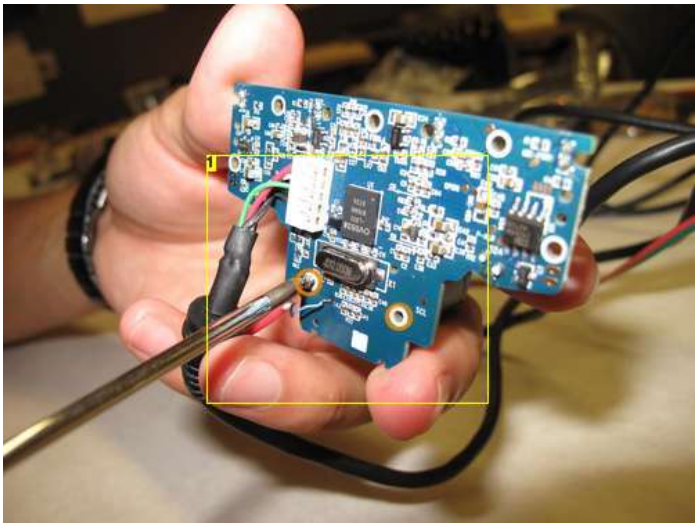


Image Notes
1. Unscrew the 2 screws holding the lens in place. Be careful not to break the fragile V-Sync connection. Detach the lens and keep both screws.

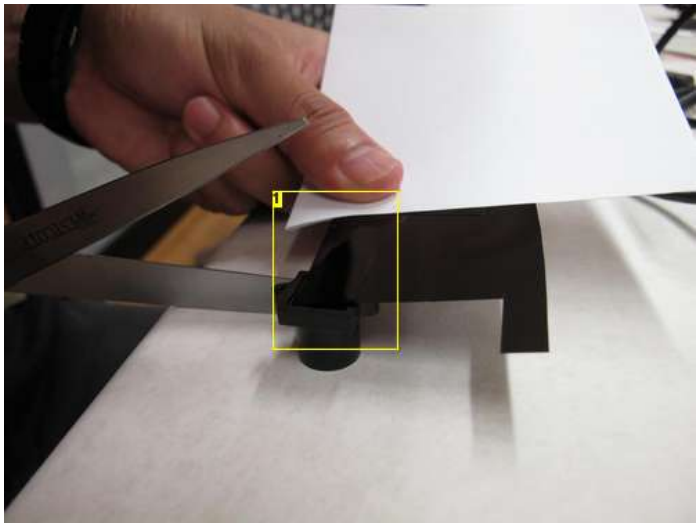


Image Notes
1. Measure the square opening of the new lens mount. Cut a square from the filter sheet that is minutely smaller, and place it into the lens mount opening.

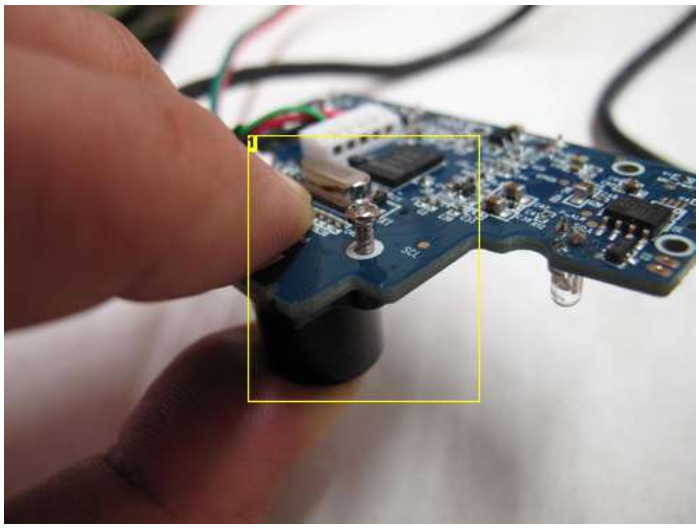


Image Notes
1. With the filter in place, screw in the new lens mount. This will require some force, and one screw will go in at an angle because the new lens mount is a little too big for the board.

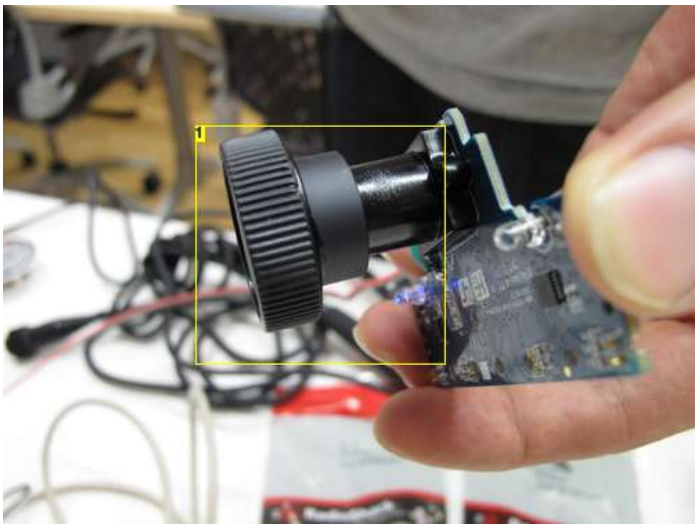


Image Notes
1. Screw the new lens into the new lens mount on the board.

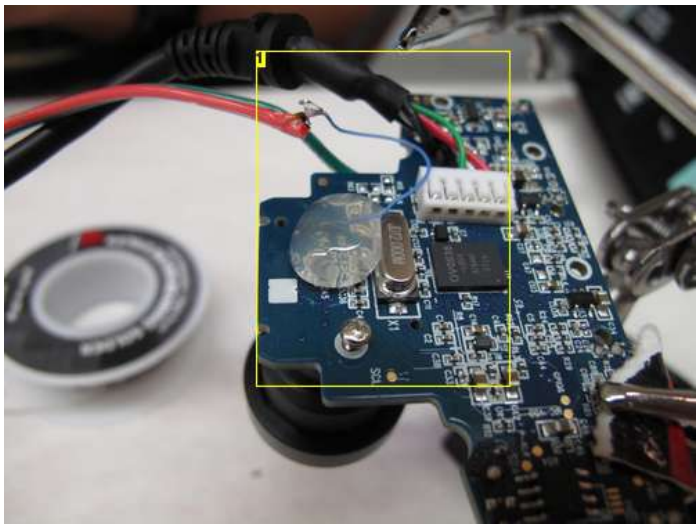
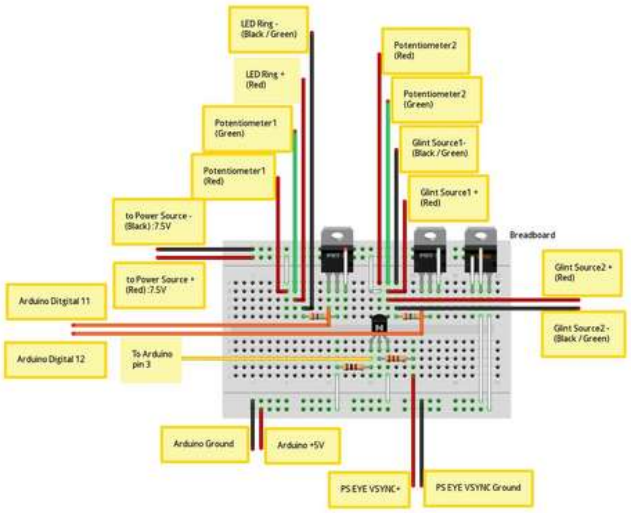


Image Notes
1. Use hot glue to cover and secure the V-Sync connection.

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.



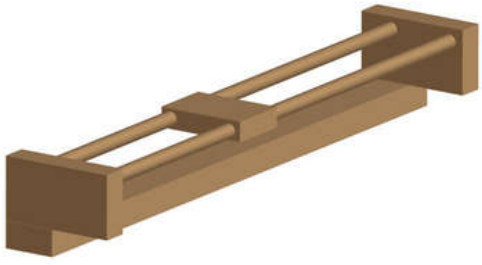
Step 12: Building a wood base

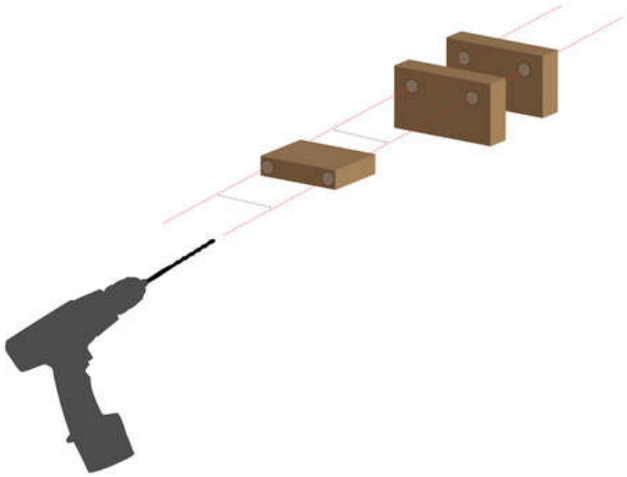
In this step we talk about building a portable wood base. It is interesting to build this so that your system can have a stable infrastructure to rest on. This makes it easier to test, calibrate, and work with the eyeWriter.

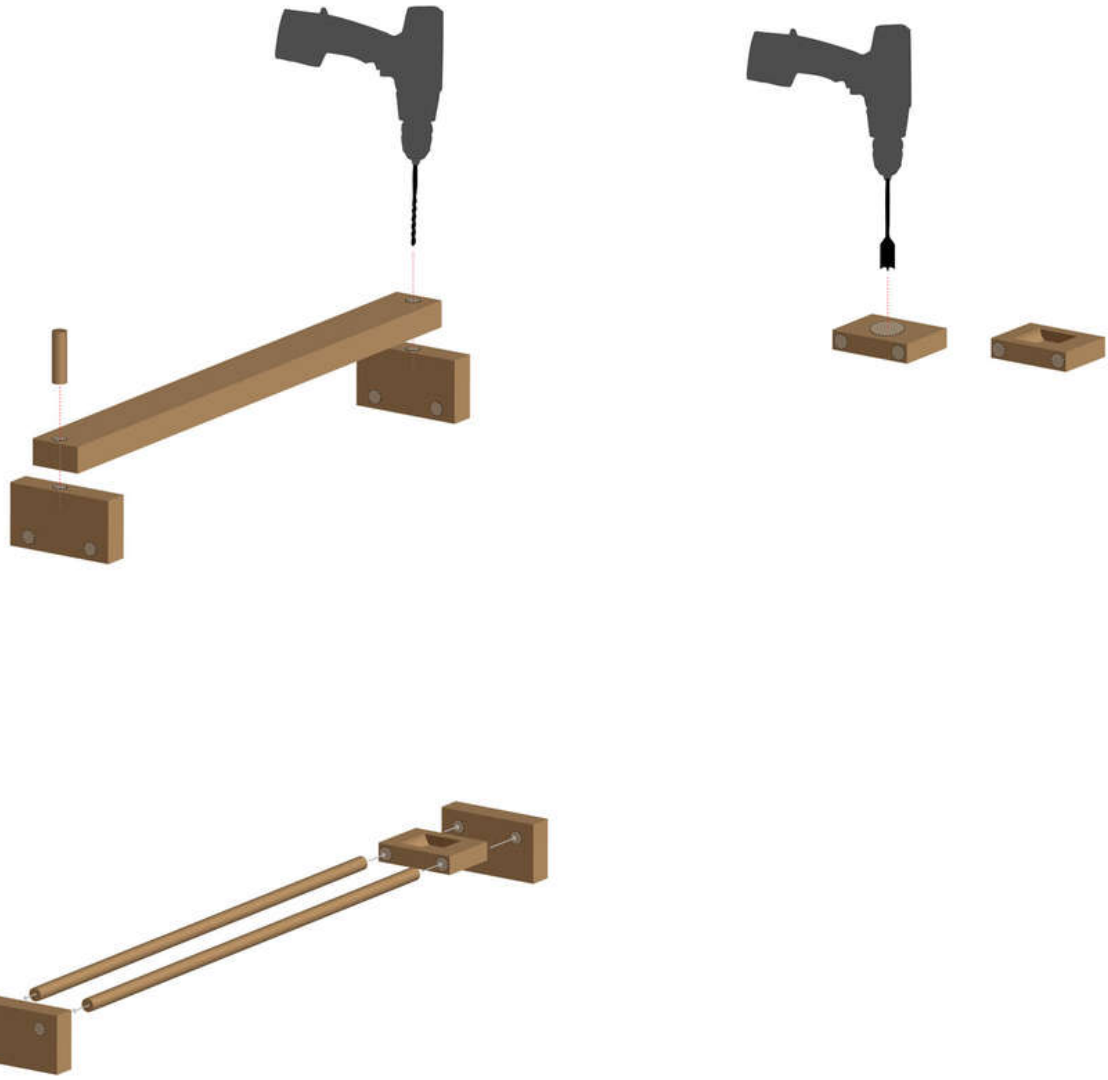
list of materials/parts needed for the base:

- 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







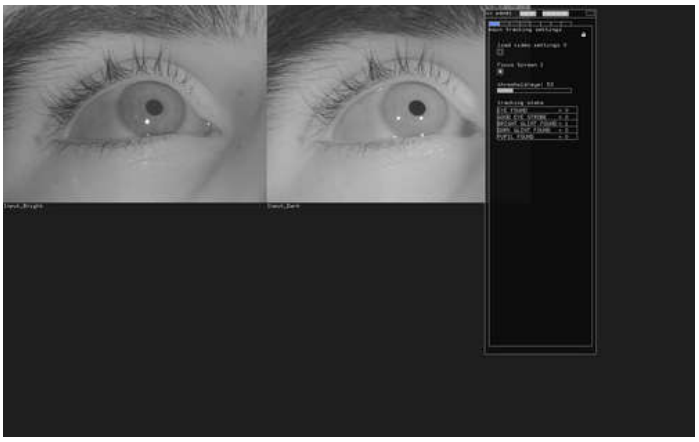
Step 13: Using EyeWriter Software - Setup & Tracking Screen

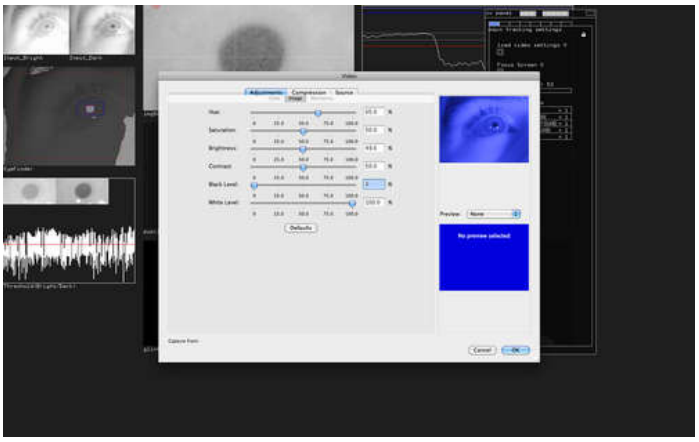
In this step we will take you around the eyeWriter software so that you can set it up.

- 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

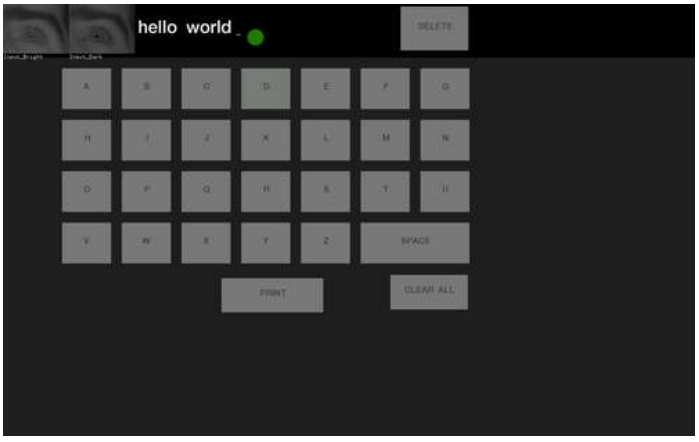
- Drawing mode starts paused by default. Before you start drawing, you can toggle the background grid on or off. The background grid can be toggled at any time by pausing.
- To start drawing, switch to recording mode by staring at the paused button. As you stare, the button will turn green and switch 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.

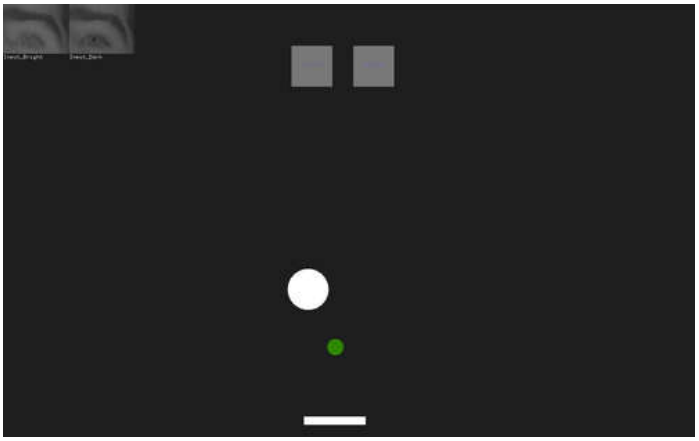
Step 17: Using EyeWriter Software - Typing

- 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

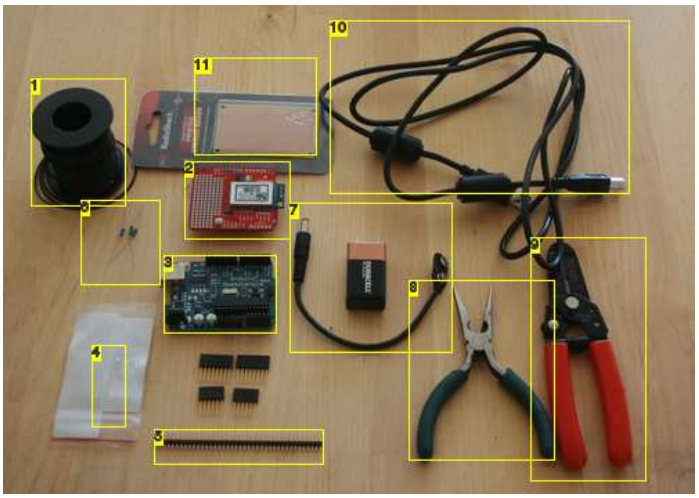


Image Notes
1. wire
2. Wifly Shield
3. Arduino Duemilanove
4. RGB LED
5. Breakaway headers
6. 3x resistors (2x100 ohm,1x180 ohm)
7. 9v battery and Barrel Jack Adapter
8. Pliers
9. Wire cutters
10. USB Cable A to B
11. small printed circuit board

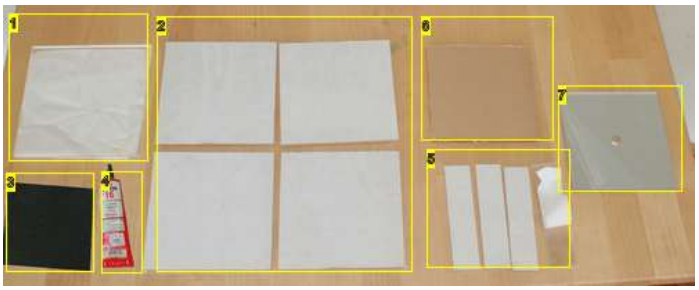


Image Notes
1. 1 x (5" x 5" x 0.25") - the top
2. 4 * (4.75" x 4.75" x 0.25") - the 4 walls
3. sandpaper
4. Acrylic solvent cement
5. 4 x (4.25 x 1" x 0.25") - the 4 inside walls
6. 1 x (4.5" x 4.5" x 0.25") - the base
7. 1 x (4.5" x 4.5" x 0.125") - the mirror with a 6mm hole drilled in the middle

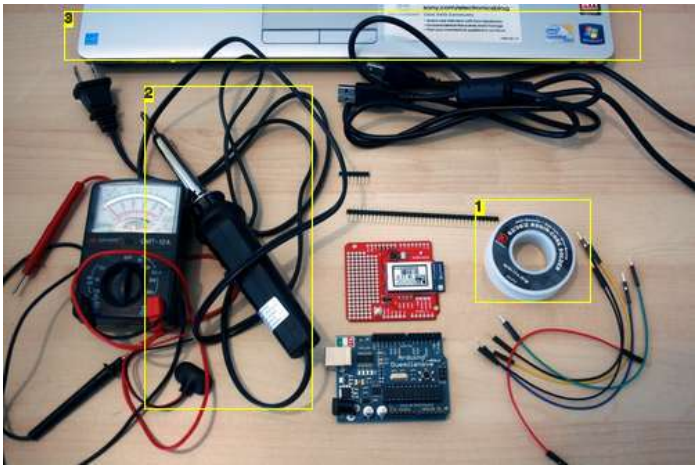


Image Notes
1. solder
2. soldering iron
3. laptop

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

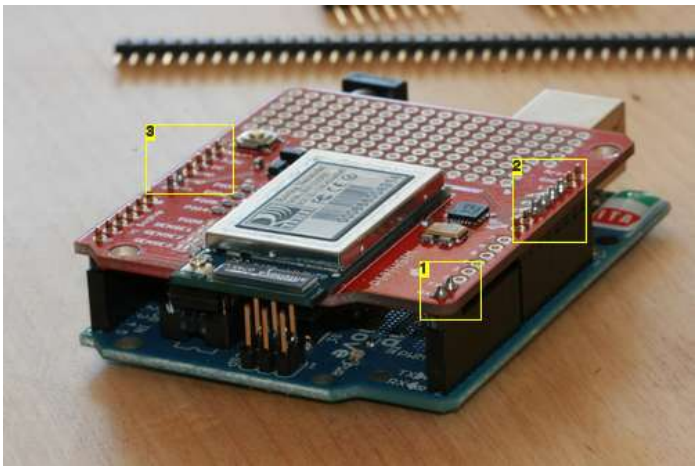
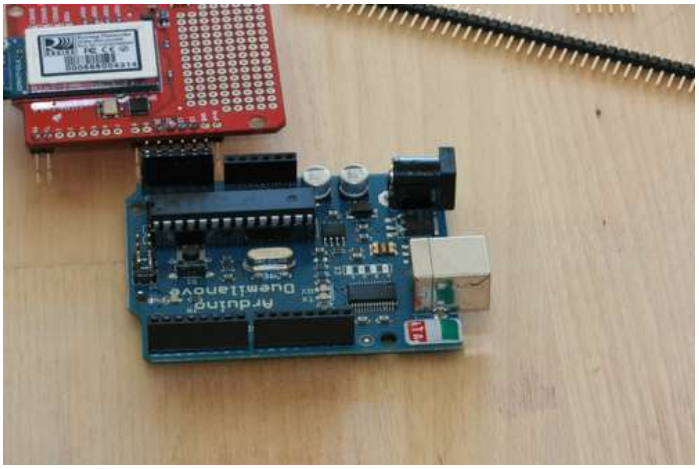
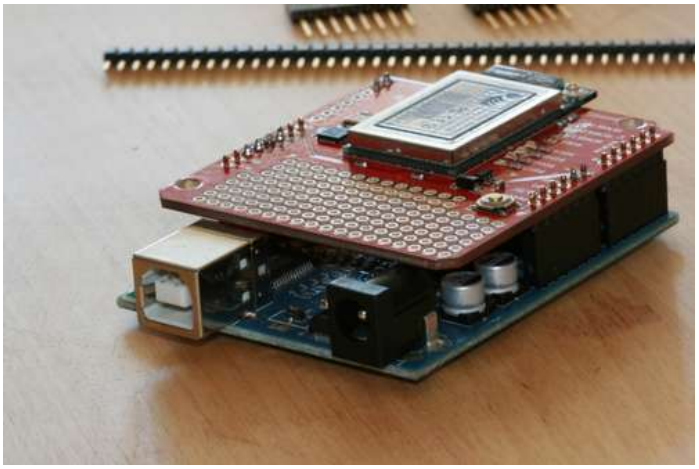
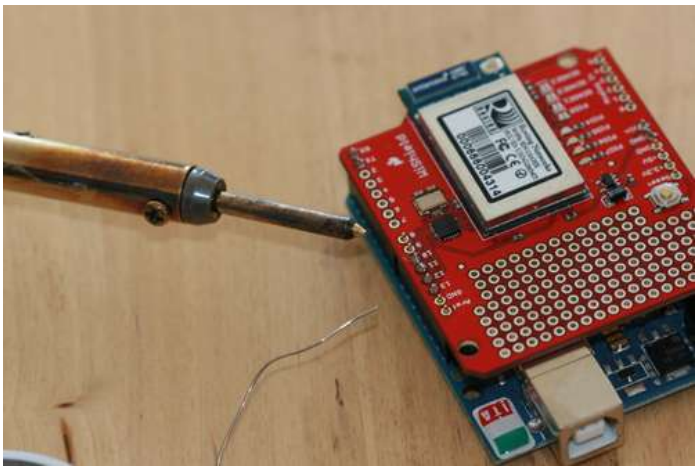
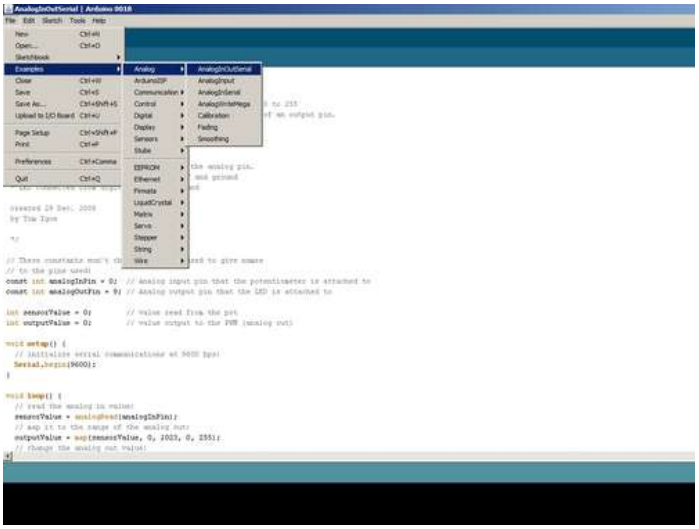


Image Notes
1. soldered RX, TX
2. Soldered 10,11,12,13
3. Soldered Vin, Gnd









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)

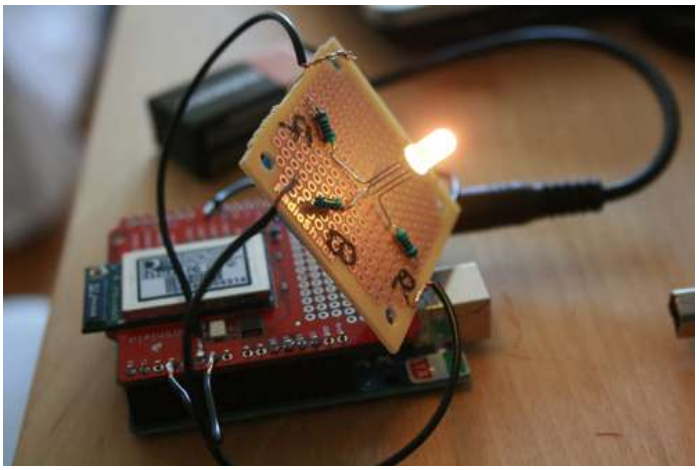
I used 3, 5 and 6.
I used the pliers to bend the legs of the LED, and mounted it on the circuit board. Each resistor is then mounted next to each of the RGB legs, and the wires are twisted together. Then I added the 4 connecting wires and twisted them. Finally, I soldered all the connections.

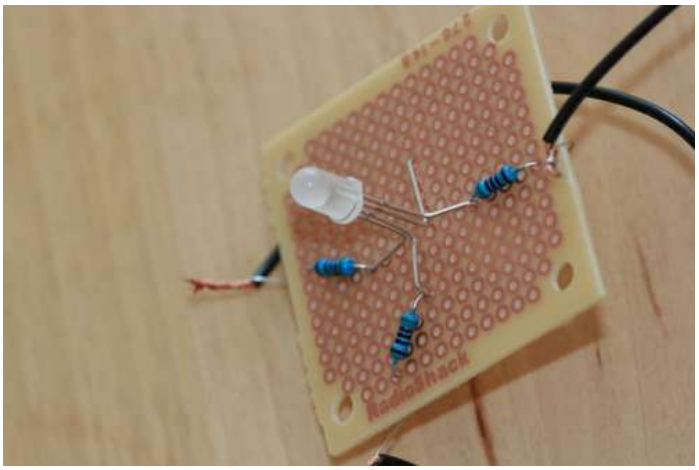
Note: The pictures illustrate using the same resistor for each colour channel, but I should have used the resistance levels in the data sheet:

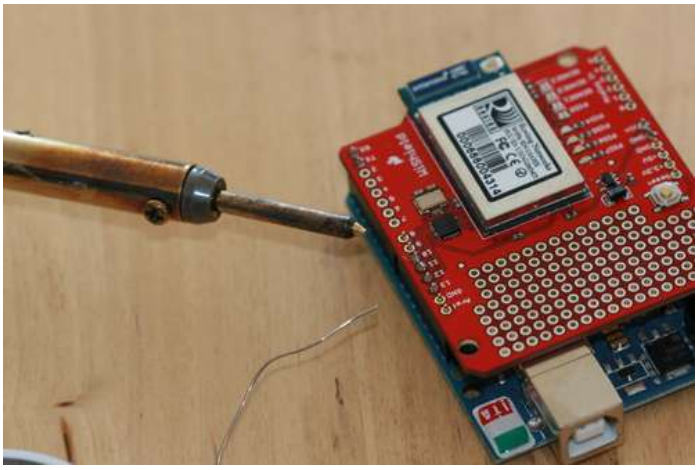
180 Ohm for Red
100 Ohm for Green
100 Ohm for Blue

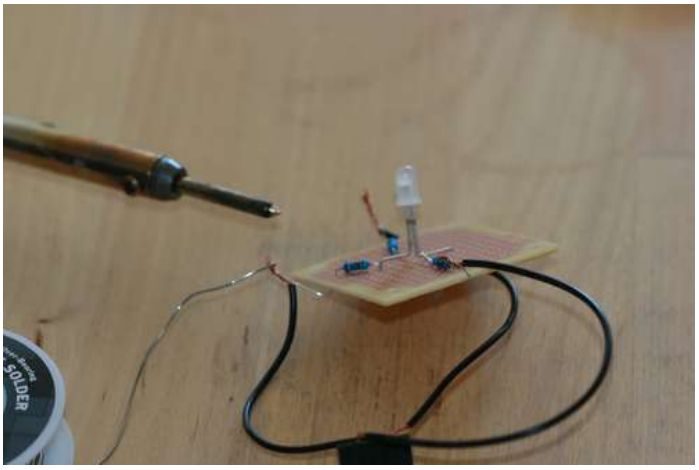
Also note, I covered the back with insulating tape to stop any shorts when putting it all into the box.
Also, from the datasheet, "the Sensor inputs SENS0-7 are extremely sensitive to over voltage. Under no conditions should these pins be driven above 1.2VDC. Placing any voltage above
this will permanently damage the radio module and render it useless."

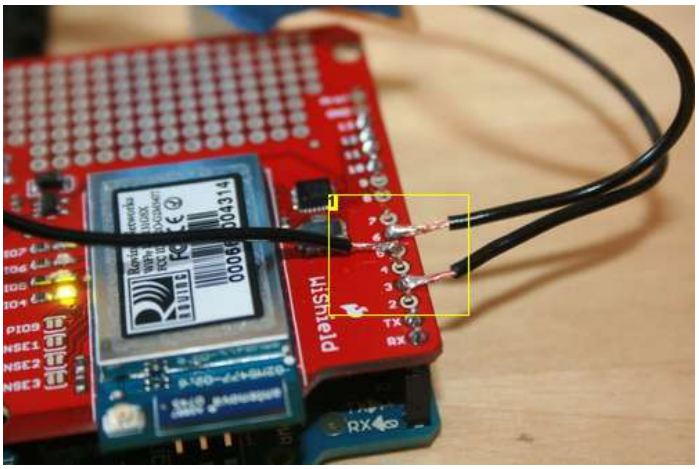
wiring.org.co/learning/basics/rgbled.html
www.sparkfun.com/datasheets/Components/YSL-R596CR3G4B5C-C10.pdf

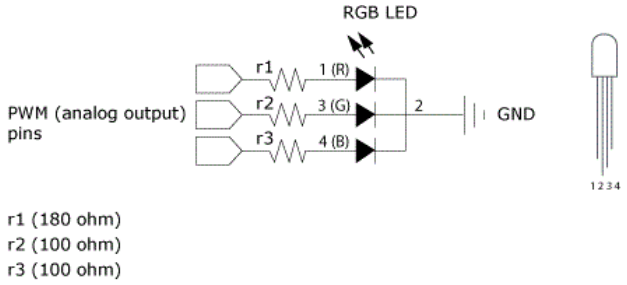












r1 (180 ohm)
r2 (100 ohm)
r3 (100 ohm)

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

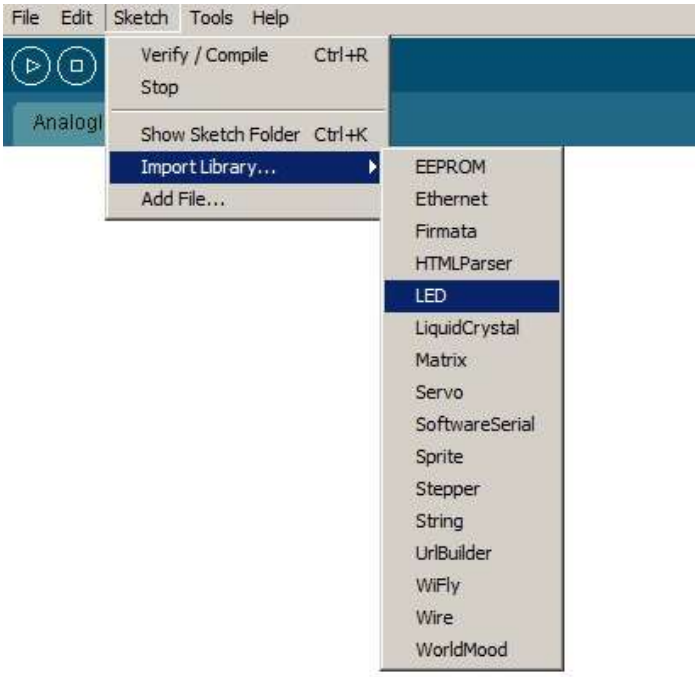
http://www.sparkfun.com/commerce/product_info.php?products_id=9367

******* 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.

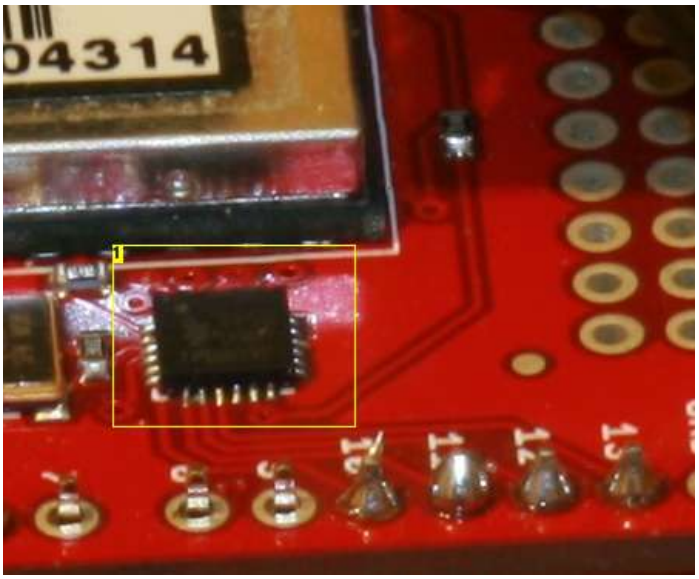
1. The PC communicates over UART with the Arduino through pins RX and TX
2. The Arduino communicates over SPI with the SPI-UART chip on the WiFly shield (SC16IS750 SPI-to-UART chip) though pins 10-13 (CS, MOSI, MISO, SCLK respectively)
3. The RN-131G wireless module accesses network and send/receive 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
}
```



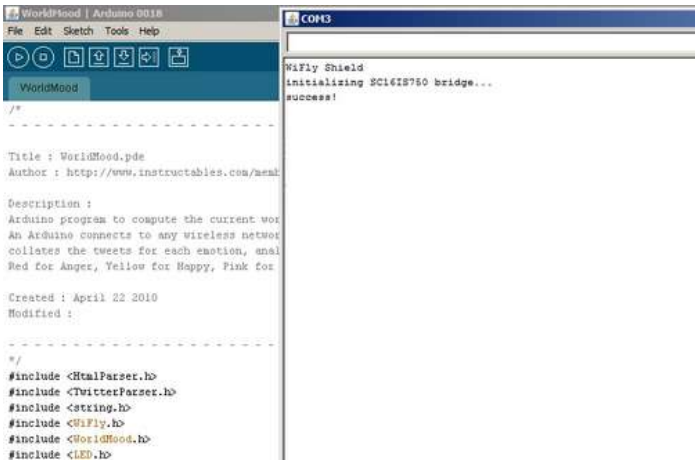


Image Notes
1. the SC16IS750 SPI-to-UART chip!

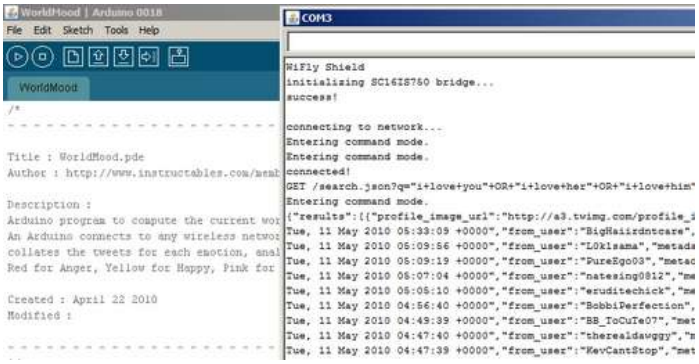
Step 8: Programming step 2: Connecting to a Wireless Network

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");
}
```



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.

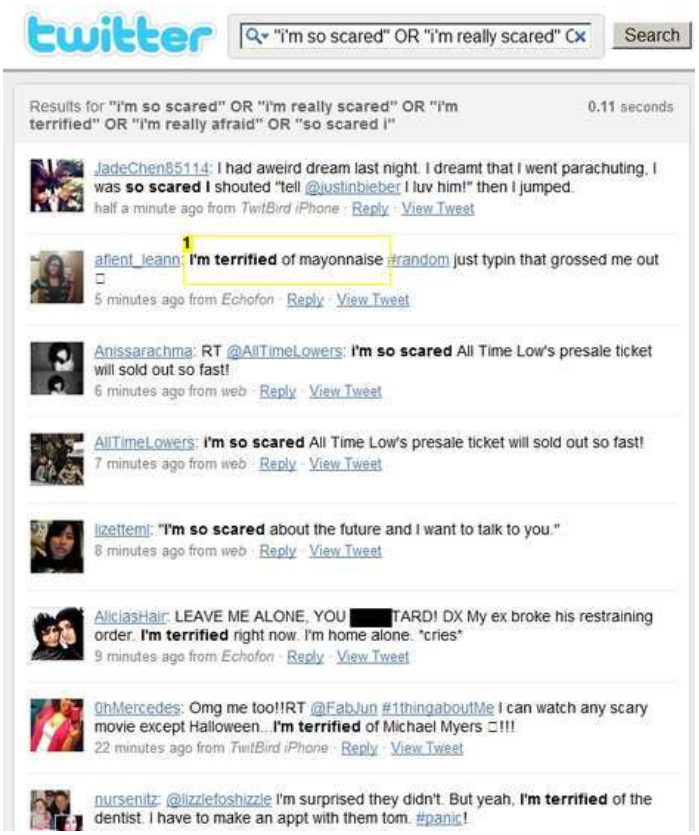
Twitter also requires a final linefeed and carriage return ("\r\n")

"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;
}



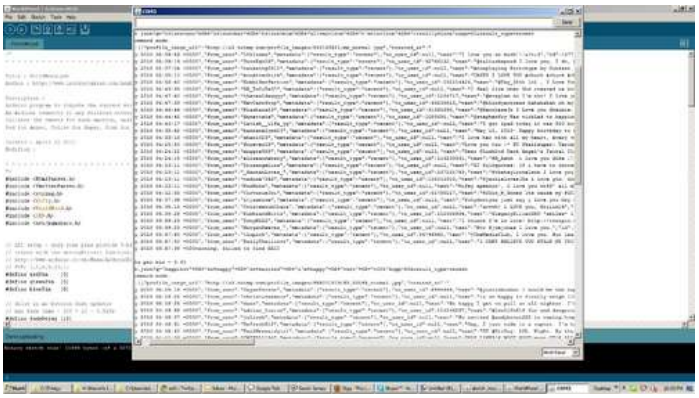


Image Notes
1. perfect

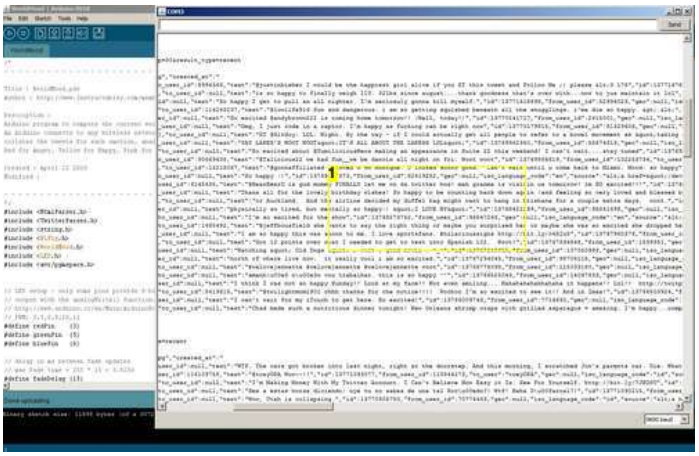


Image Notes
1. too many tweets...

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.

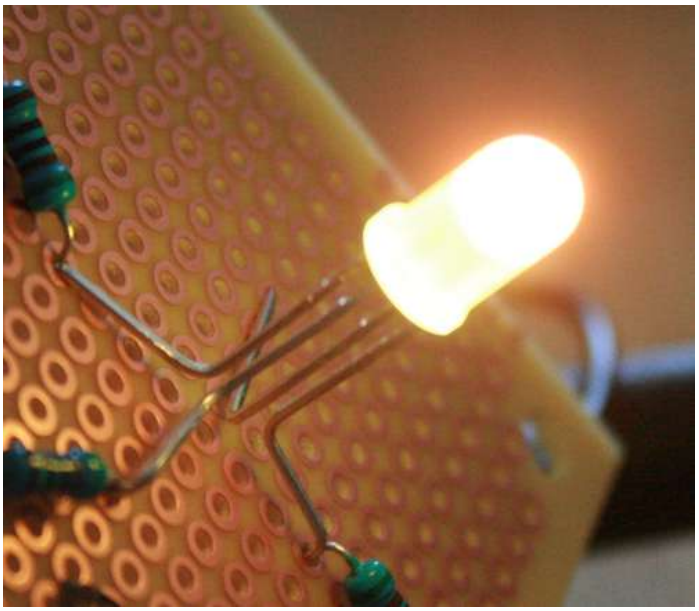
*** update ***

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?

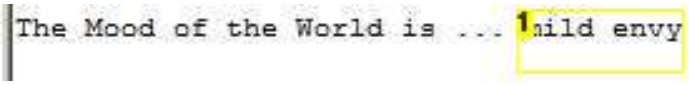


Image Notes
1. ...and so the LED turns green.

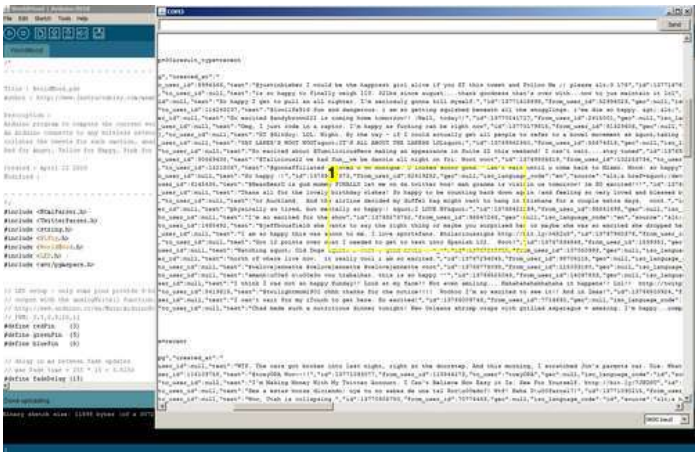
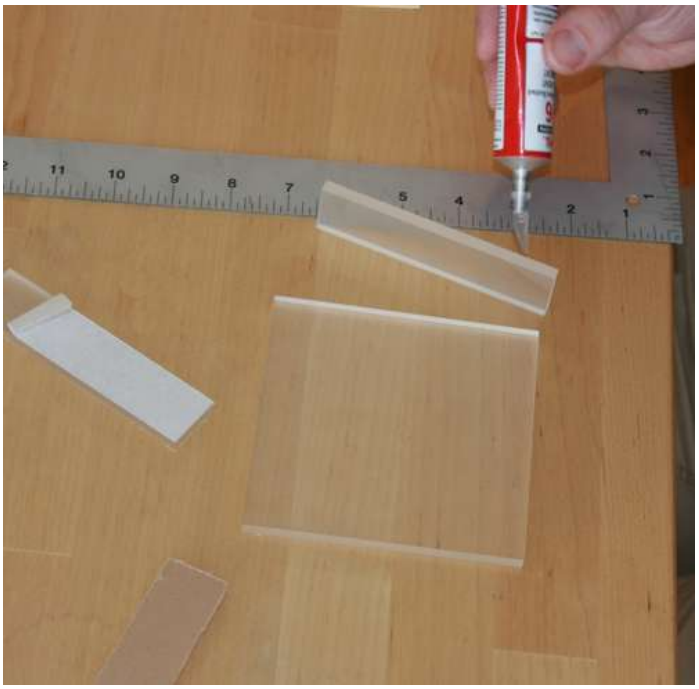


Image Notes
1. too many tweets...

Step 12: Building the Box

Build an acrylic box ala this Instructable:

www.instructables.com/id/LED-Cube-Night-Light/



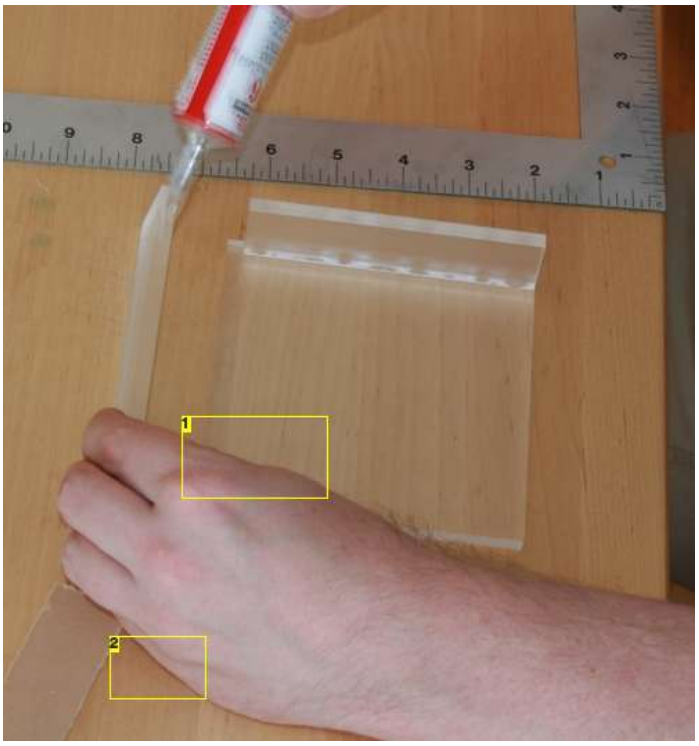


Image Notes
1. I do have a thumb
2. and pinkie

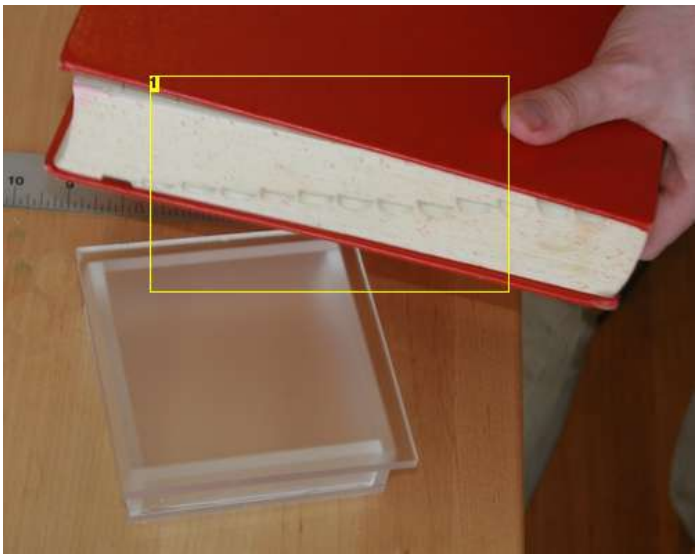


Image Notes
1. put a big book on it and leave it to dry

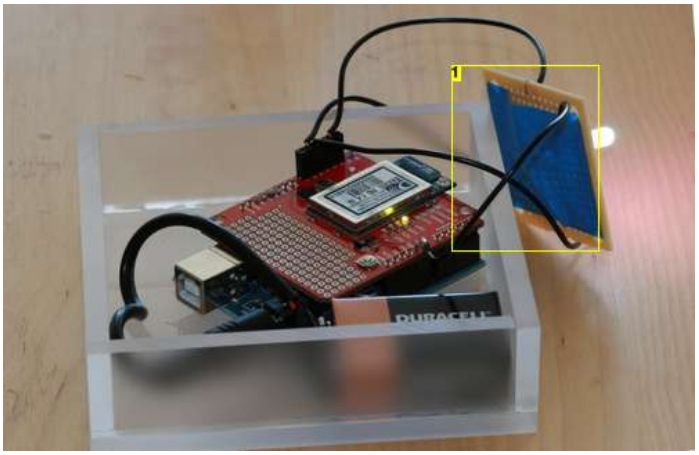
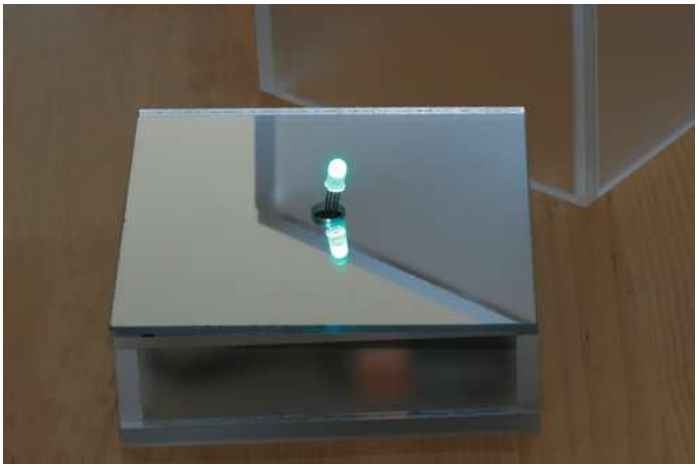


Image Notes
1. the tape helps prevent shorts



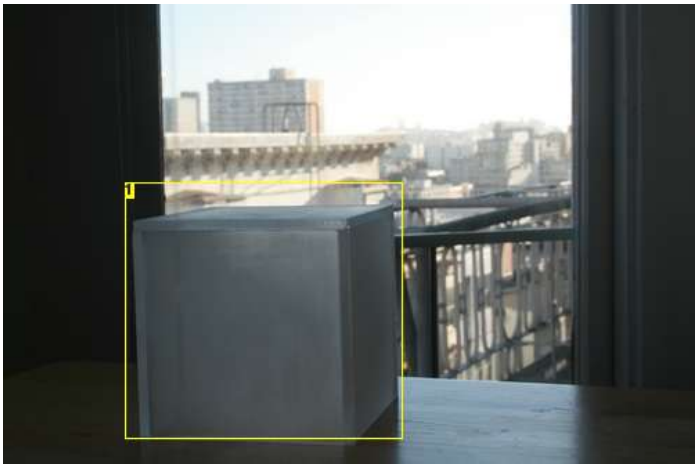


Image Notes
1. sanded to help diffuse the LED

Step 13: Enjoy!

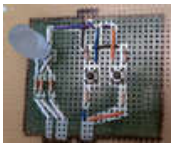
Some possible extensions include:

- 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

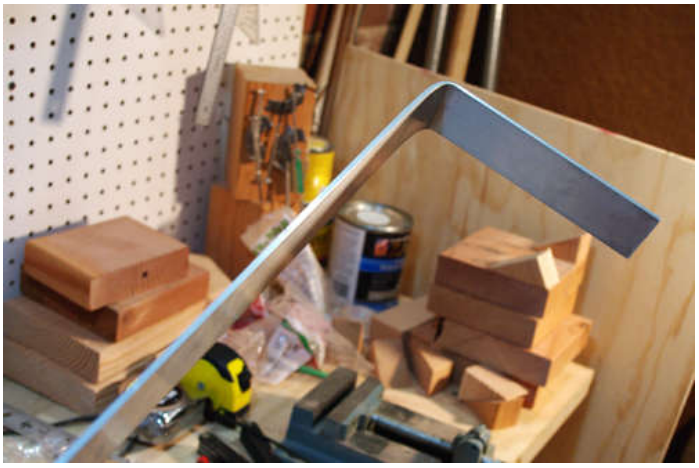
Cut out the pumpkin's face. For the larger and more complicated shapes like the mouth, it help to cut it out in smaller pieces instead of trying to remove one large chunk from the pumpkin.

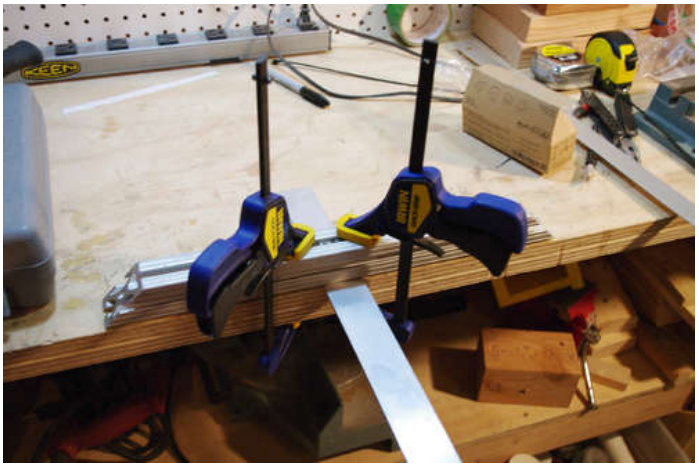
Step 7: Bend

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.



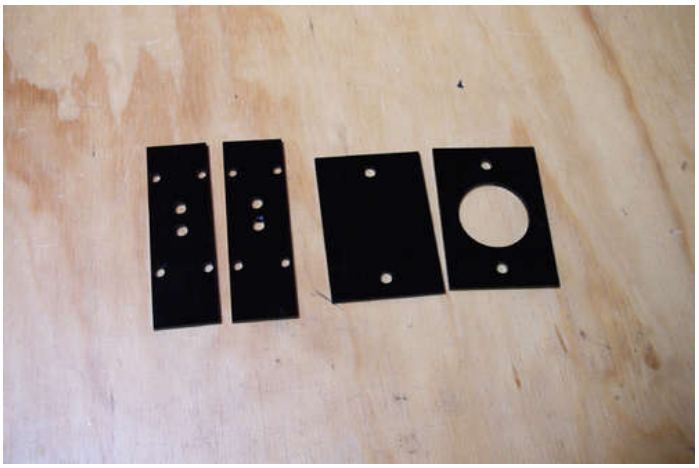


Step 8: Brackets

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']

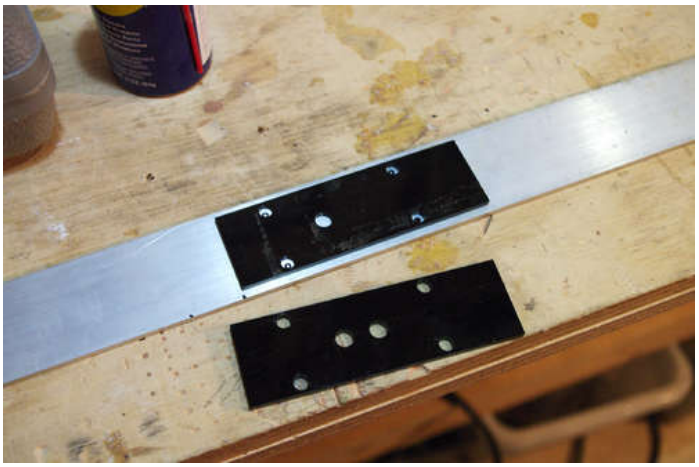
Step 9: Drill holes

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.

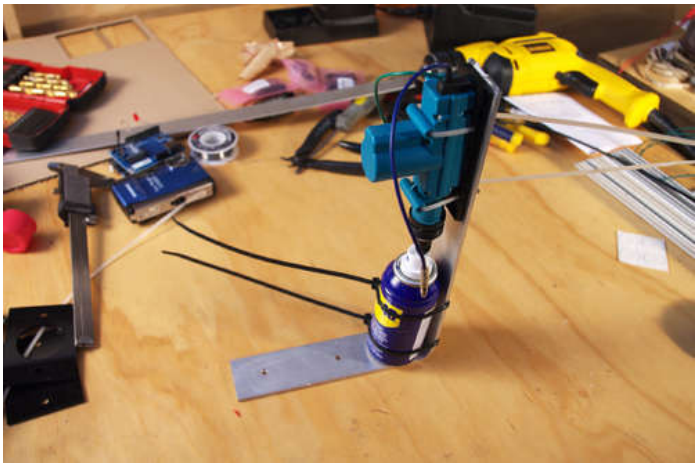
When you are done, drill 1/4" holes through the aluminum using a drill press.

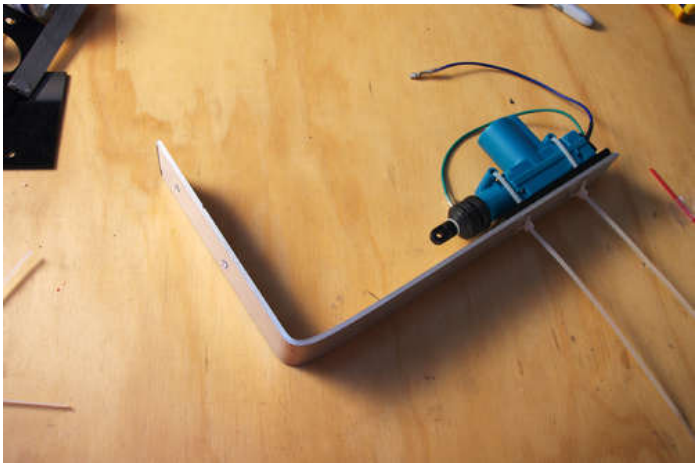


Step 10: Attach things

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.





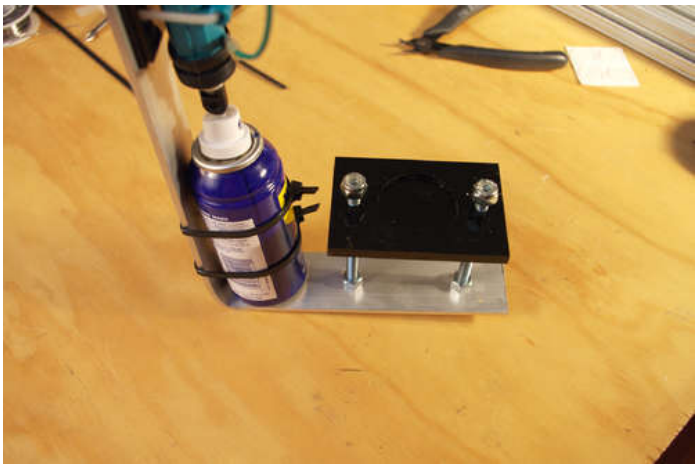
Step 11: Candle mount

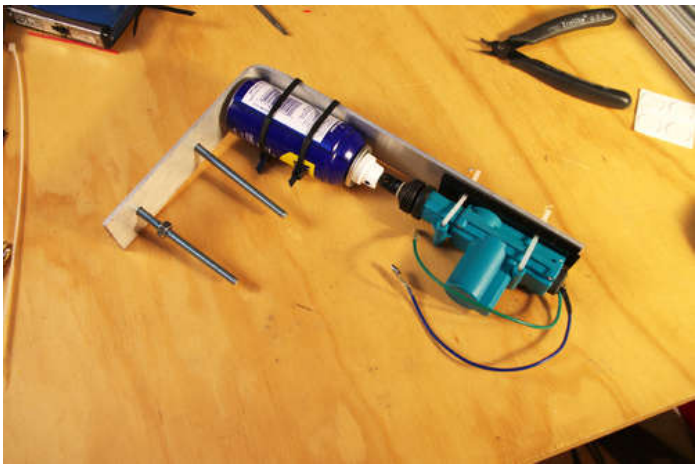
Insert the two bolts upwards through the bottom of the aluminum bracket. Fasten them in place with bolts.

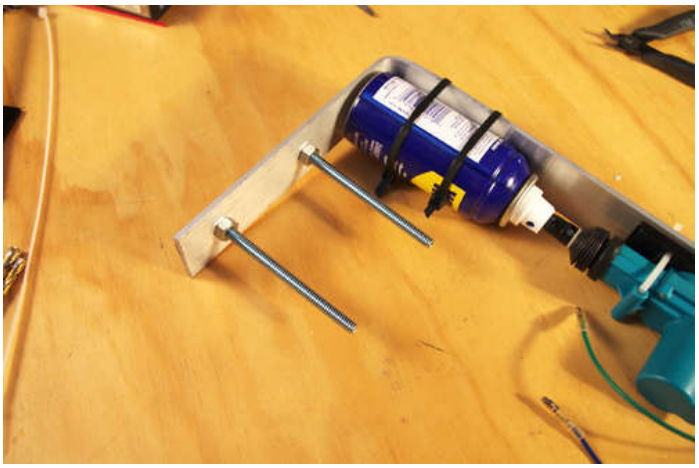
Thread on another bolts onto each. Twist this about 3/4" down.

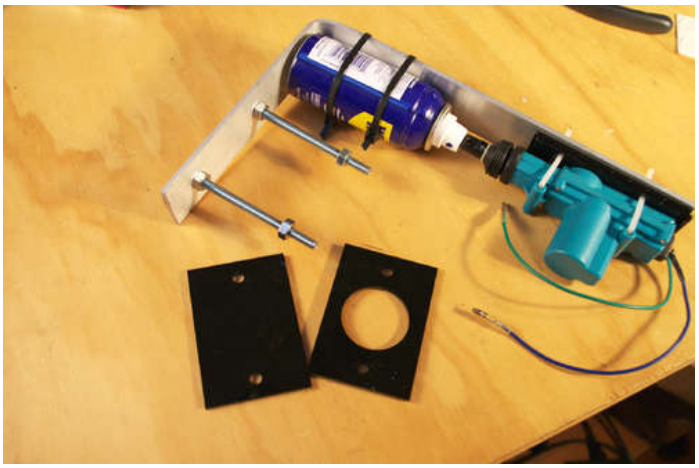
Place the bottom of the candle holder (the side without the large hole) onto the bolts. Then place the top candle holder bracket.

Fasten the whole thing in place by threading on another nut onto each.





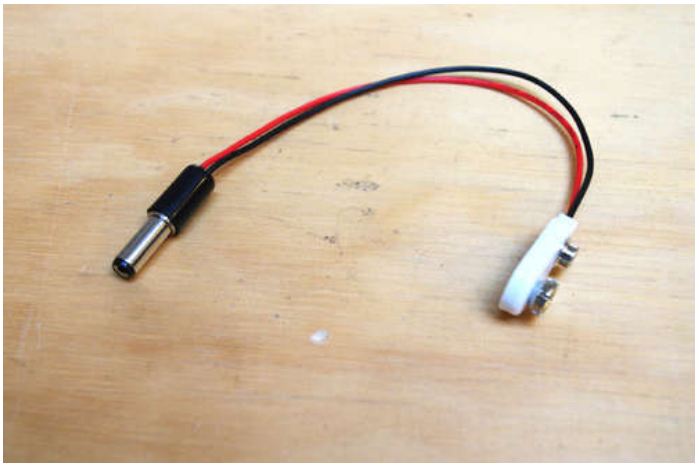




Step 12: Battery adapter

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

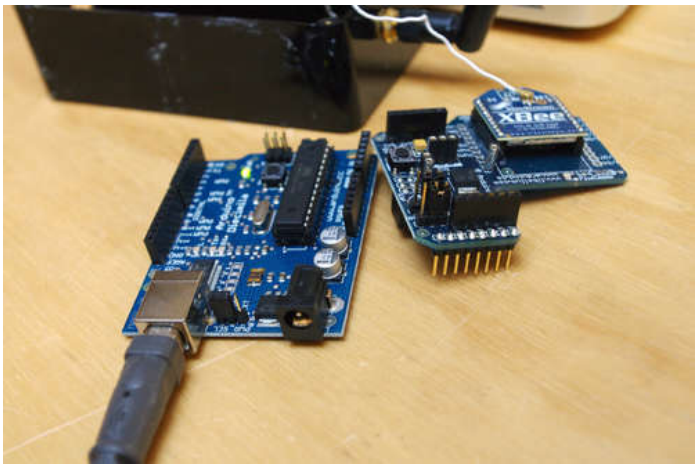
Open the SquidBee transmitter node and remove the Arduino from the XBee shield.

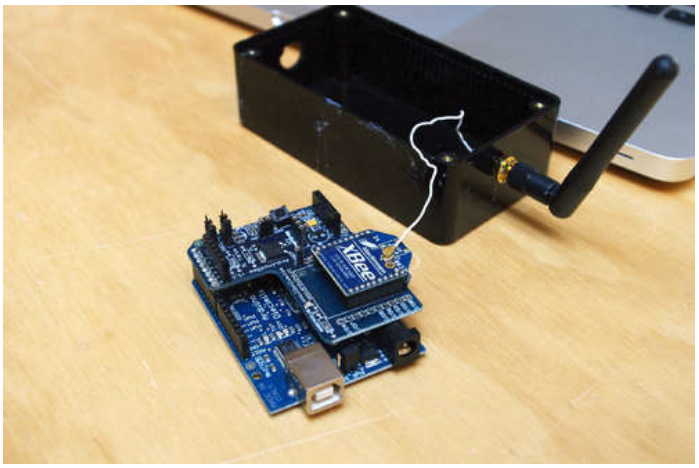
Change the power jumper on the Arduino to select USB power (if necessary).

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.

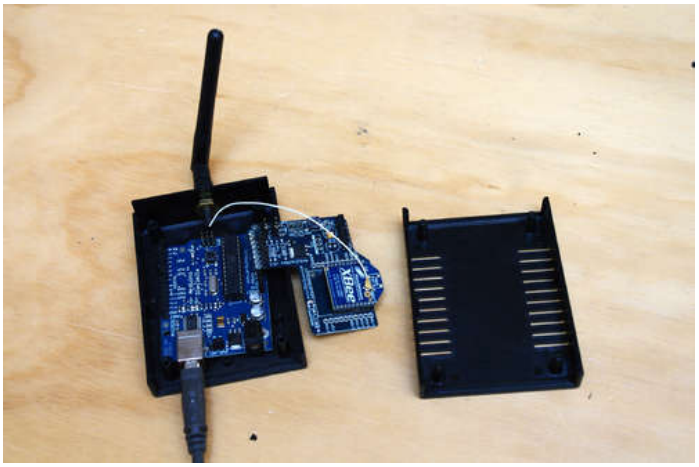
Then, like the other board, change the power selection jumper to USB, and then upload the following code:

```
/*

 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.



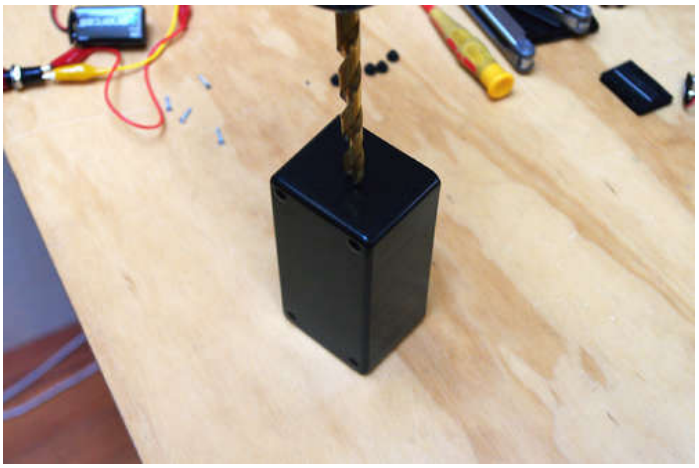
File Downloads

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

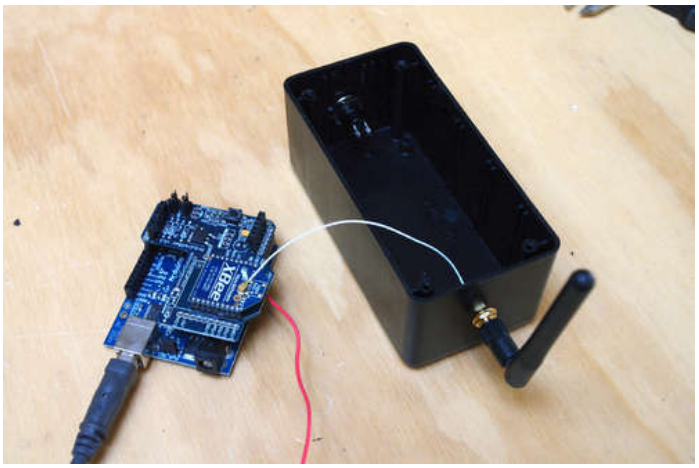
Drill a 3/8" hole (or whatever is appropriate for your switch) in the side of your project enclosure.

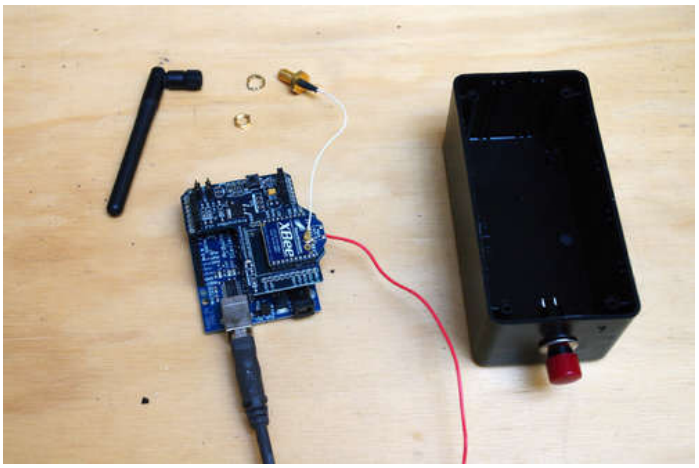
Install the pushbutton switch.

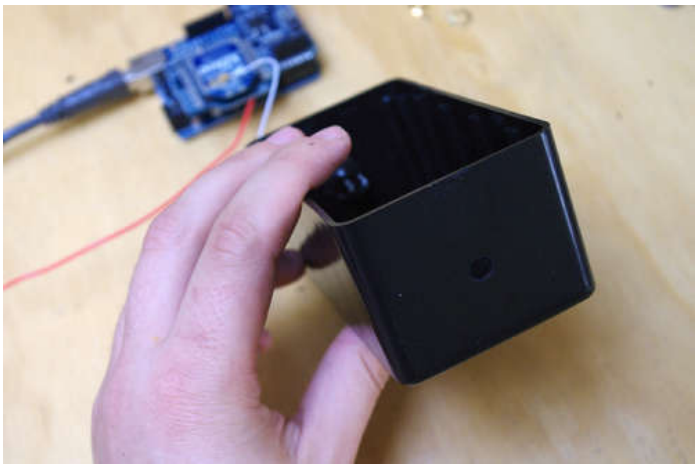


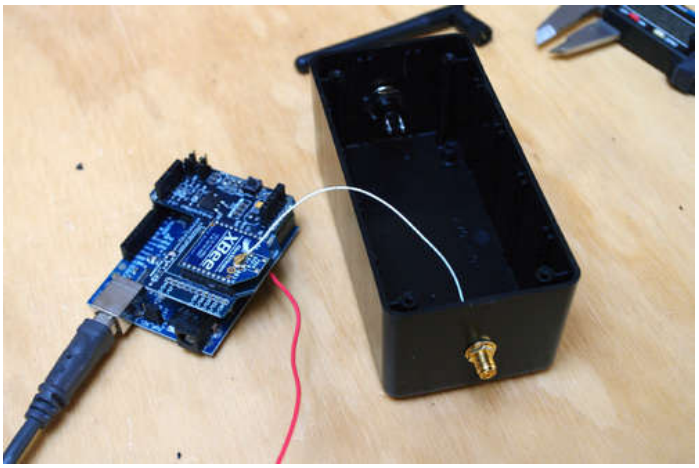
Step 16: Antenna

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.









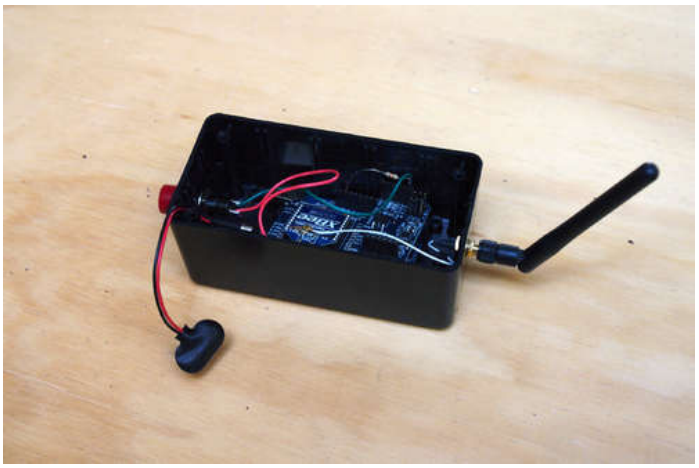
Step 17: Wire the transmitter

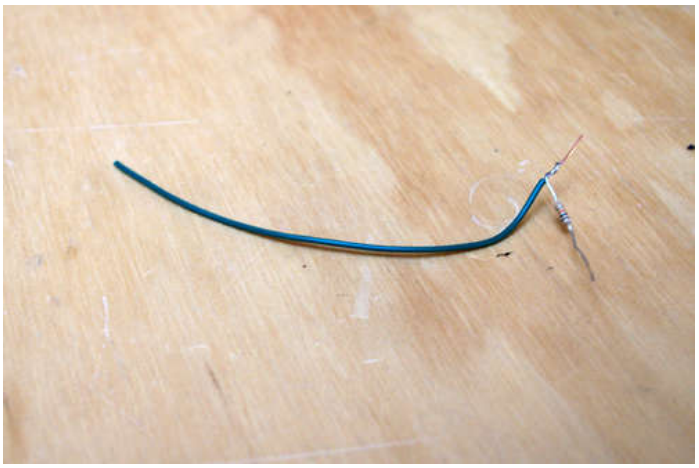
Solder a wire to one leg of a 10K resistor. Solder the opposite end of this wire to one of the switch terminals.

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.

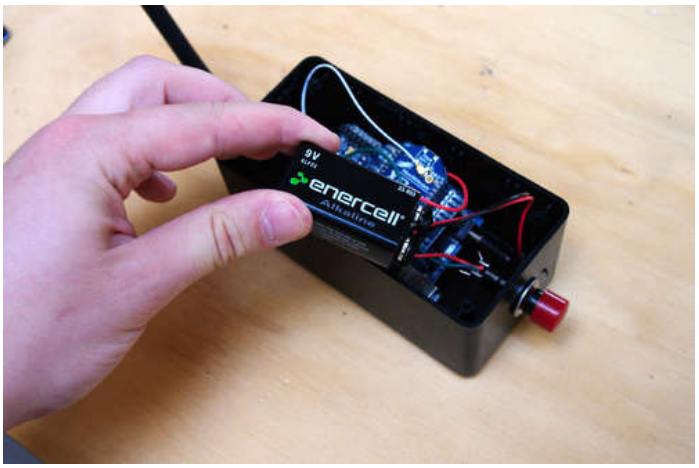
Lastly, plug your 9V battery connector into the power socket on the Arduino board.





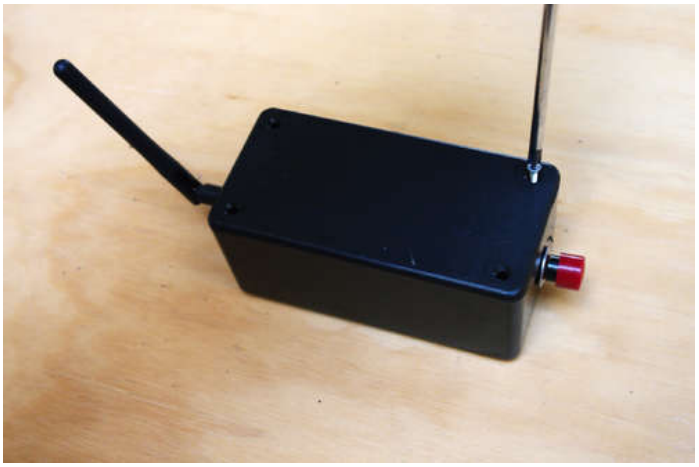
Step 18: Power

Plug in a 9V battery to power up the transmitter.



Step 19: Case closed

Fasten shut the transmitter's case.



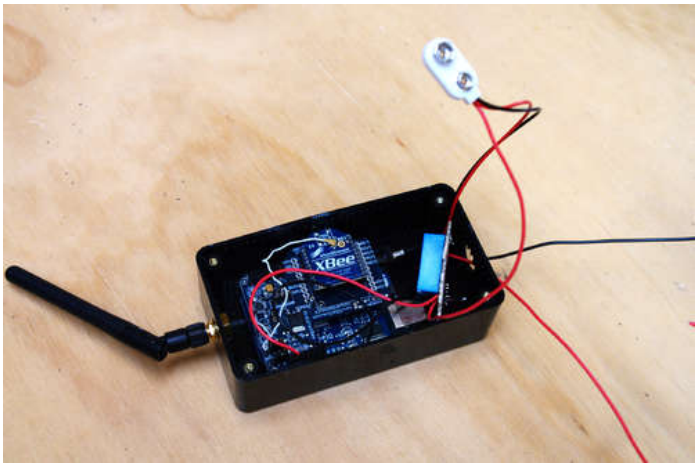
Step 20: Wire the reciever

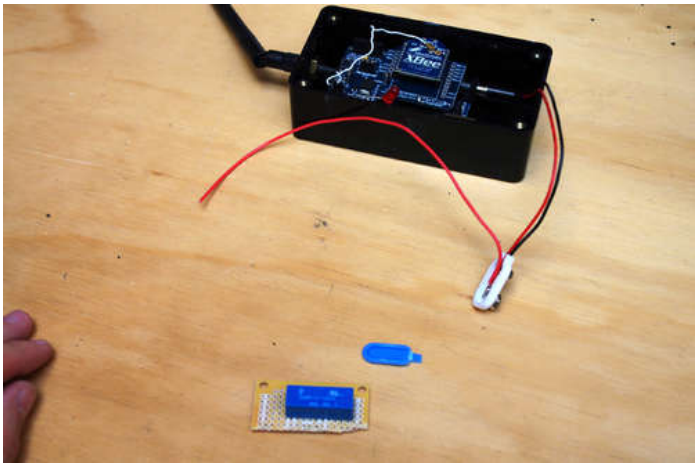
Affix the relay to a small piece of perfboard.

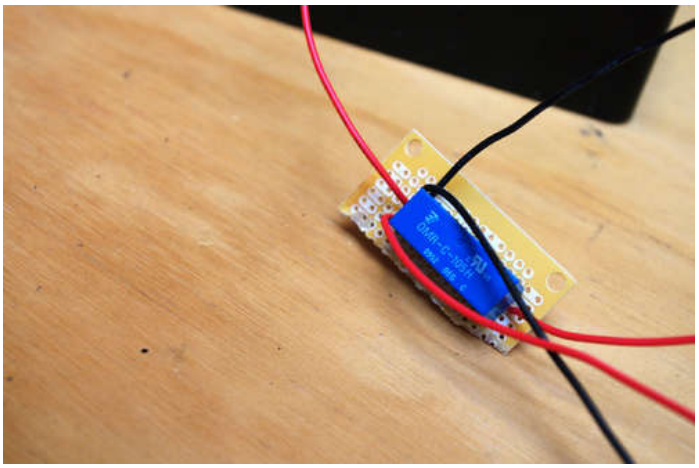
Connect one of the relay's coils to ground on the Arduino board and the other to pin 3.

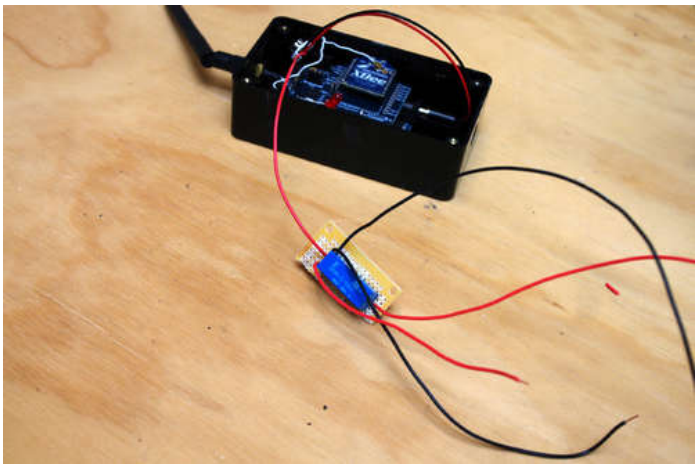
Attach 9V to one of the relay's load pins and a long red wire to the other. To get easy access to the 9V power source, I broke off the top of the 9V battery snap and soldered a wire directly to the +9V battery connector tab (notice the extra red wire coming from the 9V battery snap).

Attach an extra long black wire to ground.







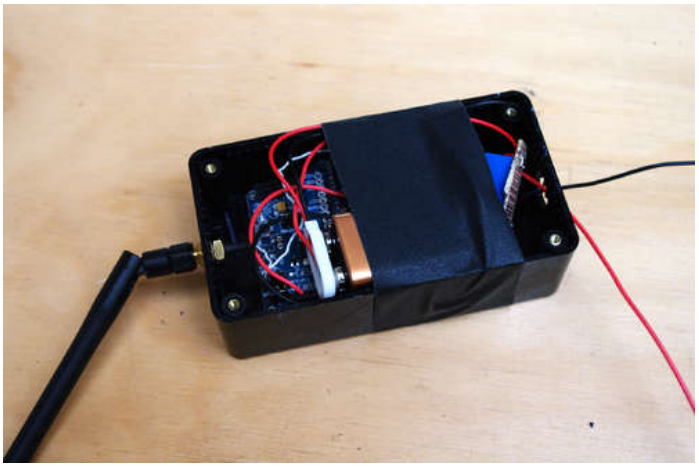


Step 21: Put it together

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.

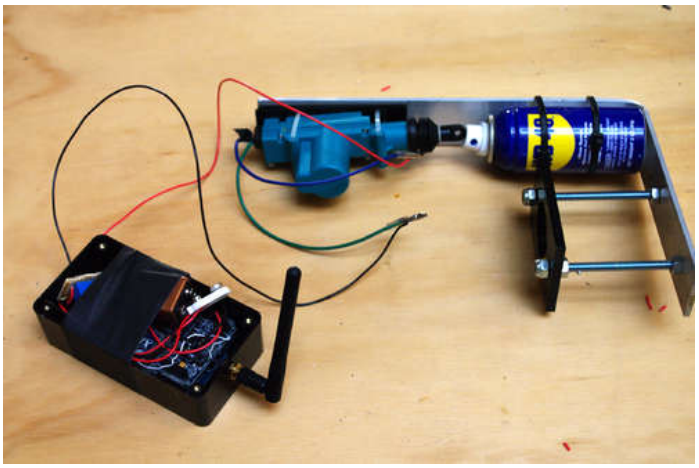
I plugged in the 9V battery and passed the red and black wires through the hole in the side of the case neat-like.

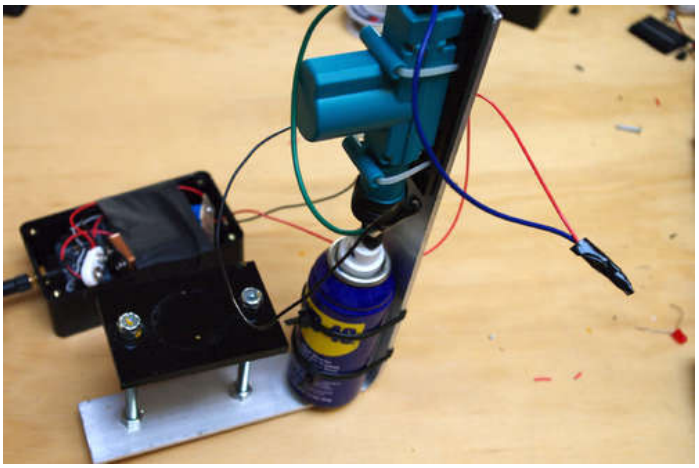
Then, I slapped a piece of black tape on top and called it a day. This will be inside the pumpkin, so aesthetics don't matter quite as much.



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.





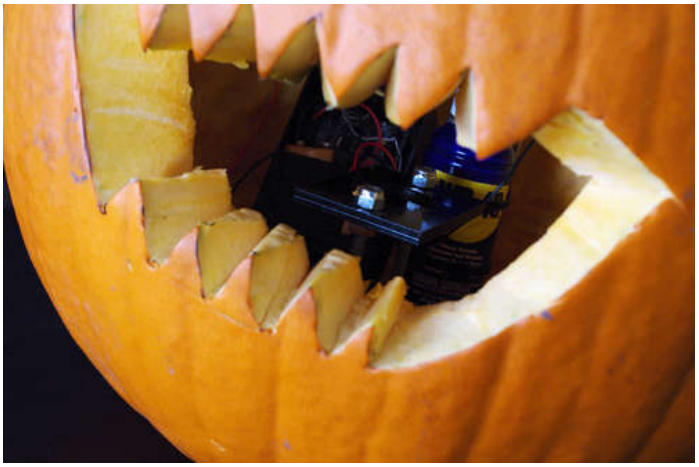
Step 23: Put it in the pumpkin

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

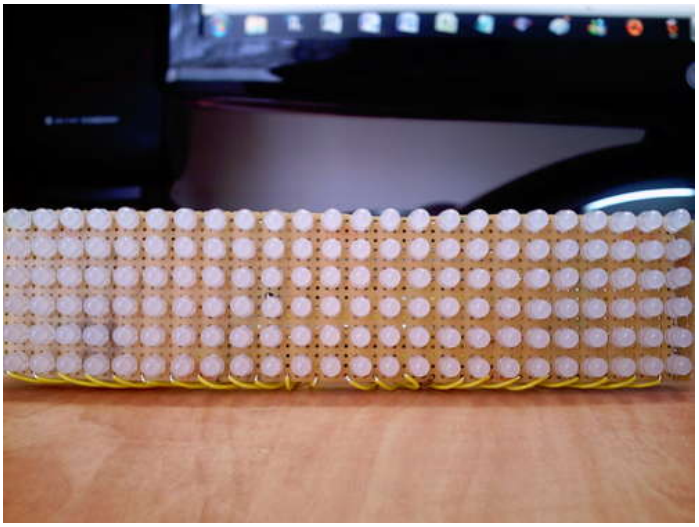
by **Syst3mX** on July 21, 2010

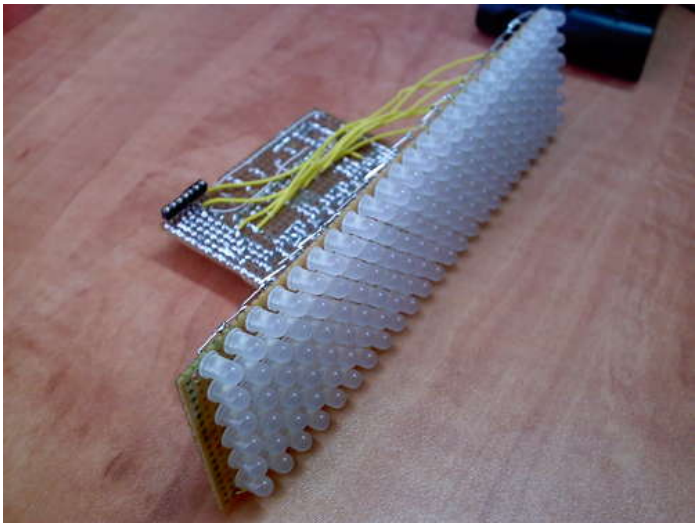
Author:Syst3mX Vadim
Electronics and LEDs what can be better ?! :D

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.

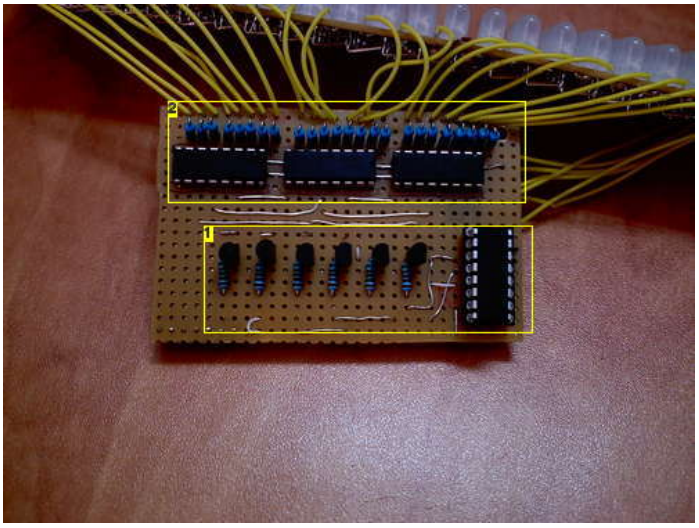


Image Notes
1. 4017 With 6 transistors for scaning the rows
2. The 3 shift registers to control the rows

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!

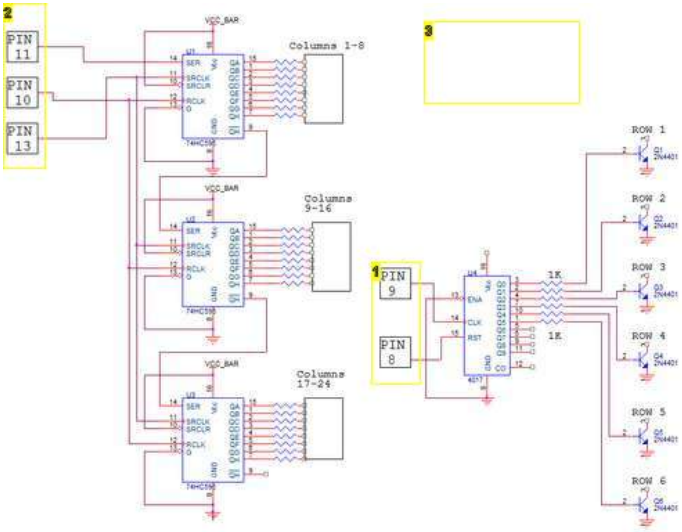


Image Notes
1. Arduino Pins
2. Arduino pins
3. Download the original file to see better.(press the "i" icon in the top left corner of the picture)

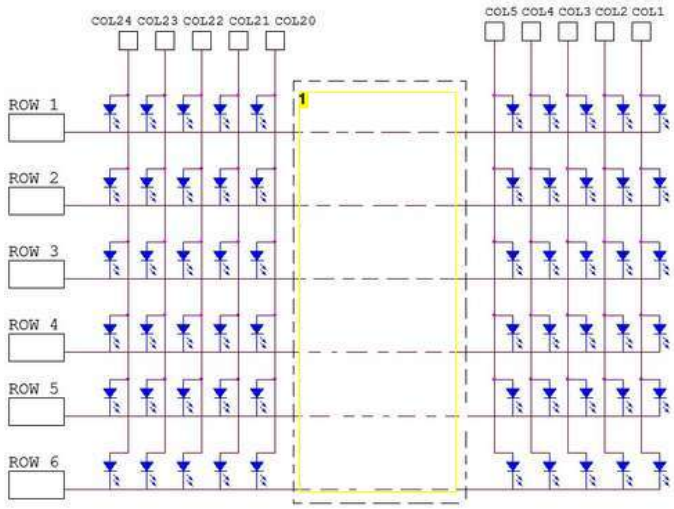


Image Notes
1. I didn't have room for the full matrix, So I hope you get the big idea

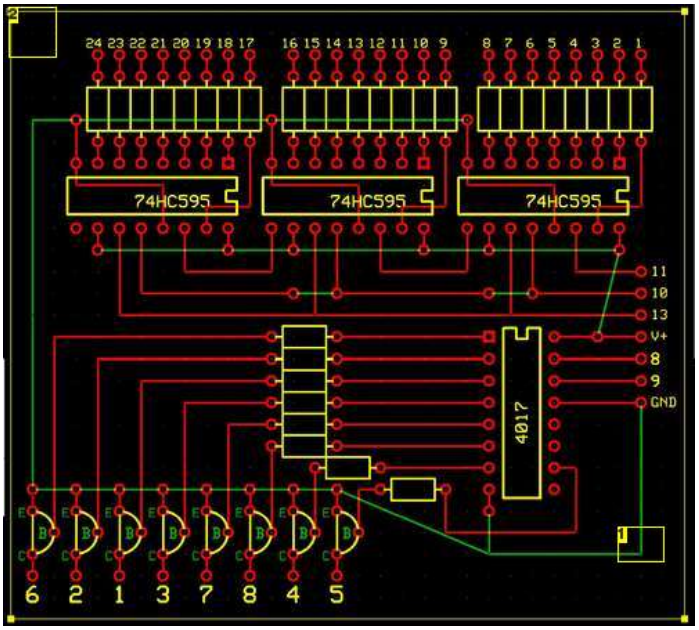
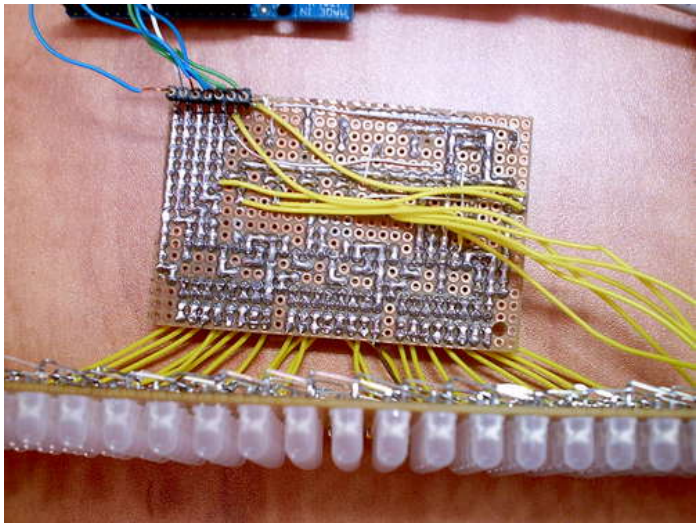


Image Notes
1. Green line are jumpers and red lines are copper traces.
2. PCB layout made by Willard2.0

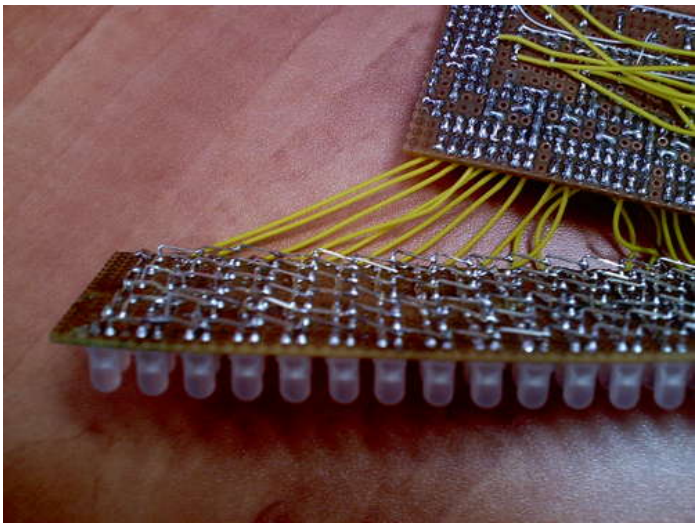


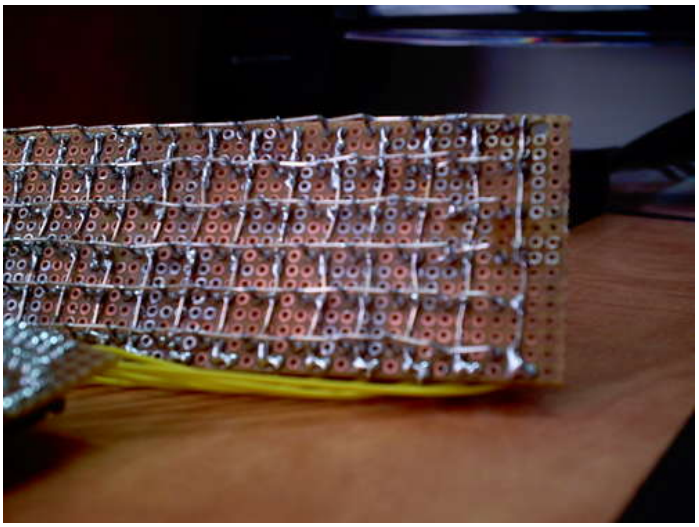
Step 4: Soldering The LEDs

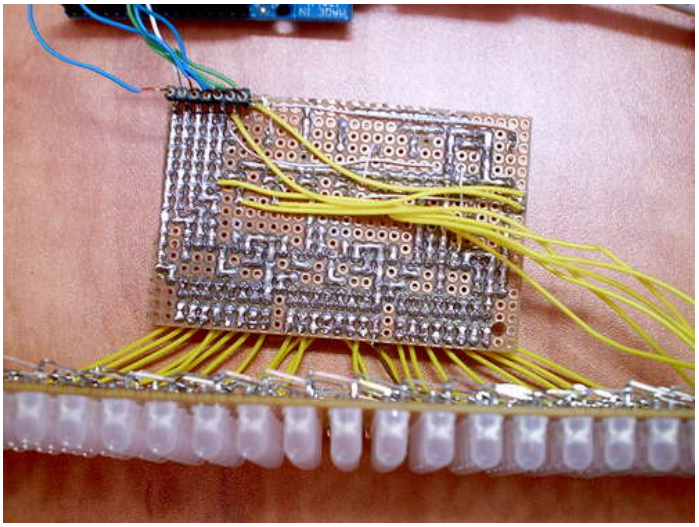
Soldering 144 LEDs in a matrix formation can be a little tricky if you don't have a general idea how.
The last time I soldered a matrix I used lots of little wire jumpers which was a pain to solder, so this time I was a little more creative and came up with this way.

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.





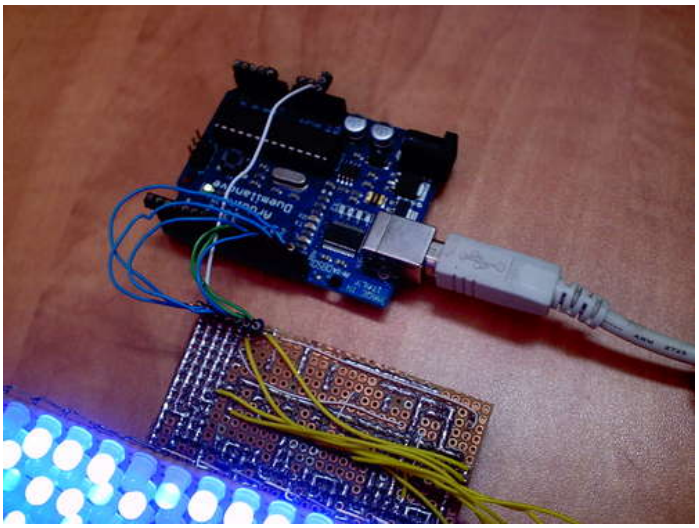


Step 5: Programming The Display

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

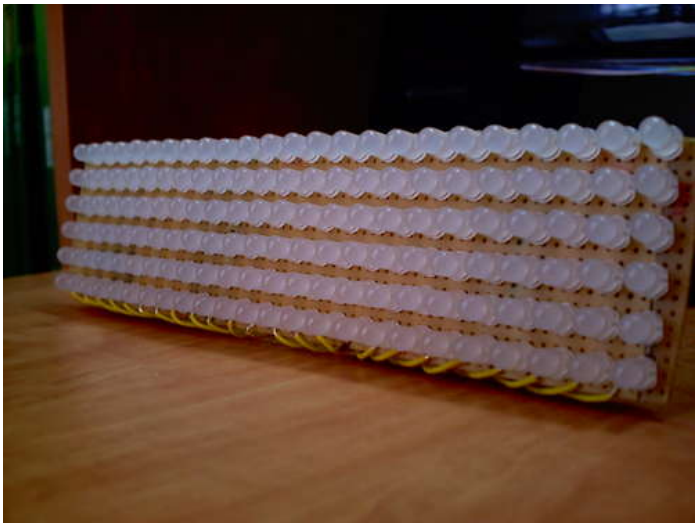
Looping text.txt (9 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Looping text.txt']

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.



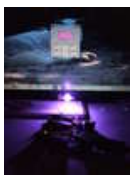
Related Instructables

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:

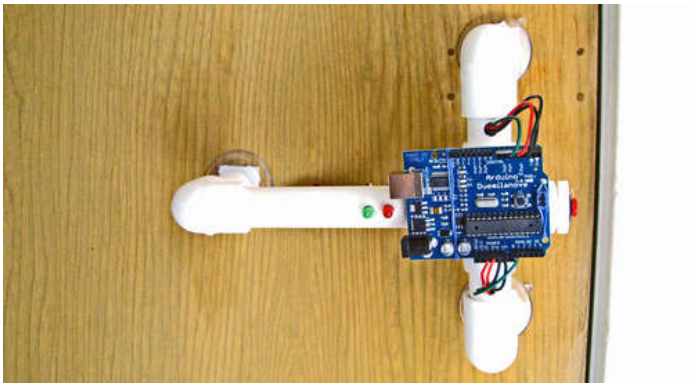
Important Notes:
(I hate to even have to say this, but since someone's going to say it, I'll say it first:)
1) This is for entertainment purposes only. Really. This decreases the security of your door by adding another way to unlock it, and it makes your unlock code known to anyone who can hear. If you 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.

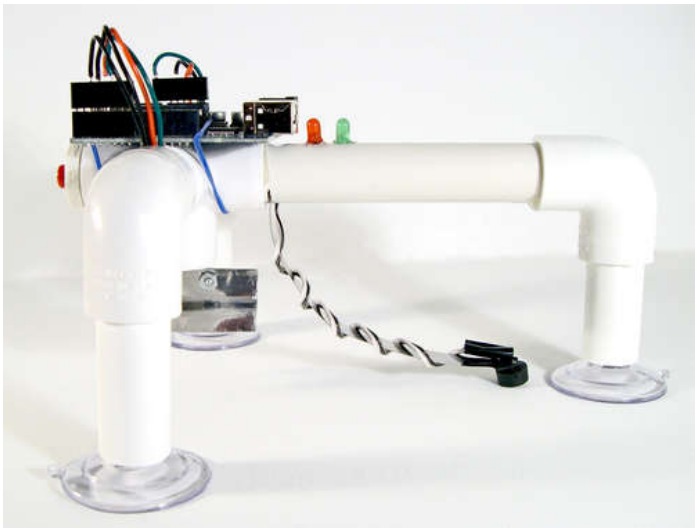
For obvious improvements to safety, security and whatever, see the final page of the Instructable.

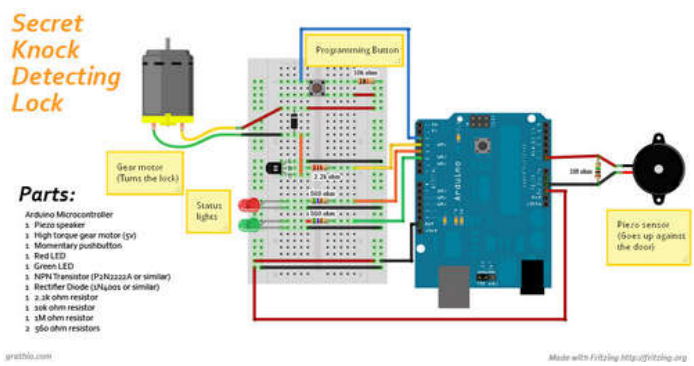
2) This is not a project for a beginner! Read through it carefully and be sure you understand it **before** you start! I will not take time to answer questions that are already in the instructions or from people who have gotten in over their head.

(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.

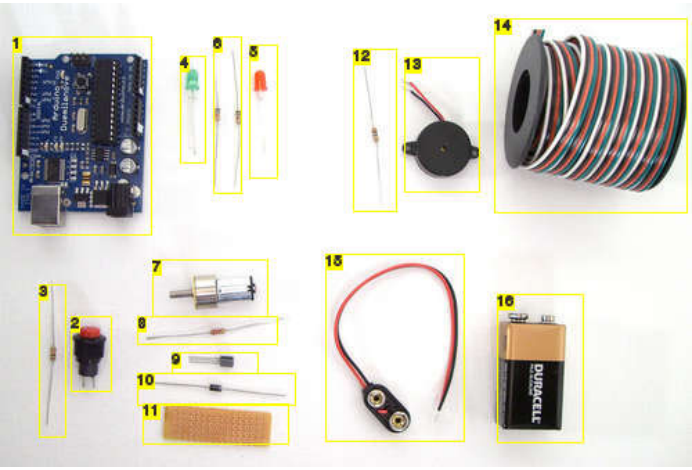


Image Notes
1. Arduino microcontroller.
2. Momentary pushbutton
3. 10K ohm resistor (brown, black, orange)
4. Green LED
5. Red LED
6. 560 ohm resistors. (Green Blue Brown)
7. Gear motor
8. 2.2K ohm resistor (red red red)
9. 2n2222 Transistor (NPN type)
10. Rectifier diode (1N4001)
11. Perf board 5x15 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.

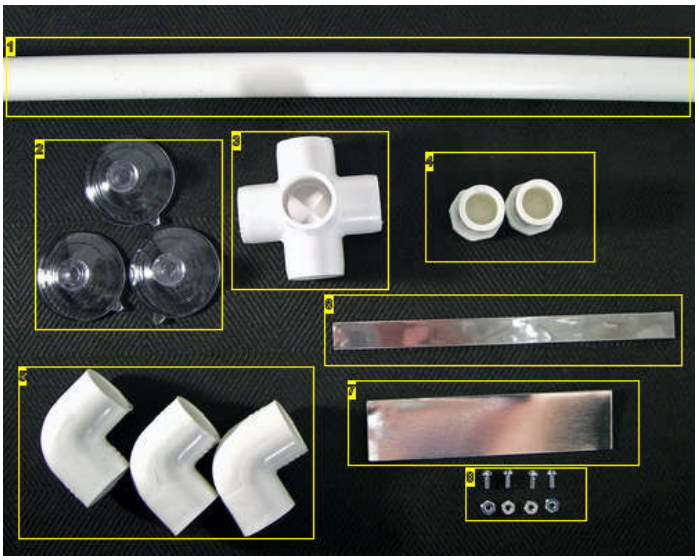


Image Notes
1. 20 inches of PVC, 1/2"
2. 1 1/2" diameter suction cups.
3. 5-way PVC connector. (Can be a little tricky to find.)
4. PVC end plugs
5. PVC right angle (90 degree) connectors
6. 1/2" wide metal strip used for a spring to press the detector to the door.
7. This metal strip will be made into 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 :

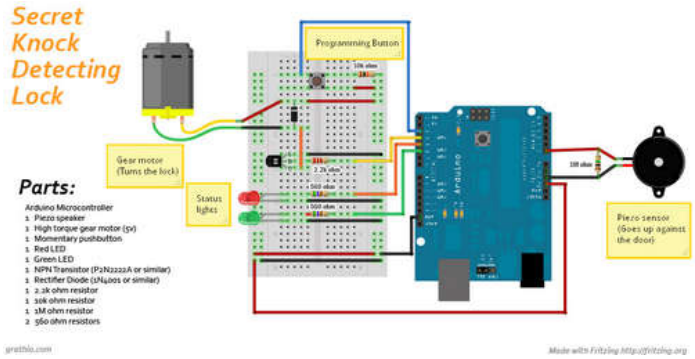
1) If the motor turns very weakly the diode might be reversed.

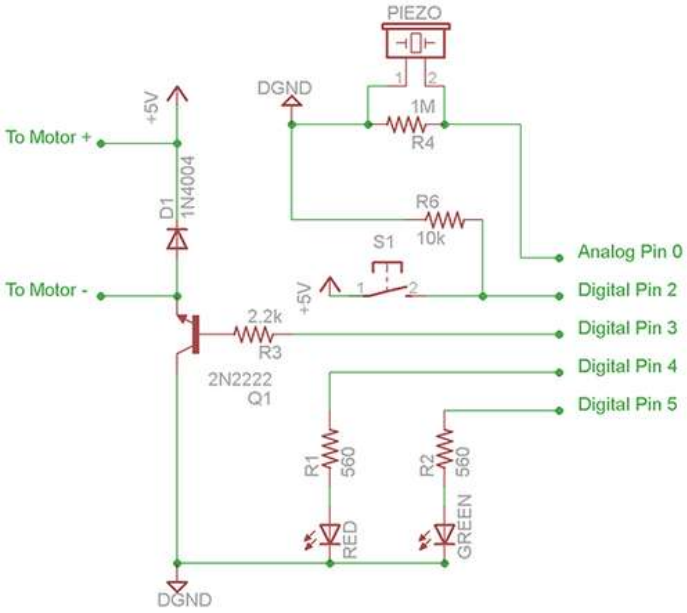
2) If you need more power on your motor make the following circuit change: Move the wire that goes from the motor to +5v to the Vin pin on the Arduino. This will supply the motor with 9v (or whatever voltage you're supplying to it.)

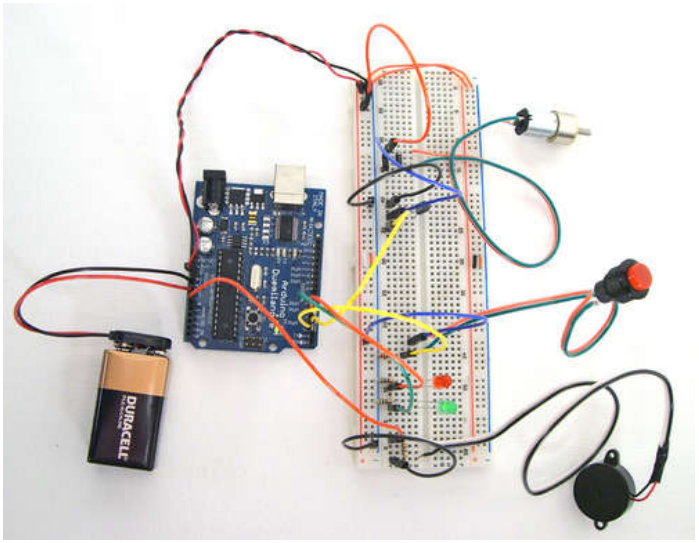
Tip : Check which way the motor turns. It should turn the same way as you turn your deadbolt lock to unlock it. If not, switch the motor's leads which should reverse the motor.

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.

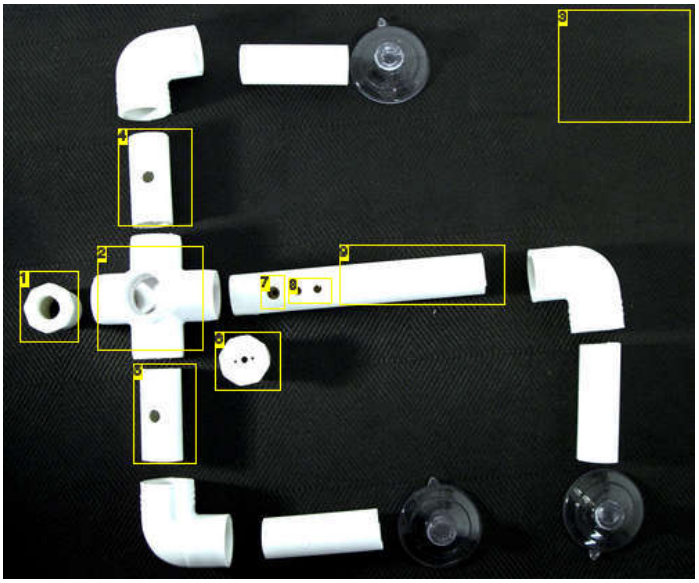


Image Notes
1. Button mount. End plug with hole drilled for the panel mount button.
2. 5-way connector. The motor goes in the center of this.
3. This is an exploded view with all of the parts laid out how they attach (though the angles are wrong on some parts.
4. Top arm. Hole drilled to let wires through.
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.)

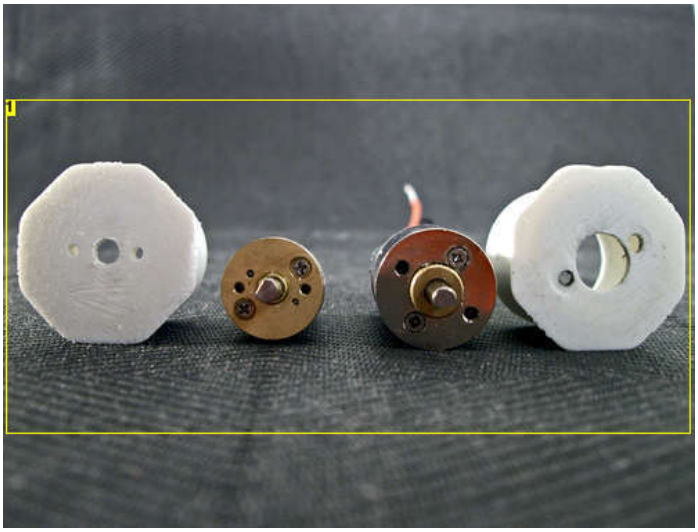


Image Notes
1. Two sample motor mounts for two different motors. The one on the left is smaller but actually more powerful. The one on the right has a larger hole to allow clearance for the bearing. The facing surface on both of these have been ground down so more of the shaft can stick out.

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.

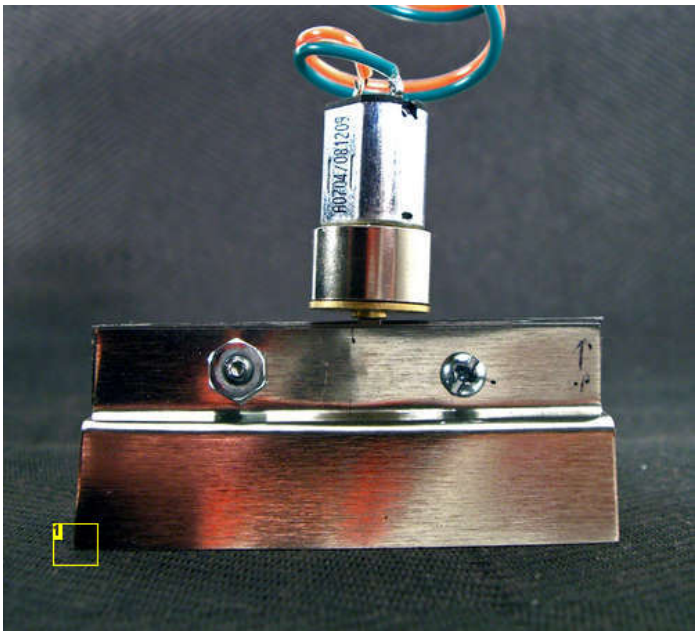


Image Notes
1. It's not a bad idea to file down the sharp corners.

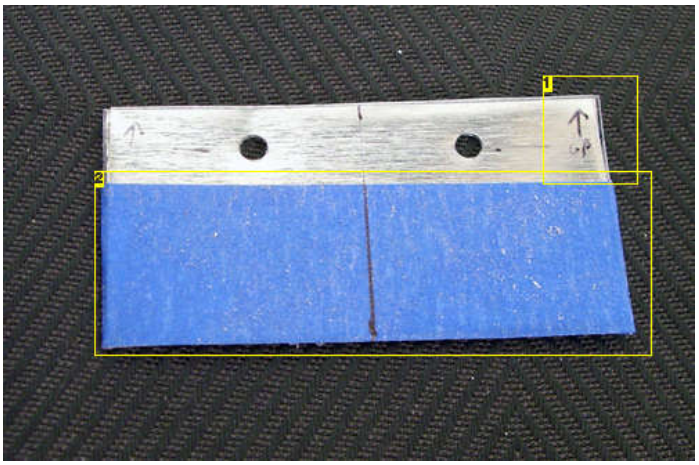


Image Notes
1. A few markings will help keep them straight when you put them back together.
2. Two pieces taped together so we get the holes lined up.

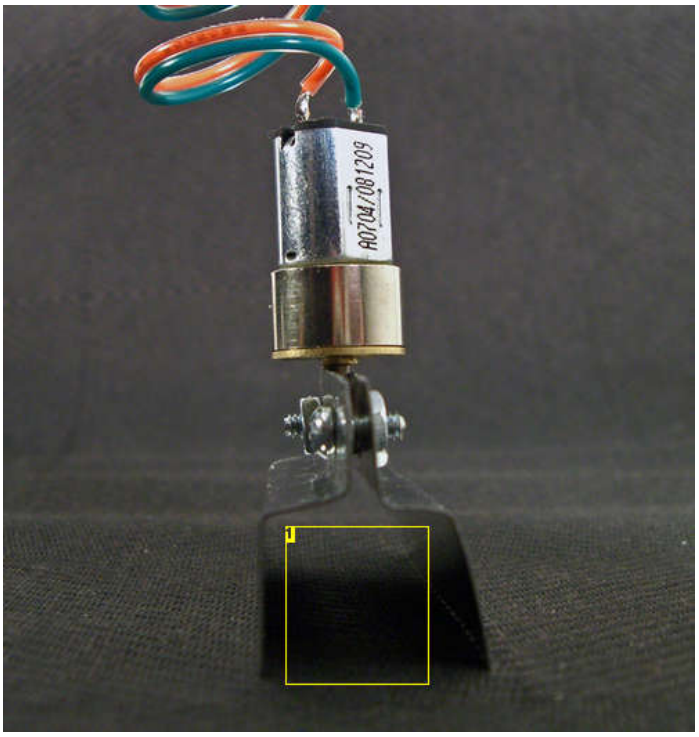


Image Notes
1. Edge view. Your lock will slot in here.

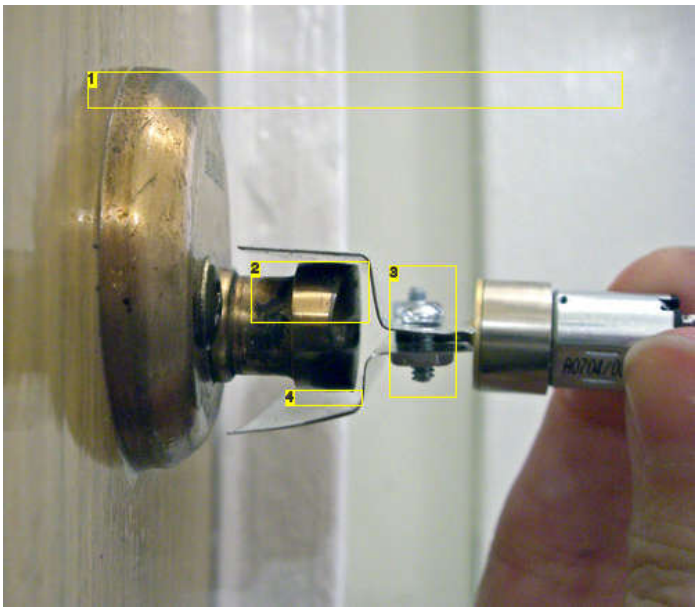


Image Notes
1. In position over the lock.
2. 1/2 of the thickness of the lock handle. This is how much we "zig" each flange.
3. Tighten the screws so they clamp down on the D-shaft.
4. Having a little bit of space is fine and will make it easier to put on.

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.

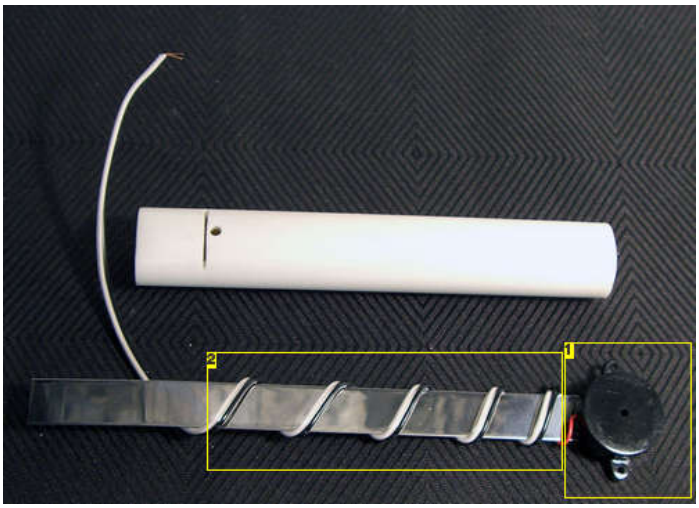


Image Notes
1. Hot glue (or otherwise attach) the sensor to the end of the metal strip. Make

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.

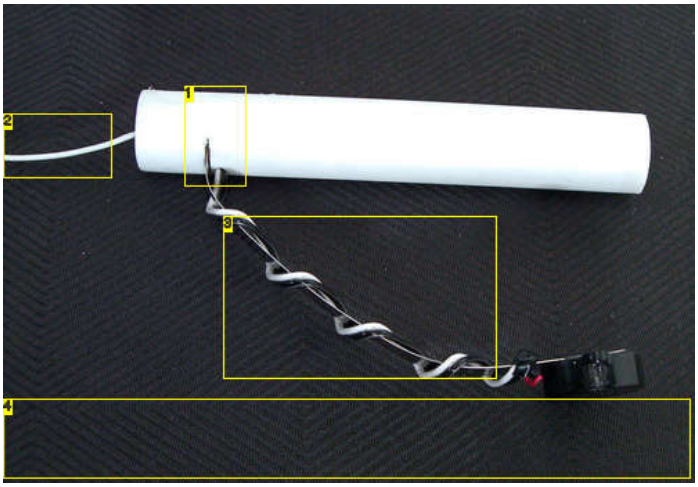


Image Notes
1. Thread the wire through the round hole and slide the free end of the metal strip into the slot.
2. The free end of our wires.
3. Bend the strip more or less like this.
4. This area is where the door will be, Try to bend it so the sensor lies more or less flat against the door.

sure the "open" end is facing out.
2. Wrap some of the remaining wire around the strap to keep it out of the way.

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.

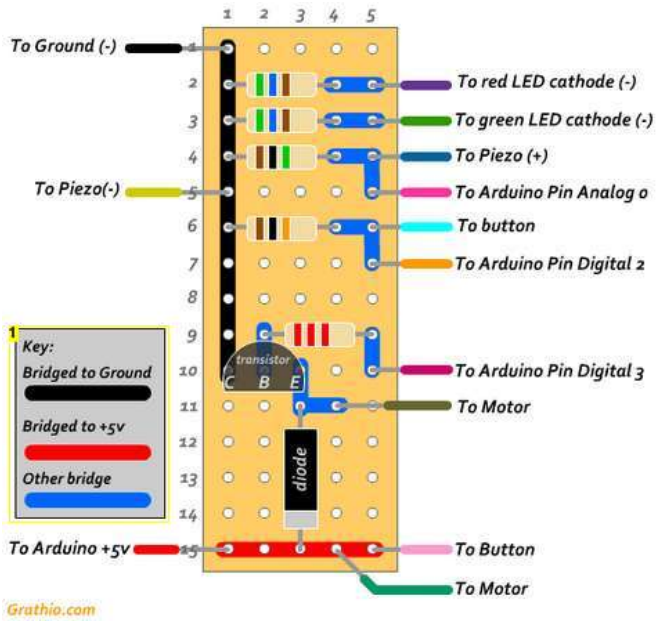


Image Notes
1. Check the pin order on your transistor. Some times they're reversed.

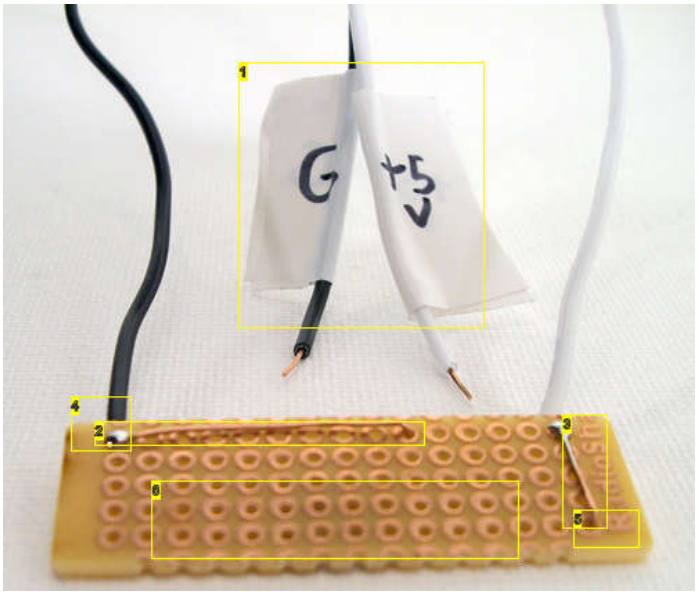


Image Notes
1. Tape labels are a good idea for this project if you're making the PVC pipe case. Misrouting wires is a real possibility.
2. The ground wire. We'll tack other components to this as we add them to the board.
3. The +5v wire. We'll tack other wires to this as we add them to the board.
4. 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.

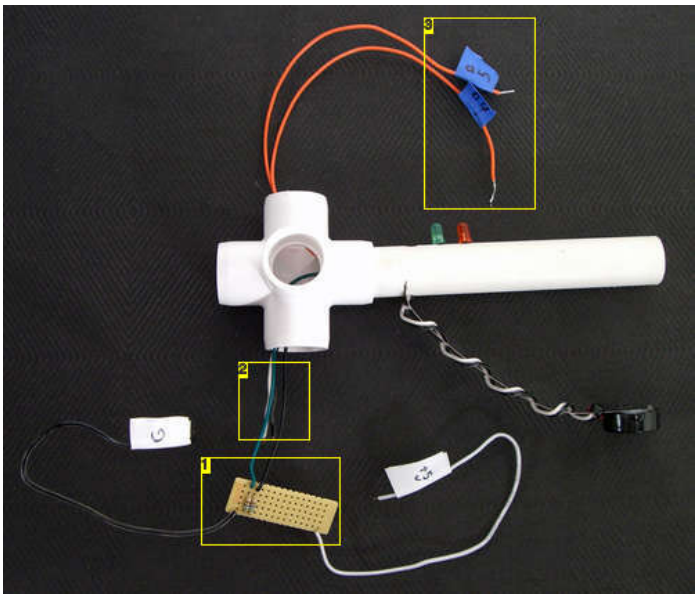


Image Notes
1. #5: The solder the resistors in place to the ground line, and the short leads of the LEDs to the resistors. Pass the leads through the Board Arm hole of the 5-way.
2. While we're threading wires we have threaded both the Piezo sensor wires through for the next step.
3. The + leads for the LEDs come out up here. And we mark them with tape so we don't confuse them later.

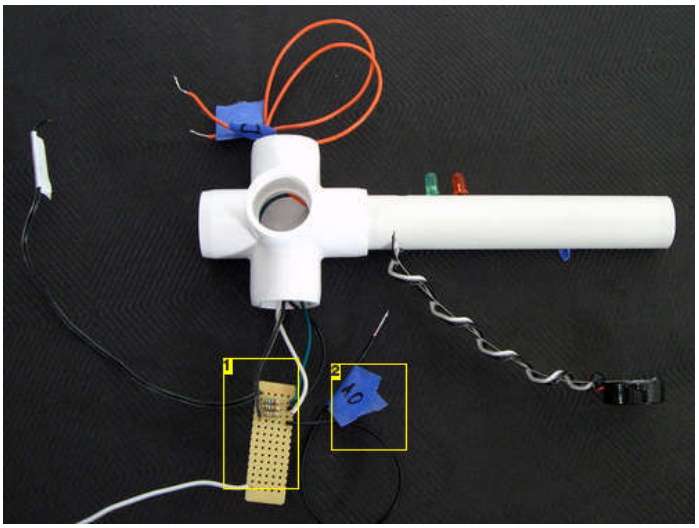


Image Notes
1. The1M resistor soldered in with the piezo sensor attached either side.
2. A lead attached that will go to Analog 0. (We don't need to thread this anywhere for right now.

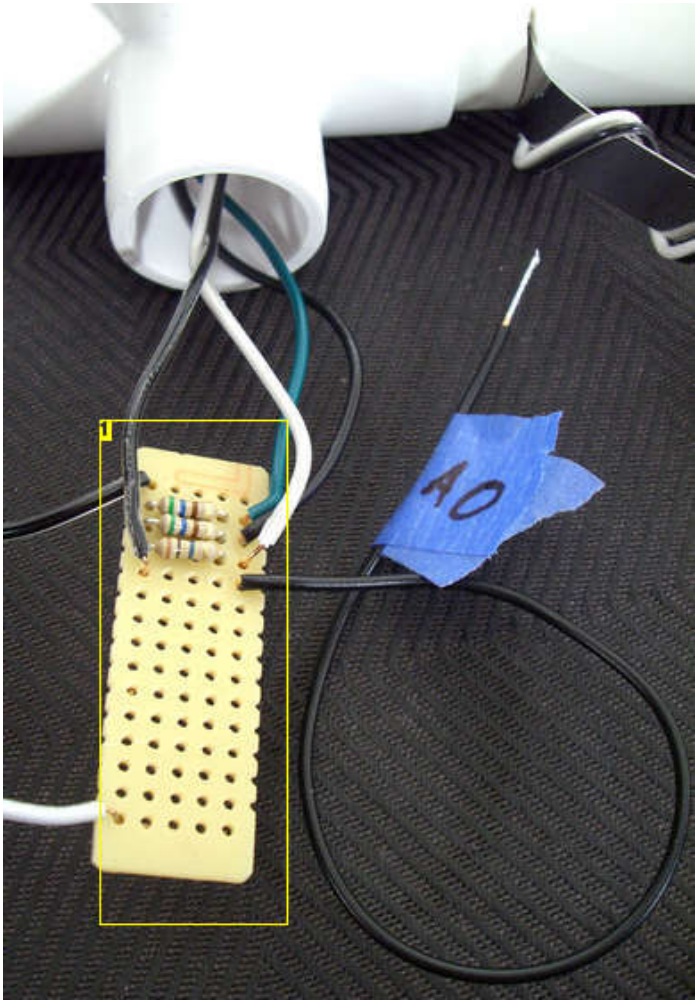
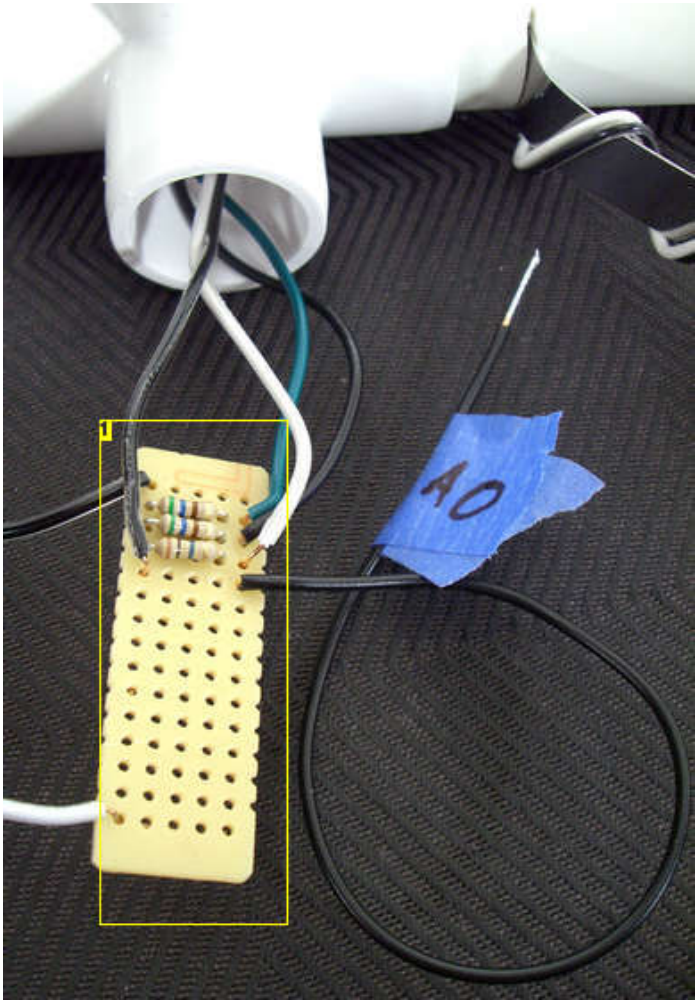


Image Notes
1. Closer look at the resistors and their connections. This is what it should look like after #7, above.



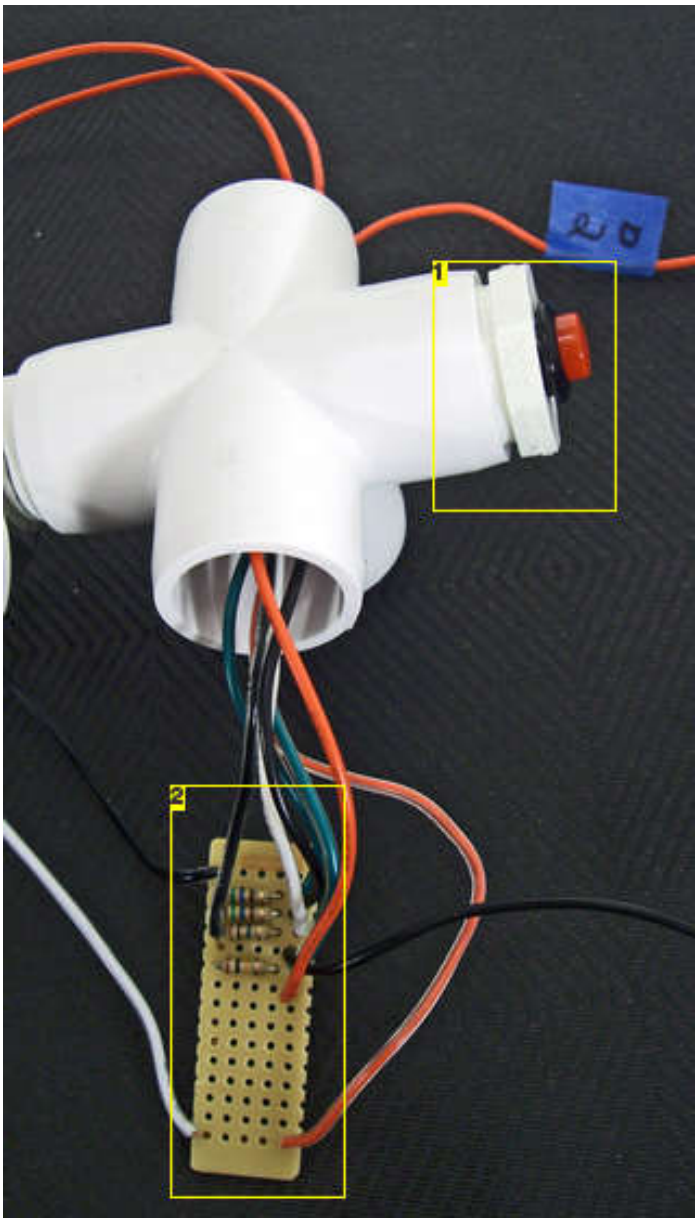


Image Notes
1. The button in place with the wires fed through the 5-way.
2. The resistor and button connections.

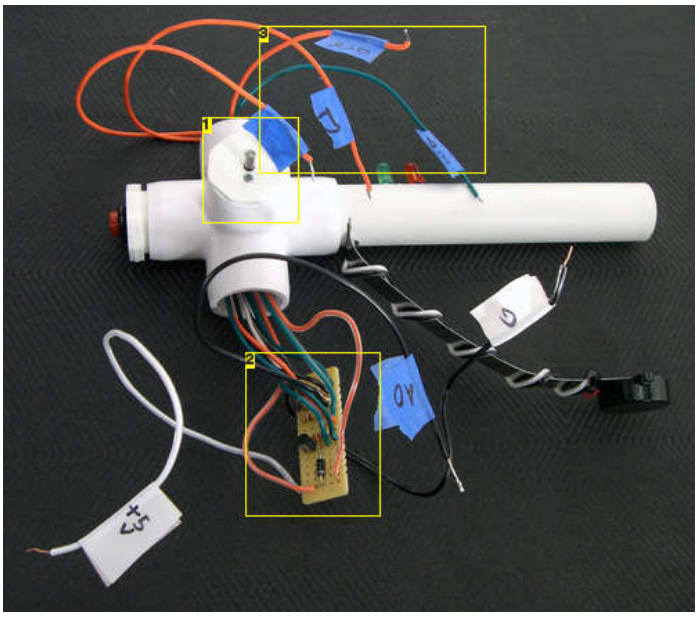


Image Notes
1. The motor in place.
2. The final circuit with the wires fed through.
3. All the connections that will go to the Digital side of the Arduino we thread through to the Short Arm side.

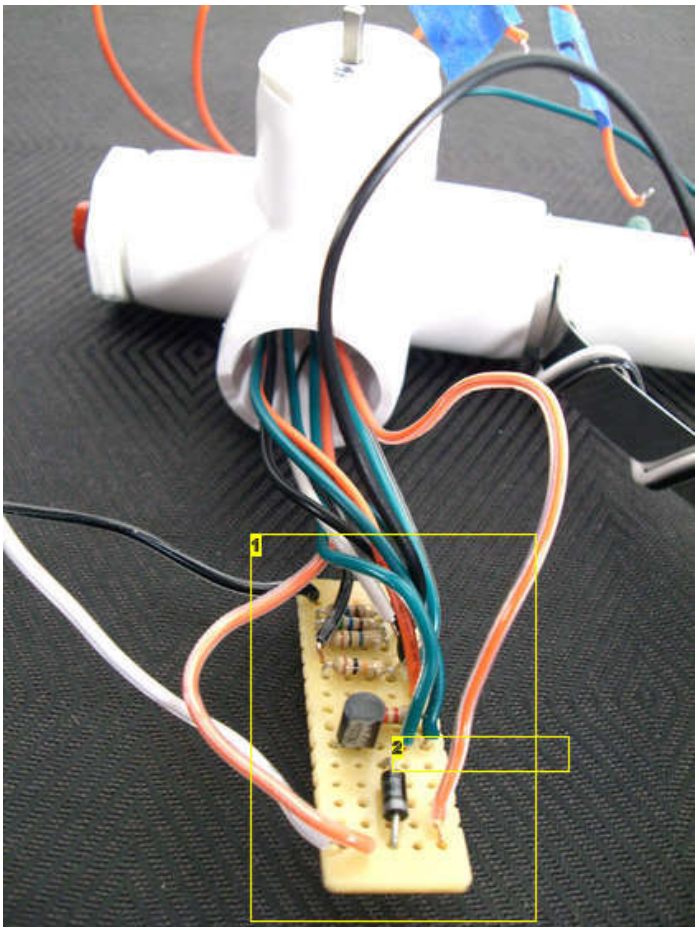


Image Notes
1. Close up of the completed board.
2. Embarrassing note: the motor connection (the green connection on the left) is not in the same hole as the diagram. It's connected the same, just in the wrong hole.

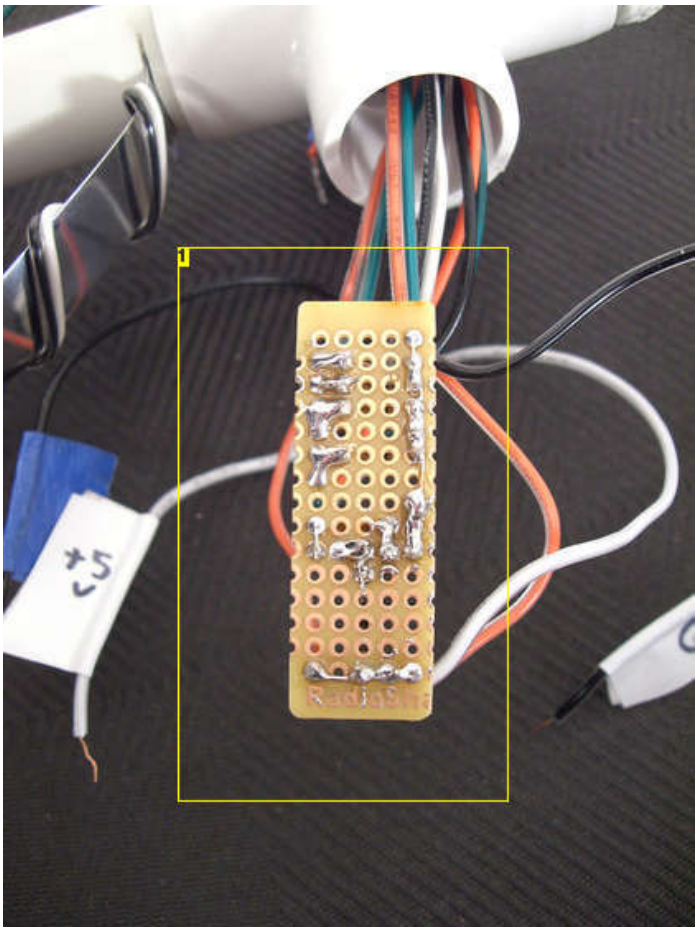


Image Notes
1. Here's the back of the board so you can see the bridged connections. And a lot of messy soldering.

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!

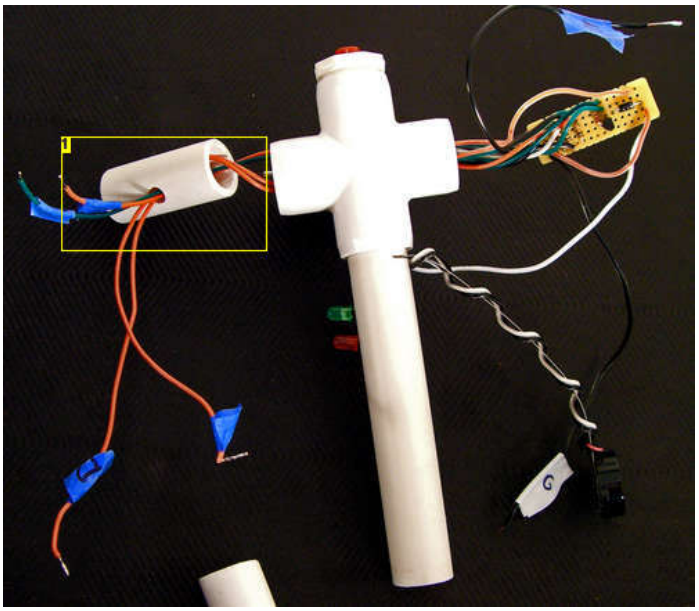


Image Notes
1. The digital pin wires threaded through the Short Arm

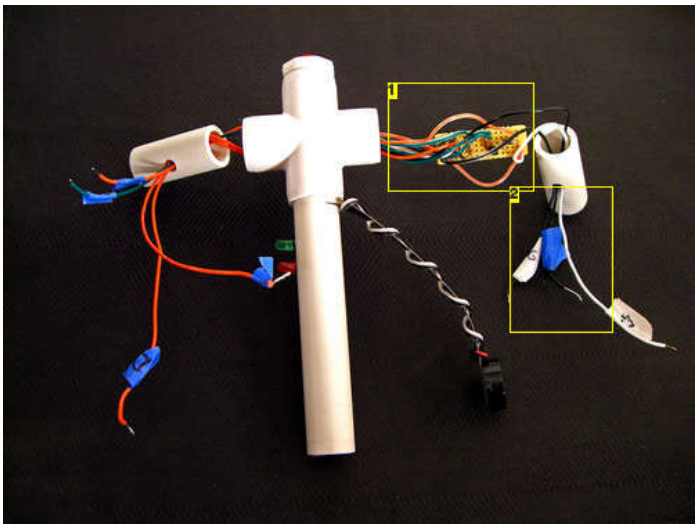


Image Notes
1. Circuit board getting ready to be crammed into the Board Arm.
2. The wires for Analog 0, Ground and +5v threaded through the hole in the Board Arm.

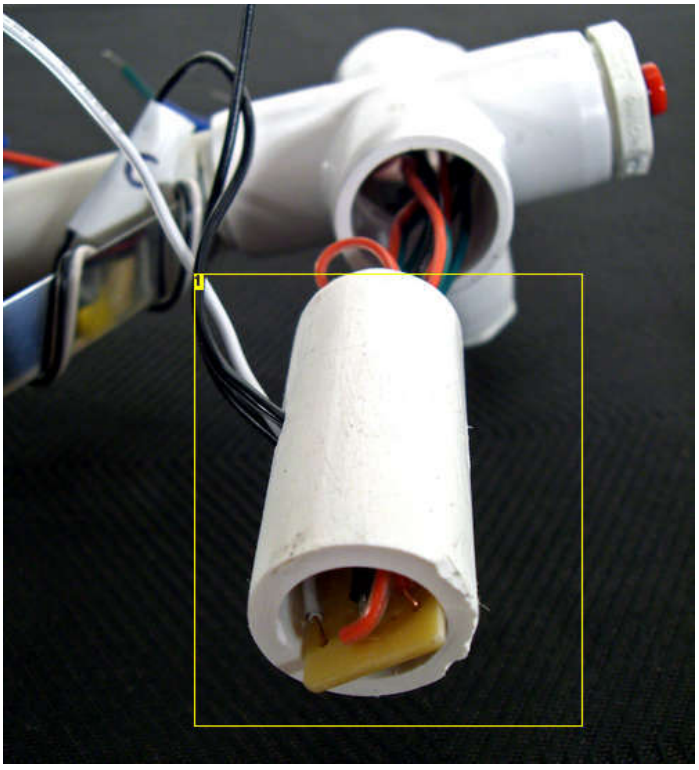


Image Notes
1. The circuit board has been pushed into the Board Arm. Having a little sticking out the far side is fine, it'll be hidden in the right angle adapter.

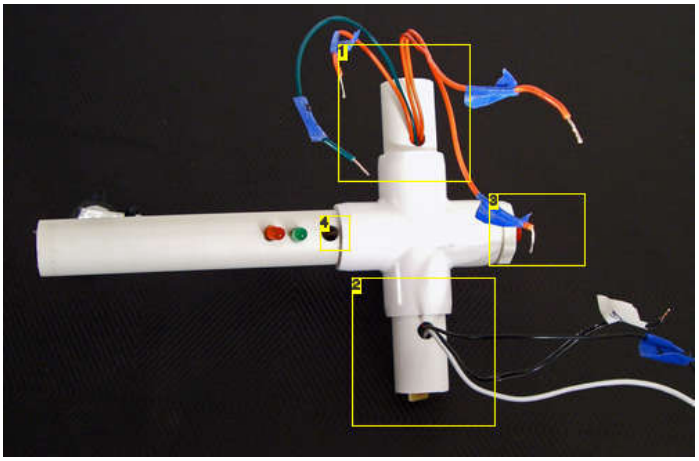


Image Notes
1. Short Arm plugged in.
2. Board Arm plugged in.
3. Button plugged in.
4. Once again, ignore this hole.

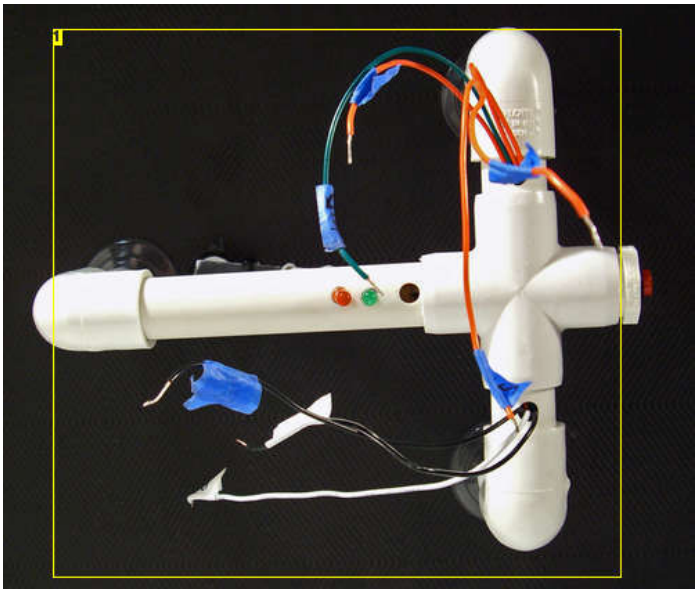


Image Notes
1. Legs and suction cups in place. at the end of the arms.

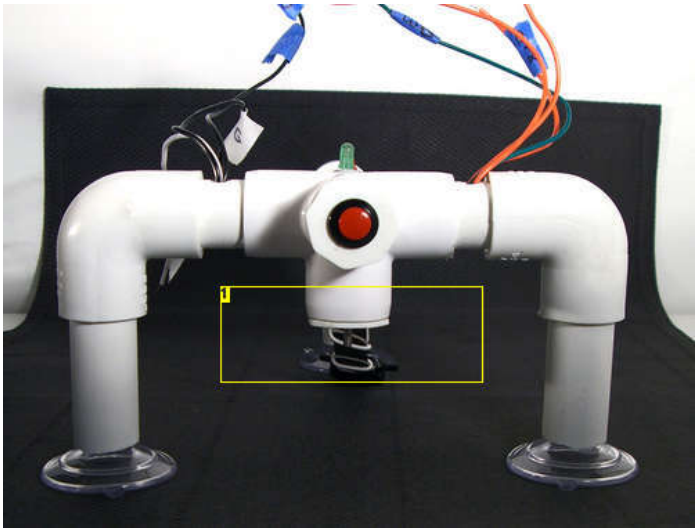


Image Notes
1. Side view. We haven't attached the lock clamp yet.

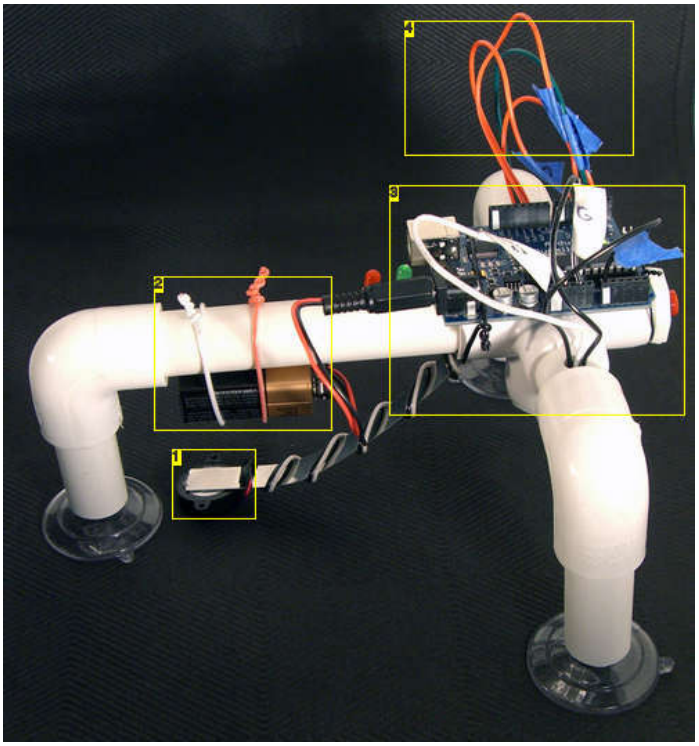


Image Notes
1. The knock sensor is arranged so it will press flatly against the door.
2. Battery attached.
3. Arduino attached and wires plugged.
4. Here is where a neat person would shorten the leads.

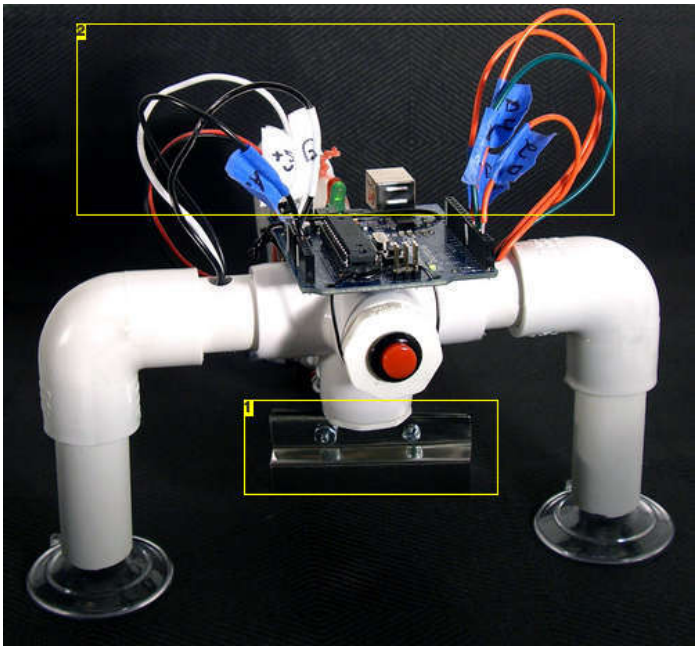


Image Notes
1. Side view. We've attached the lock clamp. Now we're ready to test it on the door!
2. Shortening the wires and removing the labels would make it look nicer, but you'll have to do that on your own time. I have an Instructable to finish!