Step 9: Mounting, Testing, and Use

First lets make sure our legs are the right length. Be sure the lock clamp is attached to your motor and the legs are in place. Attempt to fasten it to the door so the clamp fits over your lock. If you're lucky the legs will be the right length and you're ready to go! If you're unlucky the legs are too short and you'll have to cut new ones. But chances are your legs will be too long. Either grind/file/sand/cut them to the right length so the clamp fits right over the deadbolt lock handle.

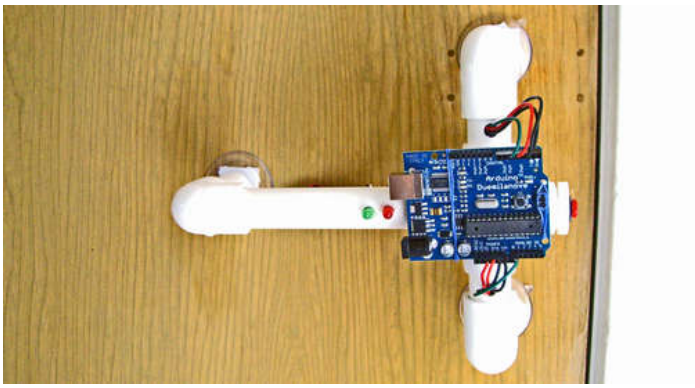
You don't have to use suction cups and in fact may doors are immune to their sucking. Two other options are to put PVC end caps on the legs and then secure them to the door with double sided foam tape (like this) or with screws through the caps, if you don't mind putting holes in your door.

Now that's its on you can give it a test. Do the first test from the inside. Lock your door and power it up. When the green light is on give it the old Shave and a Haircut and it should unlock!

Now program in a less obvious knock (or not) and your tree house will finally be safe from that smelly kid!

If the motor doesn't turn far enough to unlock your door you'll need to update the sketch to run the motor longer. (See Step 2!)

For additional Troubleshooting : See the bottom of **Step 2** . It also includes a bunch of other tweaks that might help you.



Step 10: Epilog: Changes And Improvements

There are many ways to improve or change this project if you're feeling ambitious. Here are a few to get you started, feel free to add more in the comments.

- Add an H-Bridge to the circuit so it can lock and unlock the door.
- Make it work in silent mode by removing the knock sensor and attach a capacitance (touch) sensor to the doorknob and record sequences of touches.
- Use a servo to unlock the door rather than this hacked together gear-motor+slip transmission.
- Add a potentiometer to adjust the knock sensitivity and other values.
- Build it into an actual door knocker.
- Use a more economical microcontroller and enable sleep mode for better battery life.
- Make the whole package small enough to fit inside the door.
- Store several knocks so several people can have their own 'private' knocks.
- Add a real-time clock and using different knocks for different days of the week.
- Add a knocker to provide feedback through the door. It could then offer a challenge-response security where the door starts a knock sequence and the user has to finish it correctly.
- Remove the knock sensor and record pushes of the doorbell or other hidden button.
- Remove the knock sensor and put a photosensor in the peephole, send the open code through the peephole with a keychain flashlight.

And here's a zero-technology solution to the "Yeah, but someone'll overhear your secret knocks!" problem: Scream while knocking. No one will overhear the knock over the racket you're making.

Did you build this?

Post a photo (or better yet a video!) photo of it mounted on a door will earn a Master Of Secret Knocks patch*!

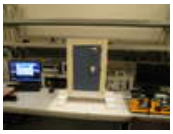
*As long as I have patches left to hand out. Which I do.

Masters of Secret Knocks:

- josiasfilho added a servo and locking ability.
- Jorad unlatches his door and added feedback in the peephole.
- Crimson-Deity added a pushbutton.
- bserrato added unlocking and a bluetooth camera to photograph people who give an incorrect knock.

Image Notes
1. Post a picture (or video!) of your secret knock detector in comments and get this fancy Master of Secret Knocks patch

Related Instructables



Secret Knock Detecting Door Lock (Photos) by vinny03

How to Break Down a Door by drums787

Magnetic Combination Lock Picture Safe by pastprimitive

Very Simple Arduino Electric Lock by RKlenka

Nintendo Keyless Entry System by action_owl

A bit of safe cracking... by killerjackalope

Easy Bluetooth Enabled Door Lock With Arduino + Android by Collin Amedee

Knock Block by jkestner

turn signal biking jacket

by **leahbuechley** on June 21, 2008

Intro: Turn signal biking jacket

This tutorial will show you how to build a jacket with turn signals that will let people know where you're headed when you're on your bike. We'll use conductive thread and sewable electronics so your jacket will be soft and wearable and washable when you're done. Enjoy!

A version of this tutorial is also on my website.

Image Notes
1. the first jacket I made

Step 1: Supplies

.

Get your supplies. You need:

-- LilyPad Arduino main board
-- FTDI connector
-- mini USB cable
-- LilyPad power supply
-- 16 LilyPad LEDs (note: these aren't available from SparkFun yet, but will be soon)
-- 2 push button switches
-- a spool of 4-ply conductive thread
-- a digital multimeter with a beeping continuity tester. This is the one I have.
-- a garment or a piece of fabric to work on
-- a needle or two, a fabric marker or piece of chalk, puffy fabric paint, a bottle of fabric glue, and a ruler
(Available at your local fabric shop or Joann Stores .)
-- a pair of scissors
-- double sided tape (optional)
-- a sewing machine (optional)

disclosure: I designed the LilyPad, so I'll make some $ if you buy one.

Image Notes
1. conductive thread and needle
2. chalk for drawing on fabric
3. LilyPad Arduino main board, power supply and USB link
4. LilyPad LEDs
5. switches
6. fabric glue
7. mini USB cable (like the one for your camera)

Step 2: Design

.

Plan the aesthetic and electrical layout of your piece.

Decide where each component is going to go and figure out how you will sew them together with as few thread crossings as possible. Make a sketch of your design that you can refer to as you work. The photos below show the sketches for my jacket. Stitching for power (+) is shown in red, ground (-) in black, LEDs in green, and switch inputs in purple.

Important note about the power supply

As you design, plan to keep your power supply and LilyPad main board close to each other. If they are too far apart, you are likely to have problems with your LilyPad resetting or just not working at all.

Why? Conductive thread has non-trivial resistance. (The 4-ply silver-coated thread from SparkFun that comes with the LilyPad starter kit has about 14 ohms/foot.) Depending on what modules you're using in your construction, your LilyPad can draw up to 50 milliamps (mA) of current, or .05 Amps. Ohm's law says that the voltage drop across a conductive material--the amount of voltage that you lose as electricity moves through the material--is equal to the resistance of the conductive material times the amount of current that is flowing through it.

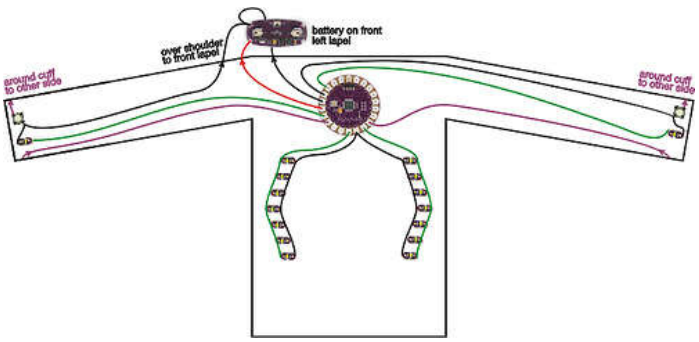
For example, if your LilyPad is a foot away from the power supply, the total resistance of the conductive material that attaches your LilyPad to your power supply is about 28 ohms. (14 Ohms in the conductive thread that leads from the negative terminal of the power supply to the negative petal on the LilyPad and 14 Ohms in the conductive thread that ties the positive terminals together). This means we can expect a drop of 1.4 Volts (28 Ohms * .05 Amps.) This means that while 5 Volts is coming out of the power supply, the LilyPad will only be getting 3.6 Volts (5 Volts - 1.4 Volts). Once the voltage at the LilyPad drops below about 3.3 Volts, it will reset. The resistance of the traces from + on the power supply to + on the LilyPad and - on the power supply to - on the LilyPad should be at most 10 Ohms. Plan the distance accordingly.

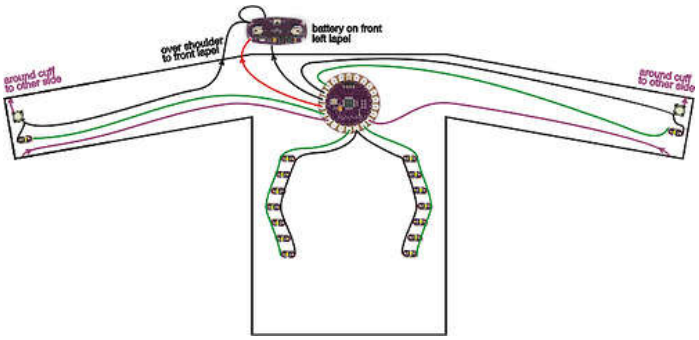
If all of this was confusing, don't worry! Just keep the LilyPad and power supply close to each other in your design.

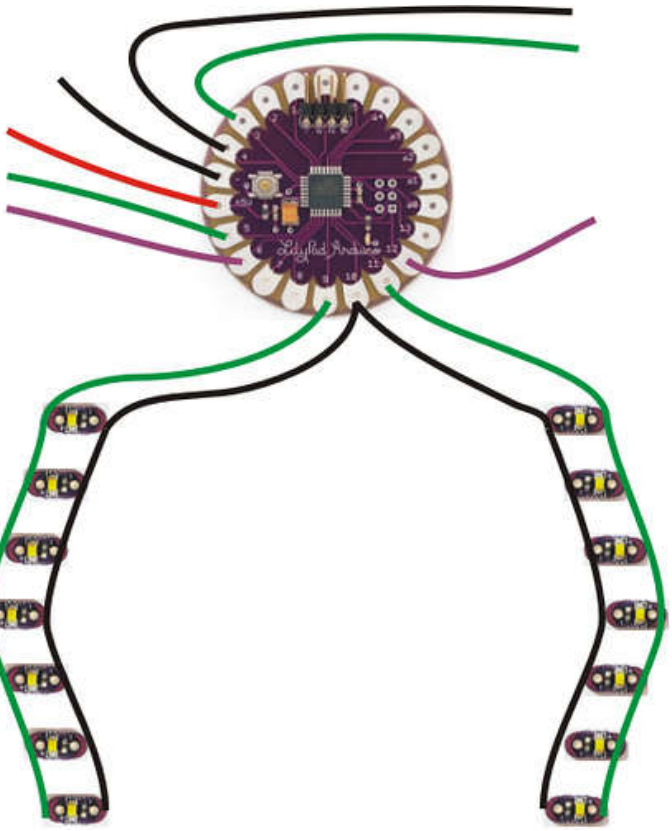
Transfer the sketch to your garment.

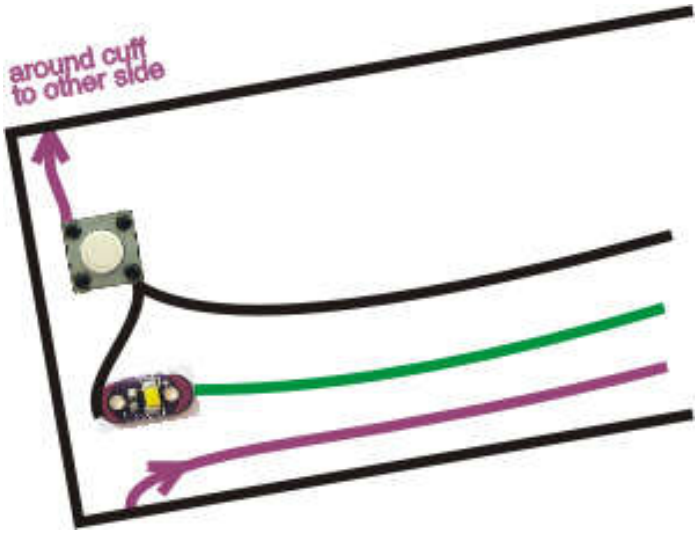
Use chalk or some other non-permanent marker to transfer your design to the garment. If you want, use a ruler to make sure everything is straight and symmetrical.

Use double sided tape to temporarily attach LIlyPad pieces to your garment. This will give you a good sense of what your final piece will look like. It will also keep everything in place and, as long as the tape sticks, make your sewing easier.




around cuff to other side
over shoulder to front lapel
battery on front left lapel
around cuff to other side




around cuff to other side

Step 3: Sew your power supply and LilyPad to your jacket

.

First, trim the leads off of the back of the power supply

Get out your LilyPad power supply piece and trim the metal parts that are sticking out the back of it. Small clippers like the ones shown in the photo work well, but you can also use scissors.

Stabilize your battery on the fabric.

Generally, you want to do everything you can to keep the power supply from moving around on the fabric. I recommend gluing or sewing the battery down before starting on the rest of the project. You may also want to glue or sew something underneath the power supply to help prevent it from pulling on the fabric and bouncing around as you move.

If you are working on a thin or stretch piece of fabric--first of all, reconsider this choice! It's much easier to work on a heavy piece of non-stretchy fabric. If you are determined to forge ahead with a delicate fabric, choose the location for your power supply wisely. It's the heaviest electronic module, so put it somewhere where it will not distort the fabric too badly. definitely glue or sew something underneath the power supply.

Sew the + petal of the power supply down to your garment.

If you are new to sewing, check out this great introduction before you start for info on how to thread a needle, tie knots and make stitches. Cut a 3-4 foot length of conductive thread. Thread your needle, pulling enough of the thread through the needle that it will not fall out easily. Tie a knot at the end of the longer length of thread. Do not cut the thread too close to the knot or it will quickly unravel.

Coming from the back of the fabric to the front, poke the needle into the fabric right next to the + petal on the power supply and then, from the front of the fabric, pull it through. The knot at the end of the thread will keep the thread from pulling out of the fabric. Now make a stitch going into the hole in the hole in the + petal on the power supply. Do this several more times, looping around from the back of the fabric to the front, going through the + petal each time.

Pay special attention to this stitching. It is the most important connection that you'll sew in your project. You want to make sure you get excellent contact between the petals on the power supply and your conductive thread. Go through the hole several times (at least 5) with your stitching. Keep sewing until you can't get your needle through anymore. Do not cut your thread, just proceed to the next step.

Sew from the battery to the LilyPad.

Once you've sewn the + petal of the battery down, make small neat stitches to the + petal of your LilyPad. I used a jacket with a fleece lining and stitched only through the inner fleece lining so that no stitches were visible on the outside of the jacket.

Sew the + petal of your LilyPad down, finishing the connection.

When you reach the LilyPad, sew the + petal down to the fabric with the conductive thread. Just like you were with the battery petal, you want to be extra careful to get a robust connection here. This stitching is making the electrical connection between your power supply and LilyPad.

When you are done with this attachment, sew away from the LilyPad about an inch along your stitching, tie a knot, and cut your thread about an inch away from the knot so that your knot won't come untied.

Put fabric glue on each of your knots to keep them from unraveling.

Once the glue dries, trim the thread close to each knot.

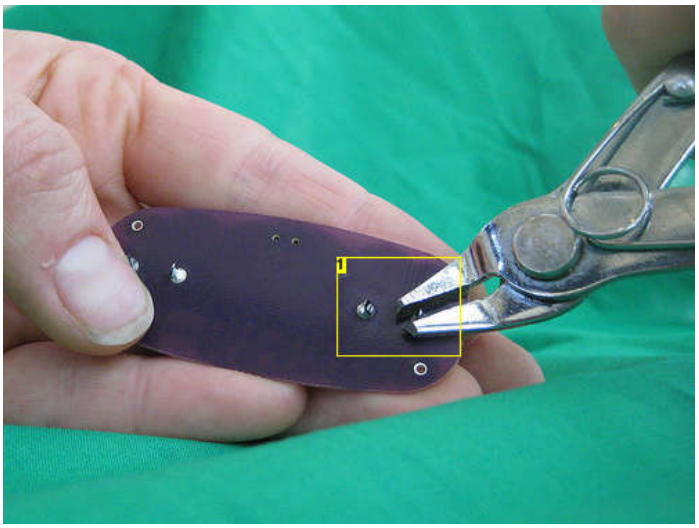
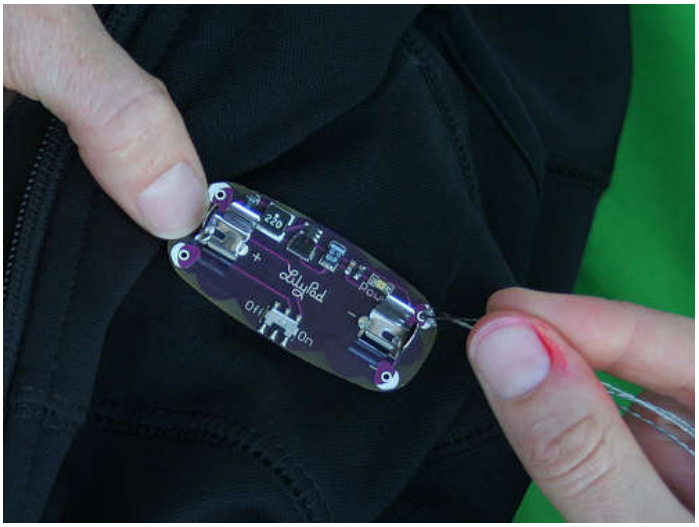


Image Notes
1. trimming the battery posts off the power supply.



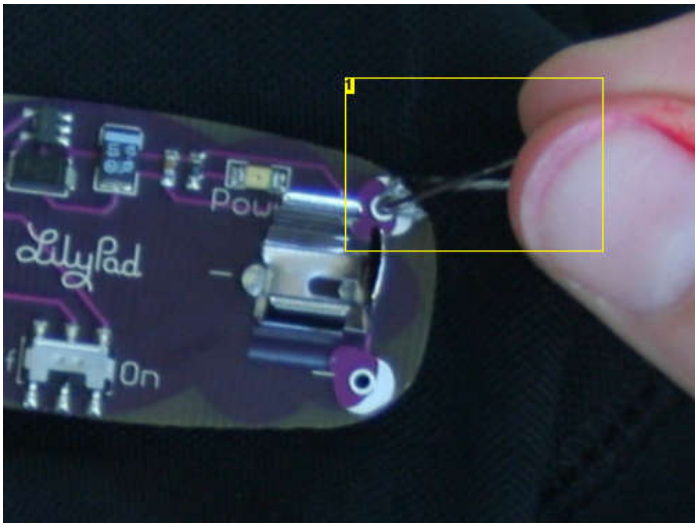
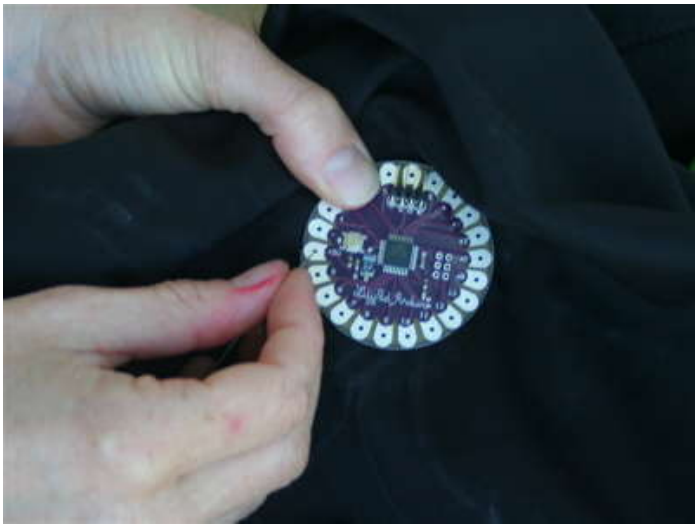
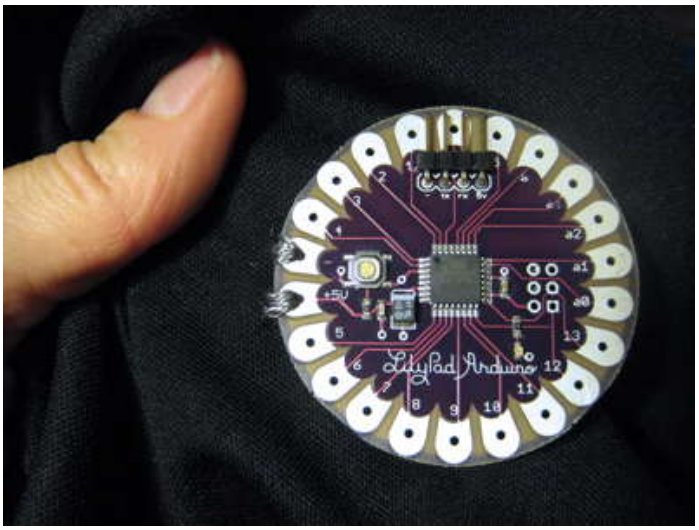


Image Notes
1. sewing on the + petal of the power supply. notice how I'm sewing through the hole from the front instead of the back, which is much harder.





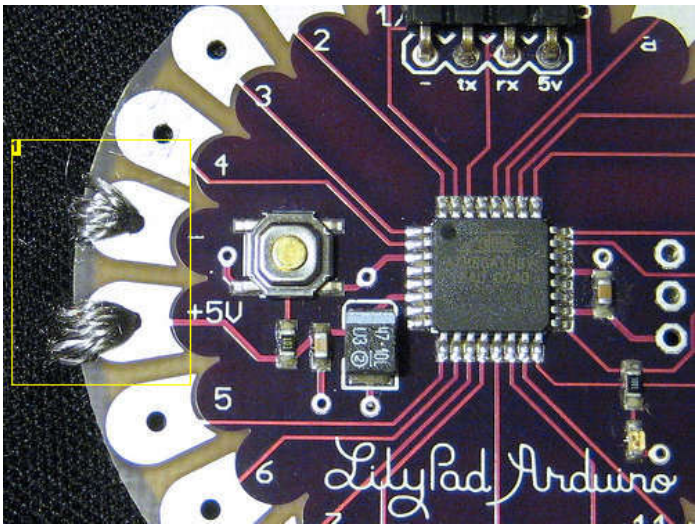
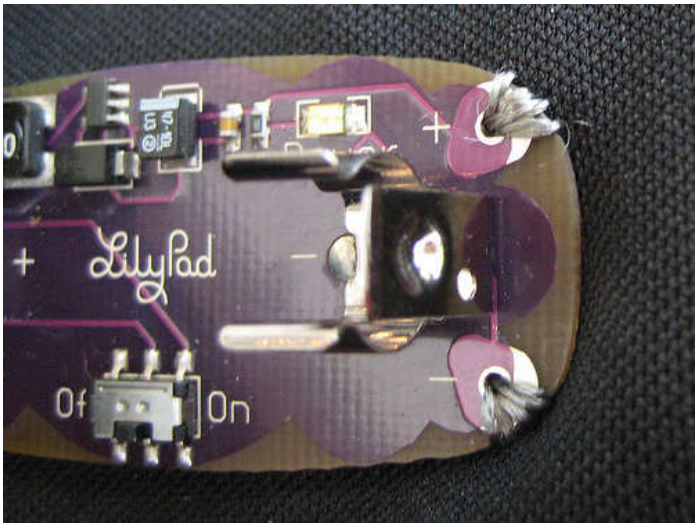


Image Notes
1. notice how dense my stitching is here. this is what your stitches should look like.



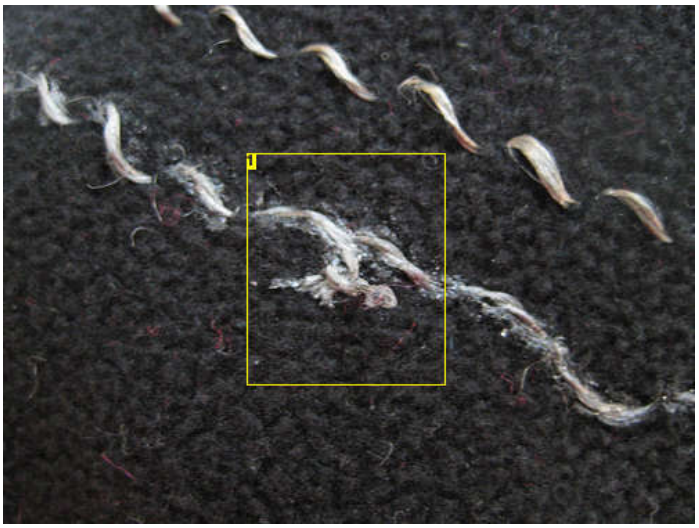


Image Notes
1. a glued and trimmed knot. knots without glue will come unraveled quickly.

Step 4: Test your stitching

.

Measure the resistance of your stitching.

Get out your multimeter and put it on the resistance measuring setting. Measure from power supply + to LilyPad + and power supply - to LilyPad -. If the resistance of either of these traces is greater than 10 ohms, reinforce your stitching with more conductive thread. If you're not sure how to measure resistance, check out this tutorial .

Put a AAA battery into the power supply and flip the power supply switch to the on position. The red light on the power supply should turn on. If it doesn't and you're sure you flipped the switch, quickly remove the battery and check for a short between your + and - stitches. (Most likely there is a piece of thread that's touching both the - and + stitching somewhere.) You can test for a short between + and - by using the beeping continuity tester on your multimeter. See this tutorial for information on how to use the continuity tester.

Also check the resistance between the + and - stitching. If the resistance is less than 10K Ohms or so, you've got a mini-short (probably a fine conductive thread hair that is touching both + and -) that you need to find and correct.

If the power supply does turn on, look at your LilyPad. It should blink quickly each time you press its switch. Once these connections are working properly, turn off the power supply and remove the battery.

Insulate your power and ground stitching

So, your jacket is now full of uninsulated conductive stitches. This is fine when a body is inside of it. A body will prevent sewn traces from contacting each other. But when the jacket is off of a person and you bend or fold it, traces will touch each other and short out. To fix this problem, cover your traces with puffy fabric paint (or another insulator like a satin stitch in regular thread). But, you don't want to cover traces until you're sure that everything works! So, use good judgment in when to coat traces.

Image Notes
1. this is the resistance measuring setting

Image Notes
1. this is the continuity testing setting

Step 5: Sew on your turn signal LEDs

.

Sew in your left and right signals.

Using the same techniques you used to sew the power supply to the LilyPad, attach all of the + petals of the lights for the left turn signal together and to a petal on the LilyPad (petal 9 for me) and all of the + petals for the right signal together and to another LilyPad petal (11 for me). Attach all of the - petals of the lights together and then to either the - petal on the LilyPad or another LilyPad petal (petal 10 for me). Refer back to my design sketches if any of this is confusing.

Remember to seal each of your knots with fabric glue to keep them from unraveling. Be careful to avoid shorts; don't let one sewn trace touch another. In this case, the - traces for the LEDs are all connected, but you want to make sure that the + traces for the left and right signals do not touch the - trace or each other.

Test your turn signals.

Load a program onto your LilyPad that blinks each turn signal to make sure all of your sewing is correct.

Note, if you don't know how to program the LilyPad, work through a few of these introductory tutorials before proceeding.

Here's my test program:

```
int ledPin = 13; // the LED on the LilyPad
int leftSignal = 9; // my left turn signal is attached to petal 9
int rightSignal = 11; // my right turn signal is attached to petal 11
int signalLow = 10; // the - sides of my signals are attached to petal 10

void setup()
{
pinMode(ledPin, OUTPUT); // sets the ledPin to be an output
pinMode(leftSignal, OUTPUT); // sets the leftSignal petal to be an output
pinMode(rightSignal, OUTPUT); // sets the rightSignal petal to be an output
pinMode(signalLow, OUTPUT); // sets the signalLow petal to be an output
digitalWrite(signalLow, LOW); // sets the signalLOW petal to LOW (-)
}

void loop() // run over and over again
{
delay(1000); // wait for 1 second
digitalWrite(leftSignal, LOW); // turn the left signal off
delay(1000); // wait for 1 second
digitalWrite(rightSignal, HIGH); // turn the right signal on
delay(1000); // wait for 1 second
```

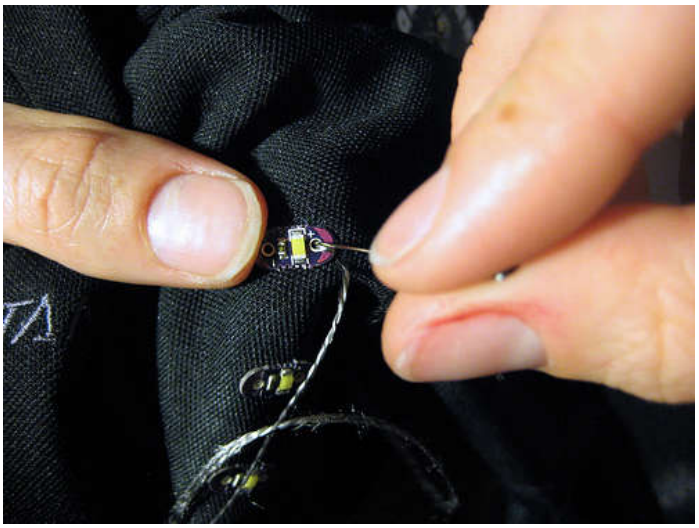
```
digitalWrite(rightSignal, LOW); // turn the right signal off
delay(1000); // wait for 1 second
}
```

If your layout is the same as mine, you can just copy and paste this program into your Arduino window.

If your turn signals don't work, use your multimeter (and the instructions from the last step) to test for shorts or bad connections and make sure that your program matches your physical layout.

insulate your turn signal stitches

Cover your traces with puffy fabric paint. Remember, you don't want to cover traces until you're sure that everything works! Use good judgment in when to coat traces.



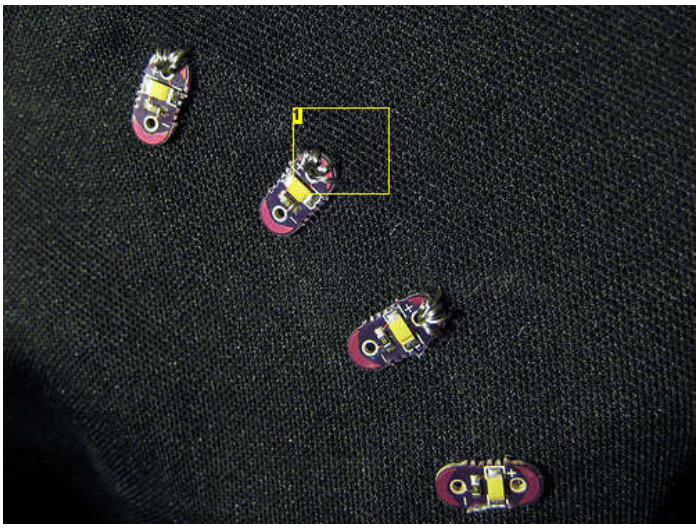


Image Notes
1. stitching in process, outside view: 3 + petals are sewn together

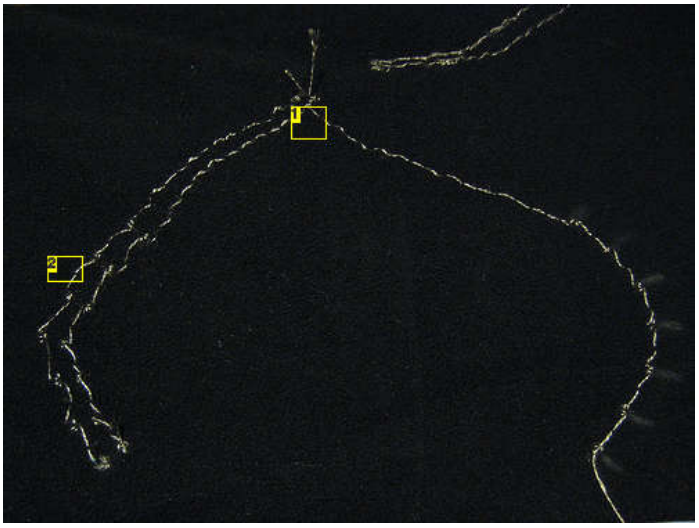


Image Notes
1. these 2 middle traces are the negative (-) traces for all of my turn signal LEDs. these traces are attached to petal 10 on my LilyPad
2. this is the stitching for the positive (+) leads of my right turn signal LEDs. (Since this is an inside view, everything is reversed.) These traces lead to petal 11 on my LilyPad.

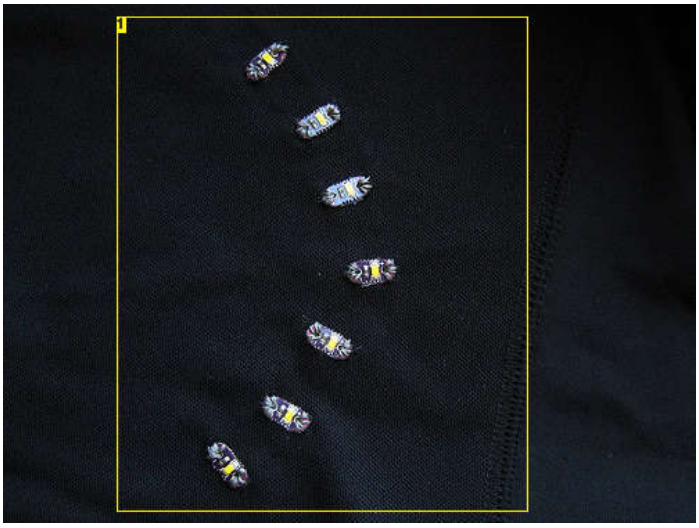


Image Notes
1. my finished right turn signal. notice how my stitching doesn't come through to the outside of the garment.

Step 6: Sew in your control switches

.

Place your switches

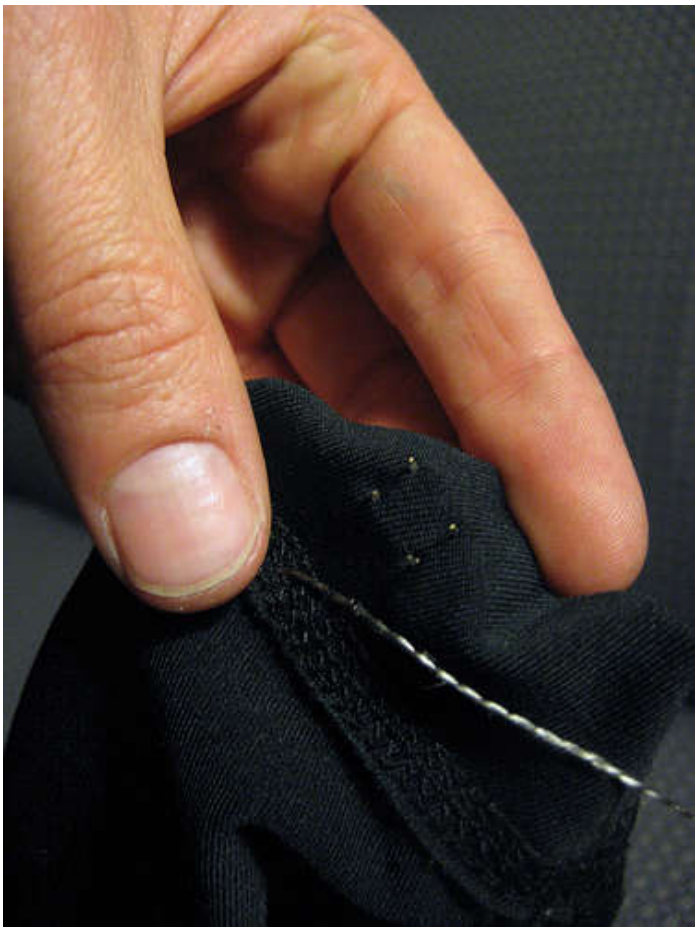
Find a spot for your switches where they'll be easy to press when you're riding your bike. I mounted mine on the underside of my wrists. I found a good spot by trying out different places. Check out the photos to see what I mean.

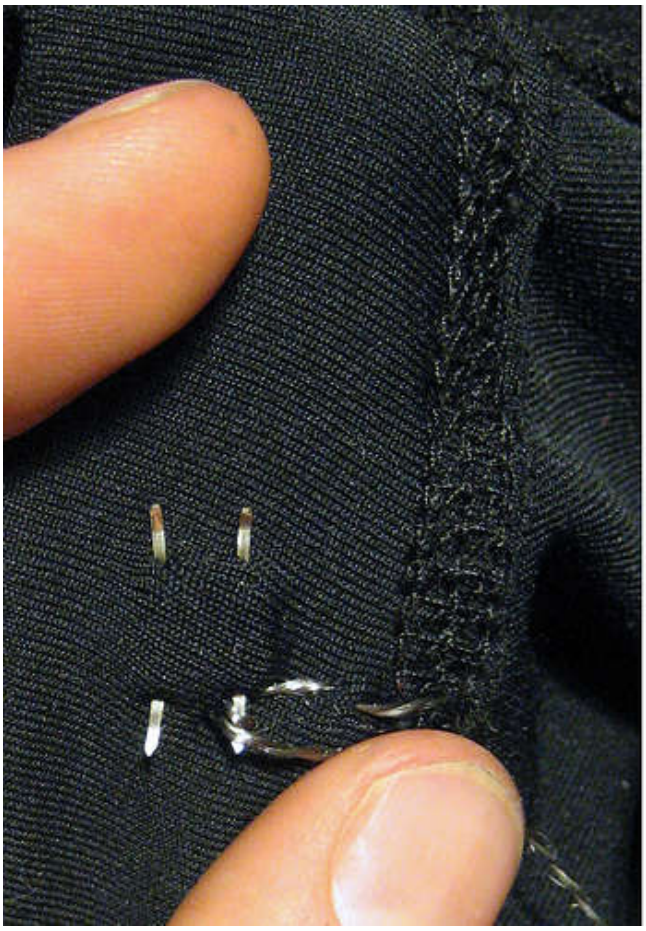
Once you've found a good position, push the legs of the switch through the fabric and bend them over on the inside of the fabric.

Sew in your switches.

Sew your switches into the garment. Sew 1 leg to the switch input petal on the LilyPad and another leg, one that is diagonally across from the first , to ground or another LilyPad petal. I used petal 6 for the switch input on the left side and petal 12 for switch input on the right side. I used - for the - connection on the left side, but petal 4 for the - connection on the right side. Refer back to my design drawings if any of this is confusing.

When you're done sewing, go back and reinforce the switch connections with glue. You don't want your switches to fall out of their stitching.





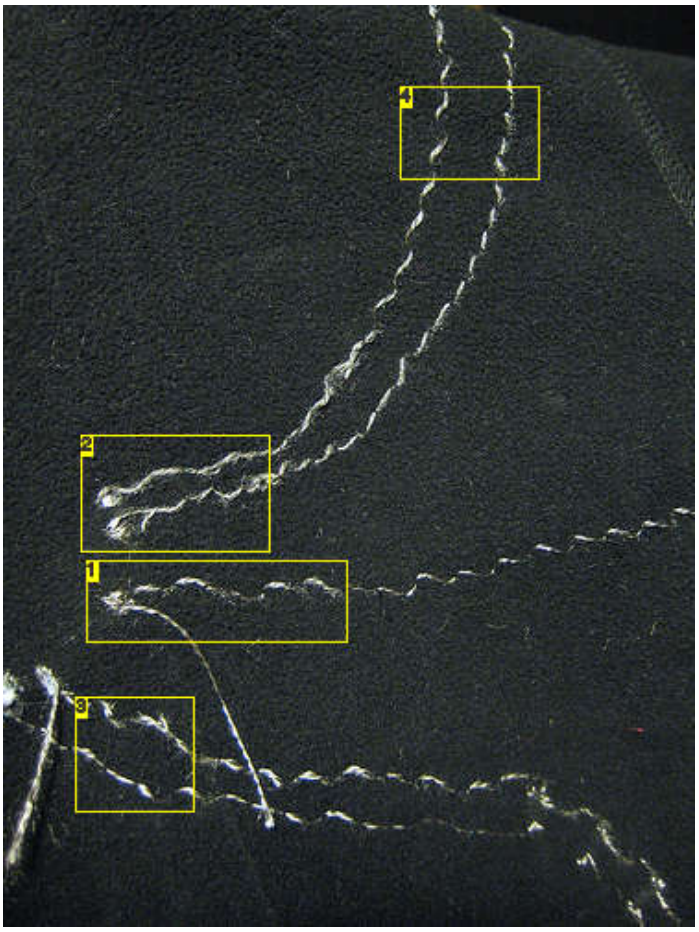
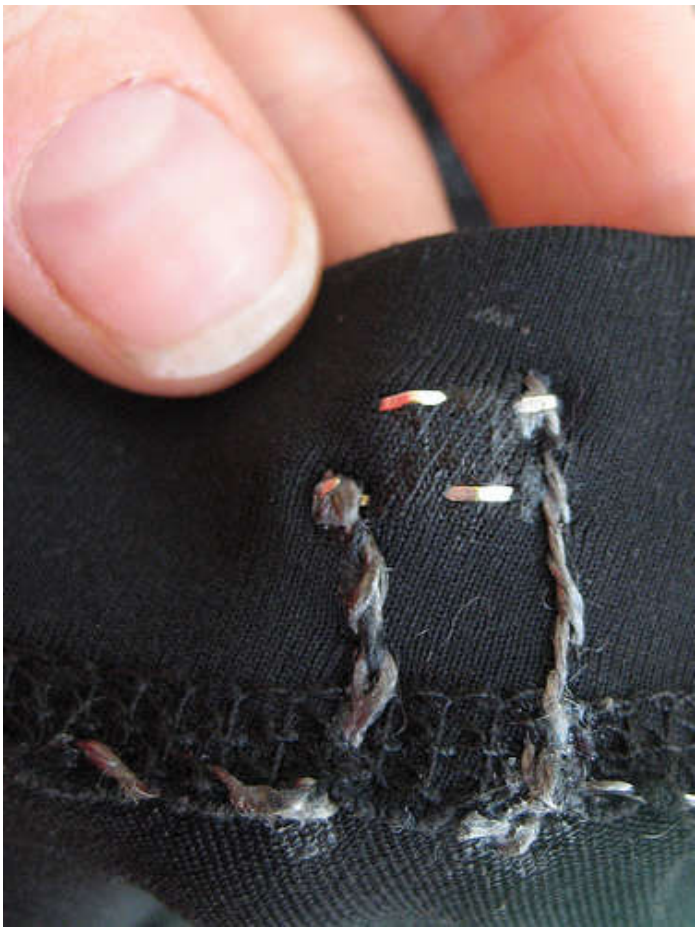


Image Notes
1. the first trace from my left switch is finished. this is the switch input trace that is tied to petal 6 on the LilyPad. I just have to glue and trim the knot.
2. these are the stitches that lead from the power supply to the LilyPad.
3. these are my left turn signal stitches. I have a knot to glue and trim on these too.
4. you might have noticed that I didn't insulate my traces. you too can leave them uninsulated, but be aware of shorts from folding/bending whenever you're not wearing the jacket! especially when you are programming and troubleshooting.



Step 7: Sew in your indicator LEDs

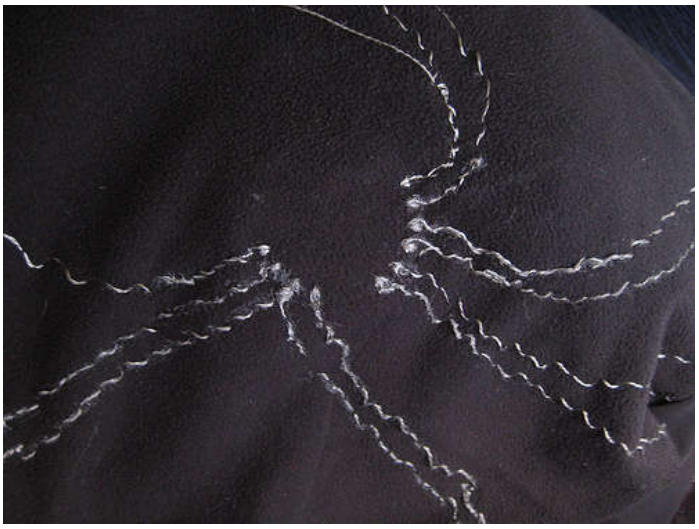
.

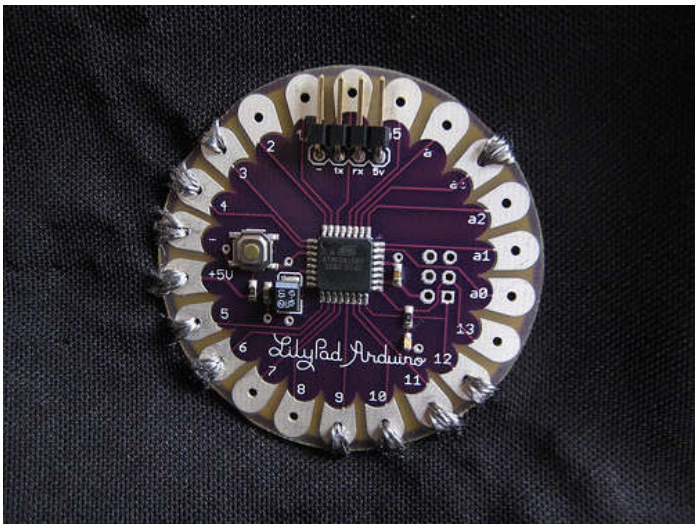
Sew a single LED onto the sleeve of each arm.

These will give you essential feedback about which turn signal is on. They'll flash to tell you what the back of your jacket is doing, so make sure they're in a visible spot. Sew the + petals of each LED to a LilyPad petal and the - petals of each LED to the - side of the switch (the - trace you sewed in the last step). I used petal 5 for the LED + on the left side and petal 3 for the LED + on the right side. Again, refer back to my design drawings if any of this is confusing.

As always, remember to glue and trim knots and be careful not to create any shorts.

Once you sew both wrist LEDs, you're done with the sewing phase of the project! Now, on to programming...





Step 8: Program your jacket

.

Decide on the behavior you want.

I wanted the left switch to turn on the left turn signal for 15 seconds or so, and the right switch to do the same thing for the right signal. Pressing a switch when the corresponding turn signal is on should turn the signal off. Pressing both switches at the same time should put the jacket into nighttime flashing mode. The wrist mounted LEDs should provide feedback about the current state of the jacket. Here's the code I wrote to get that behavior.

Program your jacket

To program your garment, copy and paste my code into an Arduino window and load it onto the LilyPad. You may have to make some small adjustments first depending on where you attached lights and switches. Play with delays to customize your blinking patterns. Follow my LilyPad introduction instructions if you need more information on how to program the LilyPad or how to make sense of my code.

Plug your battery back in and see if it works and...go biking!

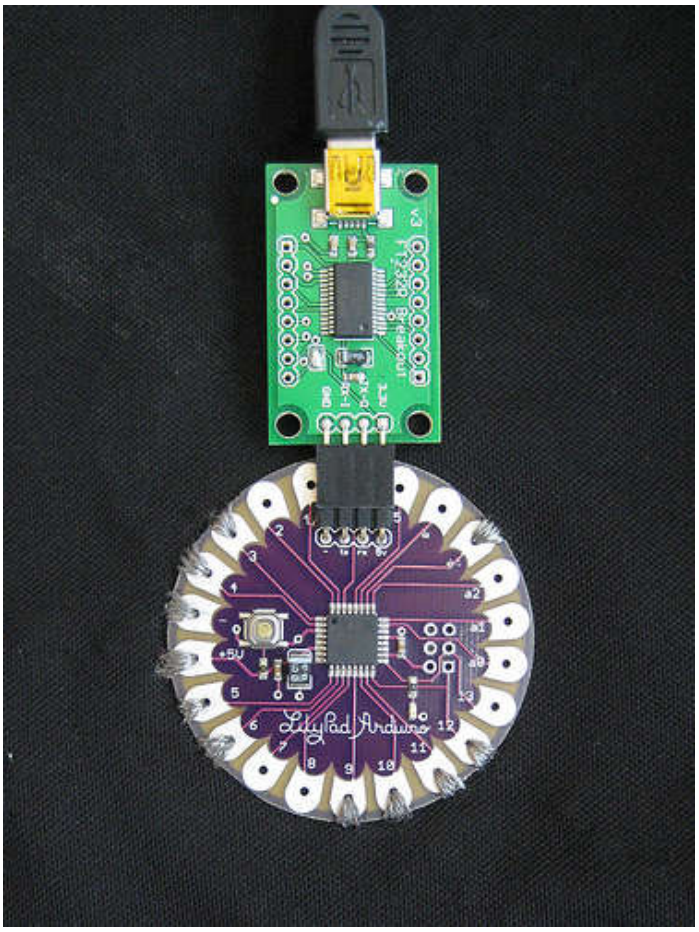
Insulate the rest of your traces

Cover the rest of your traces with puffy fabric paint. Again, don't coat anything until you're sure it works.

About washing

Your creation is washable. Remove the battery and wash the garment by hand with a gentle detergent.

Note: silver coated threads will corrode over time and their resistance will gradually increase with washing and wear. To limit the effects of corrosion, insulate and protect your traces with puffy fabric paint or some other insulator. You can also revive exposed corroded traces with silver polish. Try this on a non-visible area first to see what it does to your fabric!


LilyPad Arduino

Related Instructables

Programmable LilyPad EL-Wire Dress by quasiben

soundie: a musical touch-sensitive light-up hoodie by kanjun

EL Driver Board by quasiben

LilyPad Arduino Blinking Bike Safety Patch by bekathwia

Interactive Bee Game by quasiben

Latch-Modified Turn-Signal Jacket by quasiben

Light for life: Glowing button cycling jacket by kempton

The Motivational Moody Workout T-Shirt by Lingon

Tree Climbing Robot

by **Technochicken** on August 17, 2011

Intro: Tree Climbing Robot

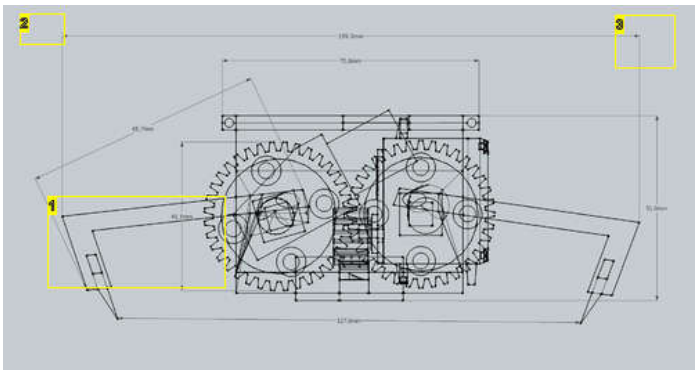
After I got comfortable programming and building with an Arduino, I decided to build a robot. I did not have any particular type in mind, so I wracked my brain (and the internet) for cool robot ideas. Eventually, somehow the idea popped into my head to build a robot that could climb trees. At first I dismissed the idea as beyond my skill level, but after further thought, and some time in Sketchup, I decided to take a shot at the challenge. This is the result of my efforts.

Step 1: Design

I started out by creating a basic design in Sketchup. The robot was to consist of two segments, joined by a spine which could be extended or retracted. Each segment would have four legs with very sharp points as feet. To climb, the legs on the top segment would pinch together and the sharp feet would dig into the bark, securing the robot. Then the spine would be retracted, pulling up the bottom segment. The legs on the bottom segment would then grip the tree, and the top segment would release. Finally, the spine would extend, pushing the top segment upwards, and the process would repeat. The climbing sequence is somewhat similar to the way an inchworm climbs.

In my original design (show in the images above), all four legs in each segment were controlled by one highly geared down motor. I decided to ditch this idea for a few reasons. Firstly, I could not find the type of spur gear needed to mesh the legs together. Also, with all the legs linked together, the robot would have a hard time gripping uneven surfaces. Finally, I decided that the robot would be much easier to build if the motors drove the legs directly.

The other significant change I made from my original design was the way the spine worked. In my model, I used a rack and pinion type gearing system to extend and contract the spine. However, I could not find the necessary parts to build such a system, so I ended up using a threaded rod coupled to a motor to actuate the spine.



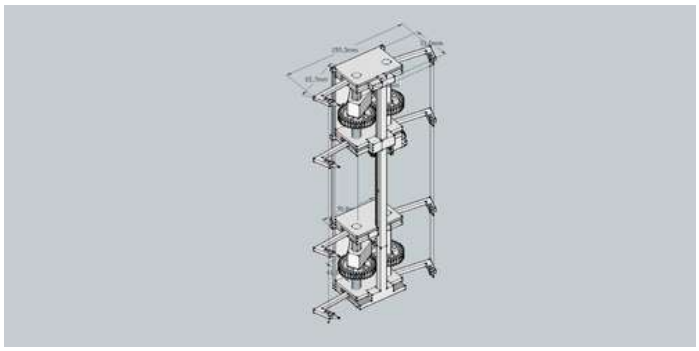


Image Notes
1. The strange leg shape came from resizing the legs all at once in Sketchup.
2. Front on

3. To make these types of images in Sketchup, turn X-ray on and turn perspective off.

Step 2: Tools and Materials

Microcontroller

- Arduino Uno (any will work)

Motor Controller

- 3X L298HN - these can be gotten for free as samples from ST
- 2.5" x 3.125" Perf Board
- Terminal Strips
- 22AWG Solid core wire
- 3X Aluminum heatsinks (I cut in half an old northbridge heatsink)
- Thermal paste

Power

- 9V Battery (to power the Arduino)
- Approximately 12V LiPo or Li-ion battery (I modified a laptop battery, so I did not even need to buy a charger)
- 5V regulator (To regulate power to the motor controller logic circutry)
- 9V Battery clip
- Barrel connector (Must fit the Arduino power connector)

Other Electronics

- 4X 7 RPM Gear Motor (These power two legs each)
- 4X Thin linear trim pots (Rotation sensors for the legs)
- DPDT Toggle switch (Power switch)
- SPDT Slide switch (User input)
- 2X Mini Snap Action Switch (Limit switch)
- 3 10K resistors (Pull down)
- Headers
- Signal Wire (Old IDE cables work really well, and let you organize your wires easily)
- Heat Shrink Tubing

Hardware

- 12' 3/4" x 1/8" Aluminum Bar (These come in 6' lengths at my local hardware store)
- 6" x 3" acrylic sheet (Electronics are mounted to this)
- 6x Standoffs with screws
- 1' Threaded rod and corresponding 1/2" nut
- 2X 1' x 3/16" steel rod
- 1' x 3/16" I.D. Brass Tubing
- 4X 5mm Aluminum Universal Mounting Hub
- Pack of large T Pins
- 4X 3/32 screws (to mount the motors)
- An assortment of 4/40 screws and nuts
- Assorted hex screws and nuts
- 4X Bic pens (I used the plastic shafts to fix the pots on the legs in place)
- 4X Locknuts
- 5 Minute epoxy
- Sheet metal scraps (For spacing and mounting things. Bits of Meccano work well)
- Stick on Velcro (For holding on the batteries)
- Hard Drive reading head bearing
- 3/4" Plastic angle
- Electrical Tape
- Zip Ties

Tools

- Electric Drill / Drill press (As well as a lot of bits)
- Hacksaw
- Soldering Iron
- Pliers
- Allen wrench
- Assorted screwdrivers
- Wire Strippers
- C Clamp (These can be used to make nice 90 degree bends in the aluminum)
- Ruler
- Files

Nonessential

- Bench PSU
- Multimeter
- Breadboard

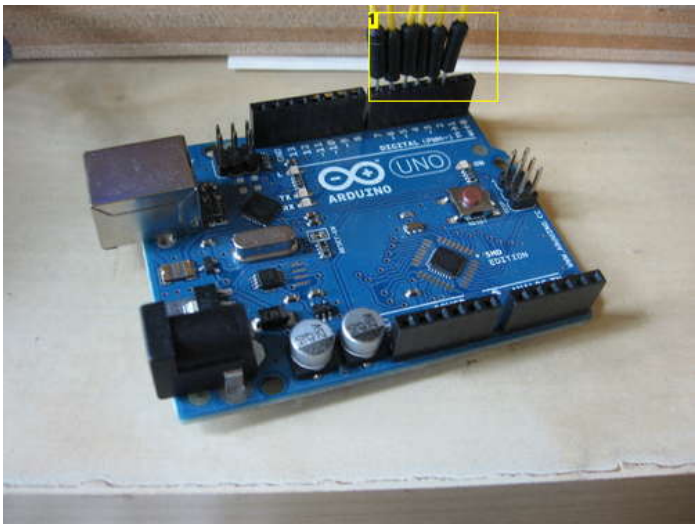


Image Notes
1. Still hooked up to my LED matrix

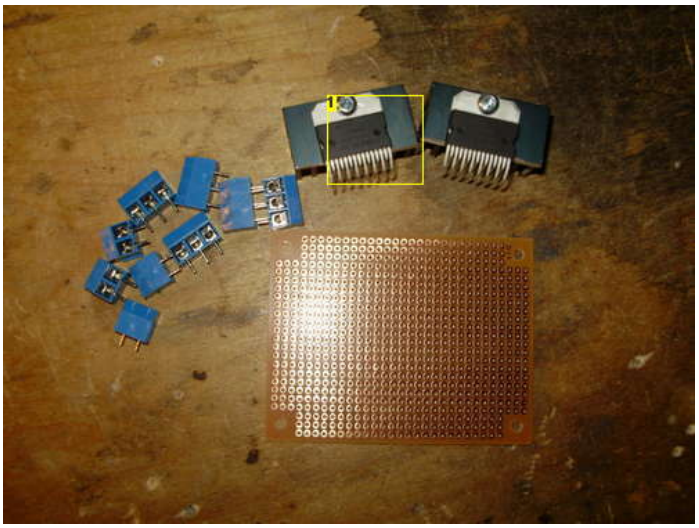


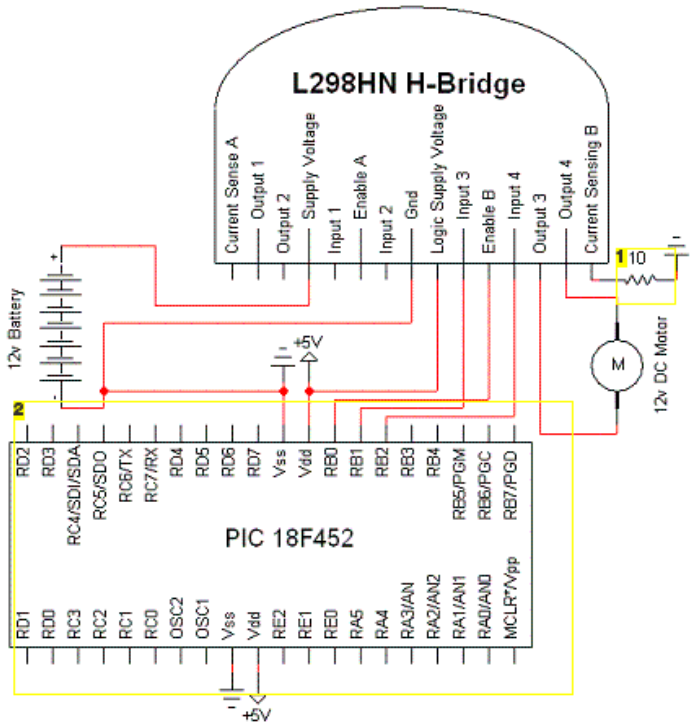
Image Notes
1. L298HN mounted to heatsink

Image Notes
1. 400:1 gearbox for lots of torque

Step 3: Motor Controller

The motor controller I built for this robot is based off the L298HN Dual Full Bridge chip. To use the chip, I followed the guide here . To start out, I placed all the components on a piece of perf board, to figure out the layout. With this chip, each motor requires three inputs to work: an enable signal and two input signals. The enable signal is used to control the motor speed with PWM, but since I did not need to control PWM, I just wired all the enable pins in parallel to a 5V line when I hooked the controller up to the Arduino. Once I figured out the layout, I soldered all the components in place, and made connections with 22AWG solid core wire. Finally, I spread some thermal paste on the back of the L298's, and screwed on the heatsinks. The particular heatsinks I used were made by cutting in half a northbridge heatsink from a computer motherboard, and drilling and tapping a hole for the screw. They are probably much larger than needed, but there is no harm in having over sized heatsinks. A higher resolution image of the labeled board can be found here .

When finished, this motor controller should be able to bidirectionally control 4 DC motors at up to 2A each (probably 2A continuously, because of the size of the heatsinks). As you may notice, this leaves me one motor short. My original design used a servo to actuate the spine, but I had to change my design to using a DC motor. To power it, I wired my third L298 chip to a molex connector (so I can disconnect the motor) and soldered on wires for all the connections. It does not look as pretty as my controller on a circuit board, but it works.



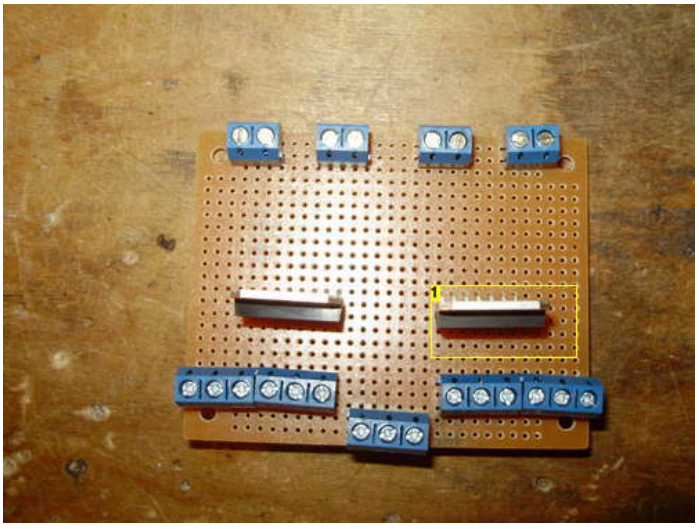
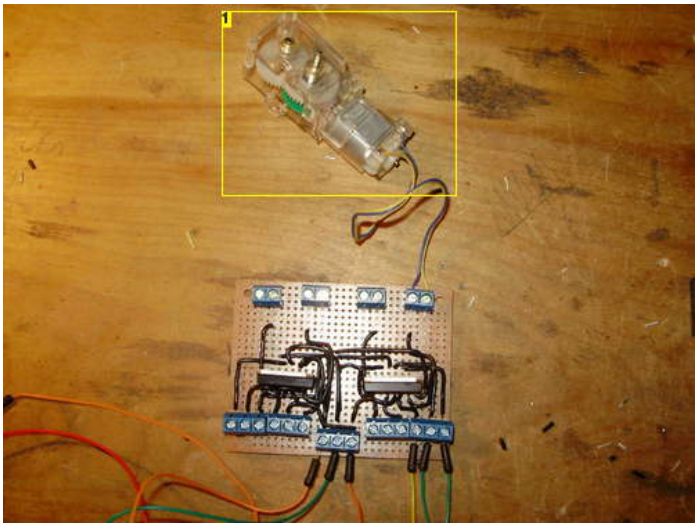


Image Notes
1. The pins on the L298HN must be manually bent for the chip to stand upright

Image Notes
1. I removed this resistor, and wired the Current Sense directly to ground
2. Ignore this bit



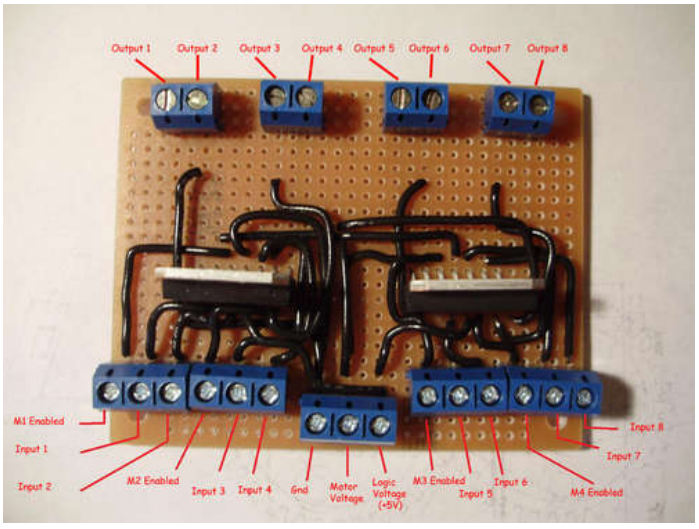
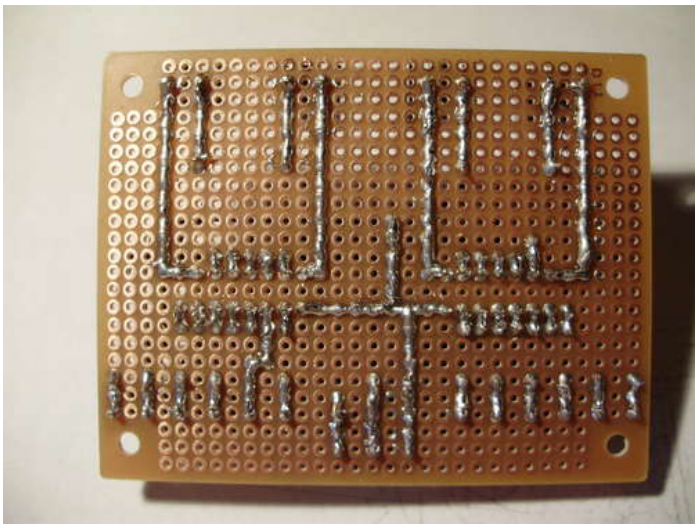
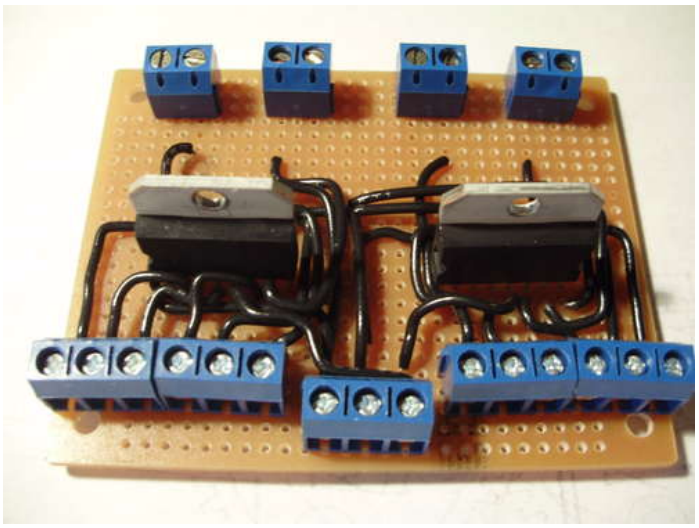
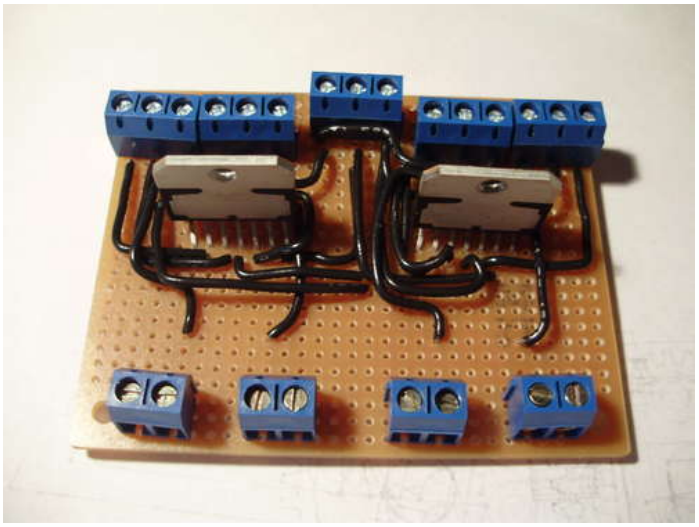
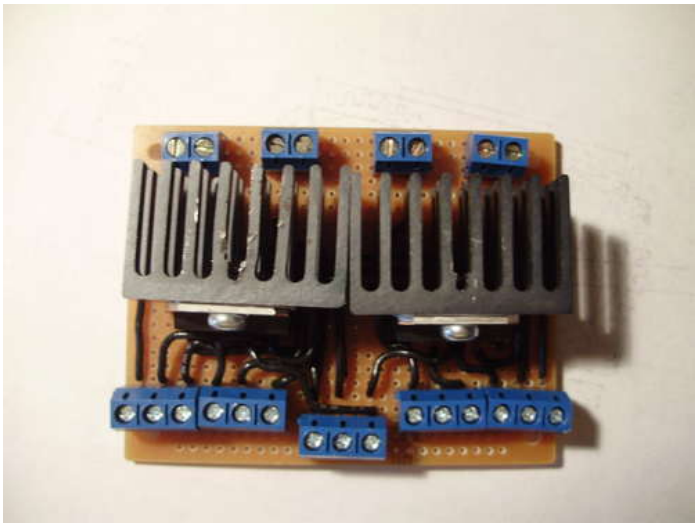


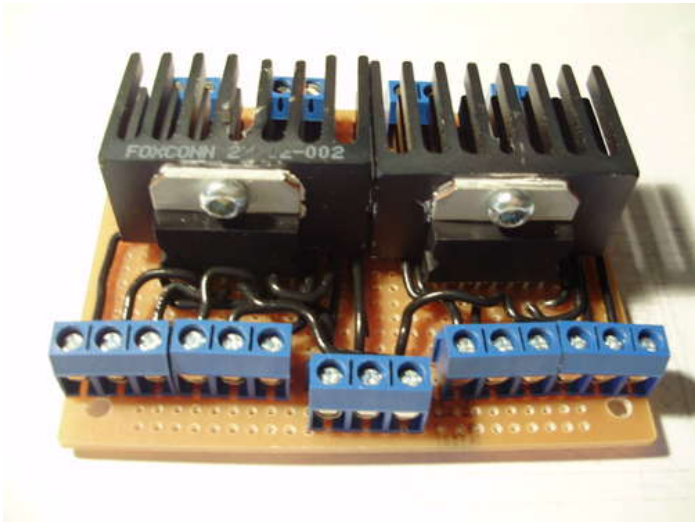
Image Notes
1. Testing with a cheap gearbox

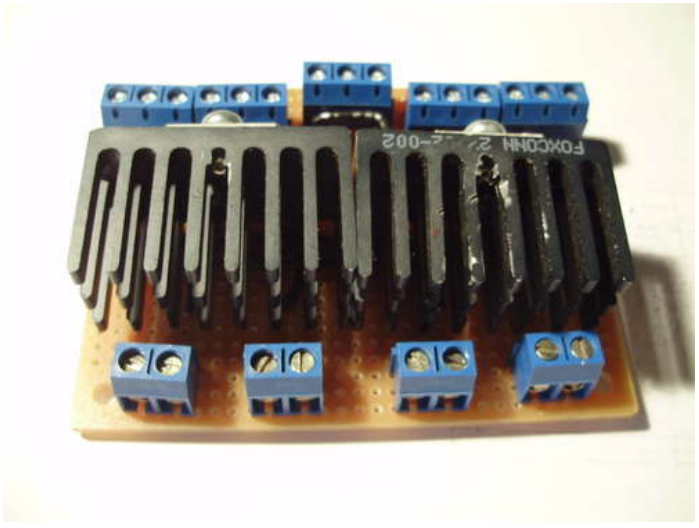


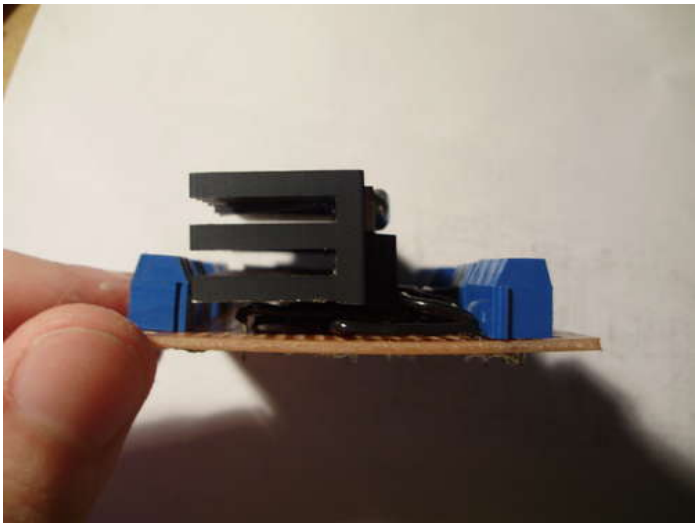







FOXCONN


FOXCONN



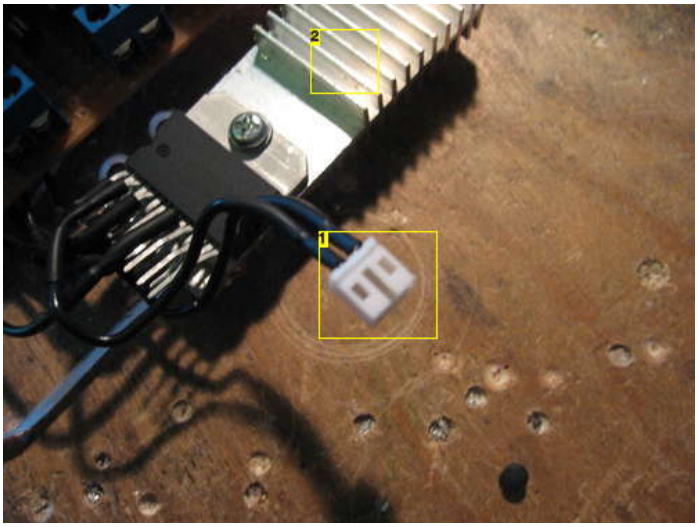


Image Notes
1. This goes to the 5th motor
2. This heatsink is part of one from an old ATX power supply.

Step 4: Power

The robot's power is supplied by two different sources. The Arduino and the motor controller logic circuitry are powered by a 9V battery, while the motors are powered by an approximately 12V Li-Ion battery pack.

I wanted to avoid having to buy an expensive LiPo/Li-Ion battery pack and charger, so I searched through my piles of electronic junk for a device with an appropriate battery. I settled on the battery from a 12" iBook laptop. The battery was 10.8V and 50Wh, but it was a little large and heavy for my needs. To fix this, I tore it open and had a look at the internals. I found that the battery was comprised of six 3.7 volt cells. These cells were organized in pairs of two wired in paralleled. The three pairs were then wired in a series, making a total 11.1V. To shrink the pack but keep the voltage, I simply removed one cell from each pair. The final battery pack had only half the capacity and half the discharge rate of the original (now only 2C), but the full voltage. I then wrapped the cells together with electrical tape so they would hold their shape, and soldered a quick-disconnect connector to the battery leads.

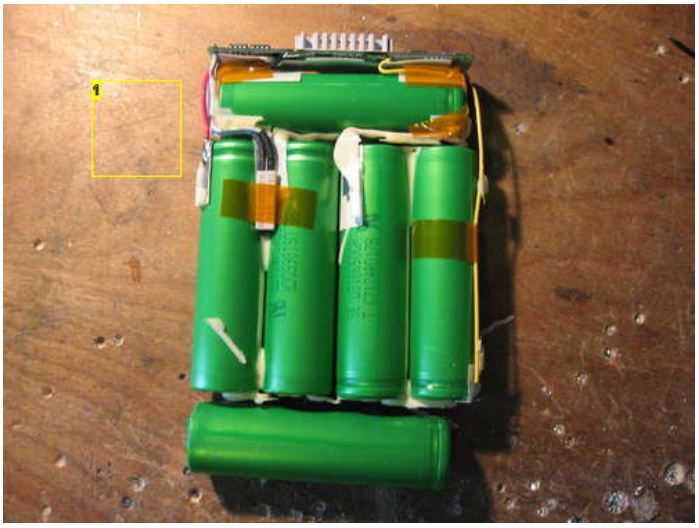


Image Notes
1. Casing removed

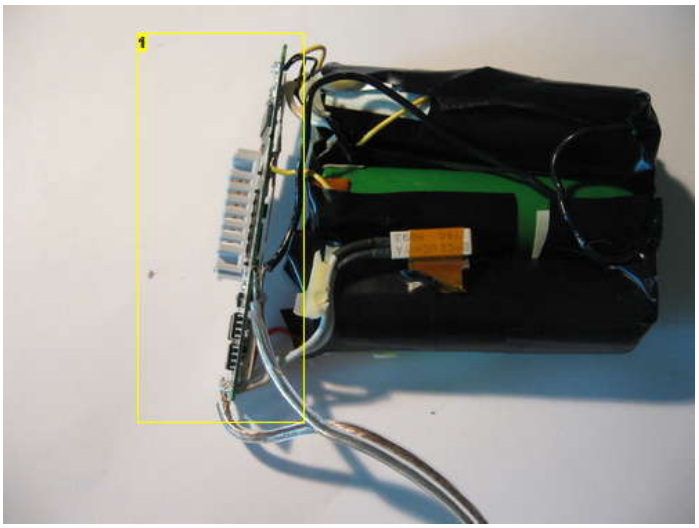


Image Notes
1. I left his bit so I could still charge the battery in the laptop

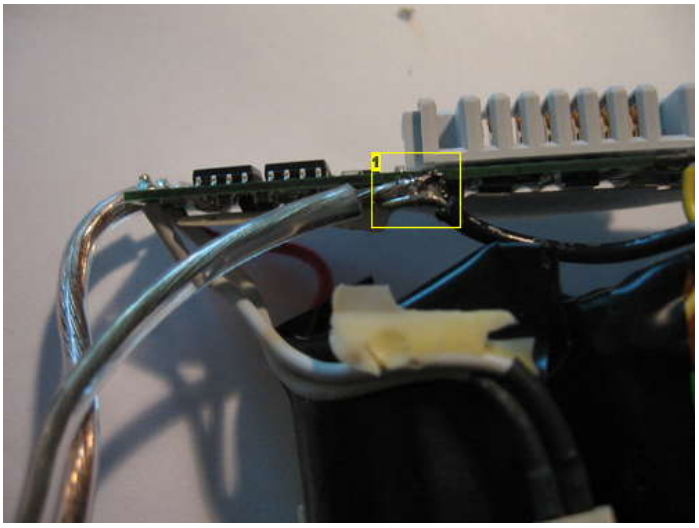


Image Notes
1. -

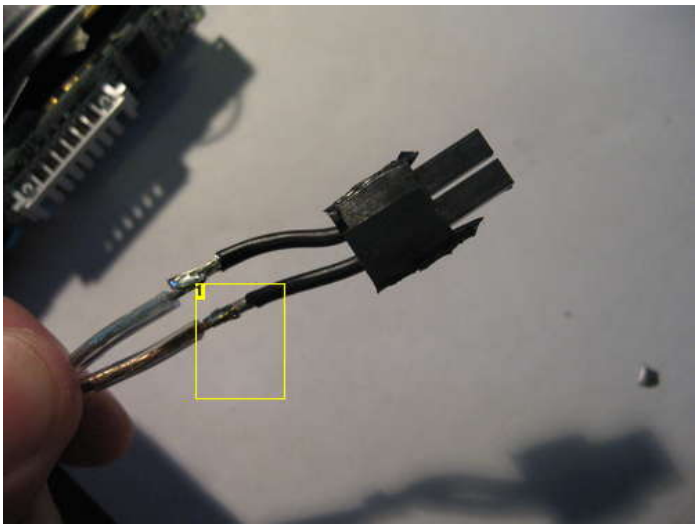
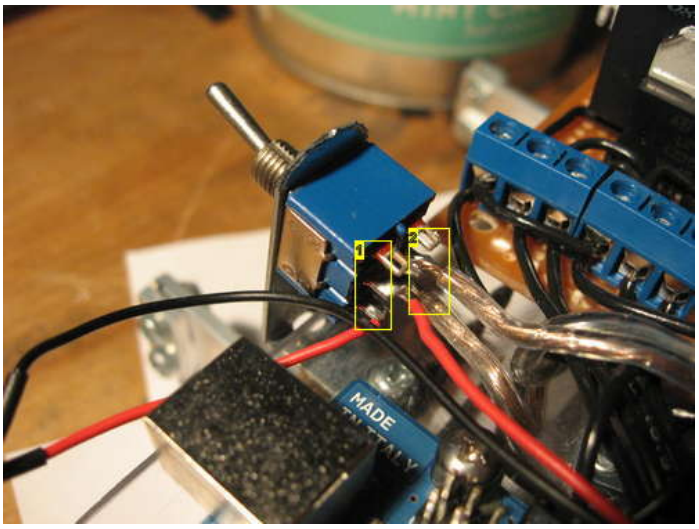


Image Notes
1. This was covered up later

Step 5: Power, cont.

The Arduino and the logic circuitry for the motor controller are both powered by a 9V battery. While the Arduino can take 9V input, the logic circuitry requires 5V, so I wired a 5V regulator in paralleled to the 9V going to the Arduino. Now, why did I not just take advantage of the Arduino's internal 5V regulator? Well, basically I ran out of pins, and I did not want to overdraw the Arduino. In addition to the regulator, I soldered a barrel power connector to the 9V end of the circuit, to fit into the Arduino. Finally, I added a DPDT toggle switch to break the 12V battery circuit as well as the 9V battery circuit.



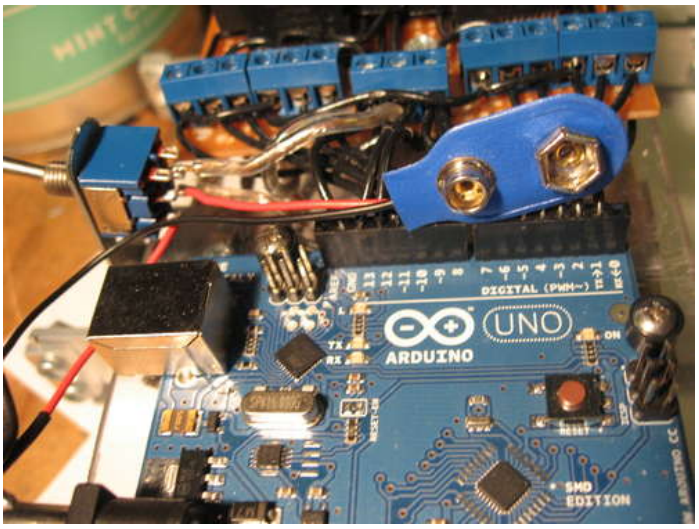


Image Notes
1. 9V breaker
2. 12V breaker

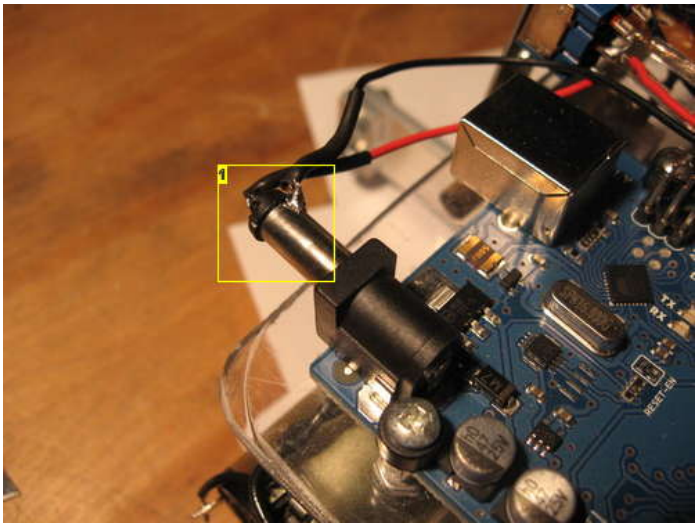


Image Notes
1. These barrel connectors can be scavenged from old wall worts

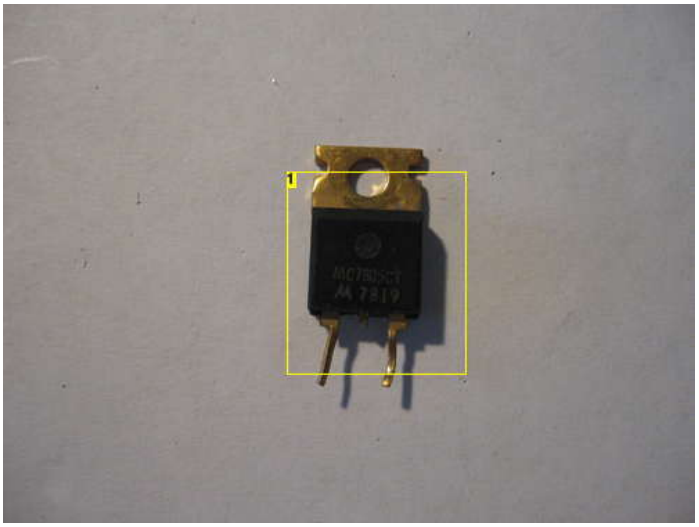
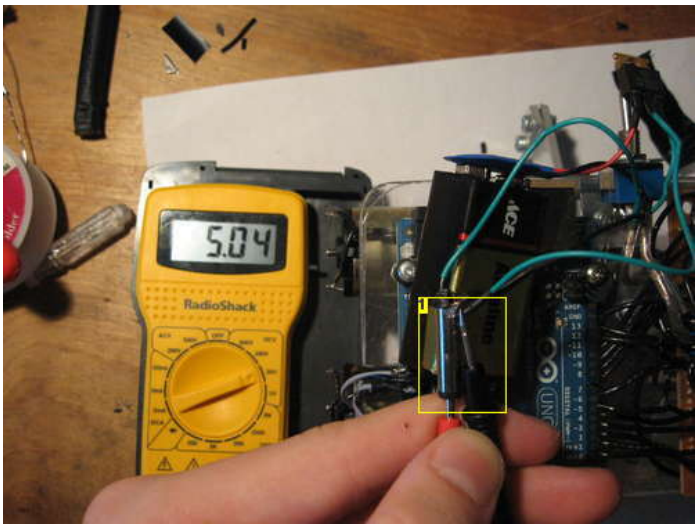


Image Notes
1. 5V regulator, up to 1A



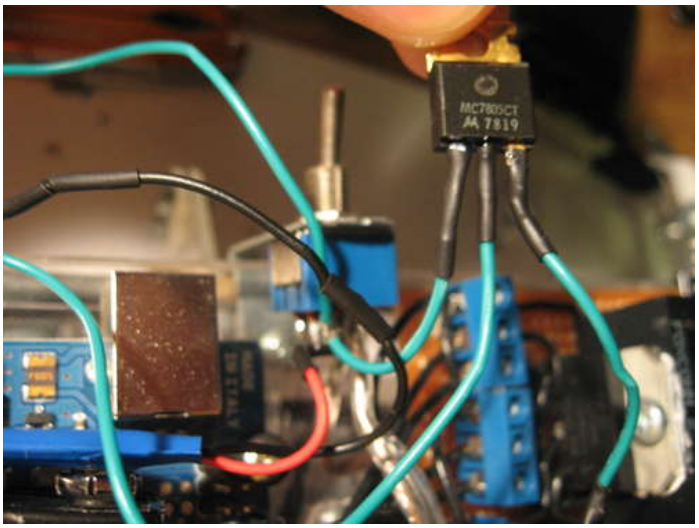


Image Notes
1. Ignore how this is wired, I did it wrong the first time around.

Step 6: Legs

The legs are some of the most important parts of this robot, because their design determines whether or not the robot can grip onto trees. I decided to have four pairs of legs, each pair controlled by one motor.

To make the legs, I cut four 8.5" lengths of the aluminum bar. I marked the segments 2.5" from each end. At those marks, I bent the aluminum at a right angle, to make a "U" shape. If you do not have a bending brace (which I don't) you can get a clean bend by clamping the aluminum with a c-clamp right on the mark, and pushing the unclamped end against a solid surface, like a work bench.

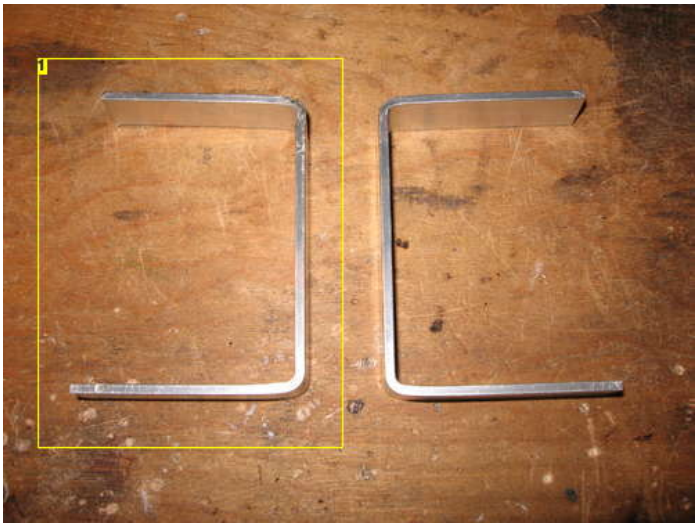


Image Notes
1. End product

Step 7: Feet

To grip the tree, the robot has very sharp feet at the end of its legs (where else?). The feet are made from jumbo-sized T pins, which you can get at your local fabric store. To fasten them to the legs, I made some clamps out of aluminum. I cut 8 3/8" or so lengths of aluminum, and filed a thin groove lengthwise into each of them, for the pins to fit into. Then I drilled a pair of holes into the aluminum, and corresponding holes into the ends of the legs. The clamps were then bolted down to the legs, with the pins inserted in the grooves. I left about 3/8" of an inch of the pins extending form the legs, but the length can be adjusted by loosening the bolts.

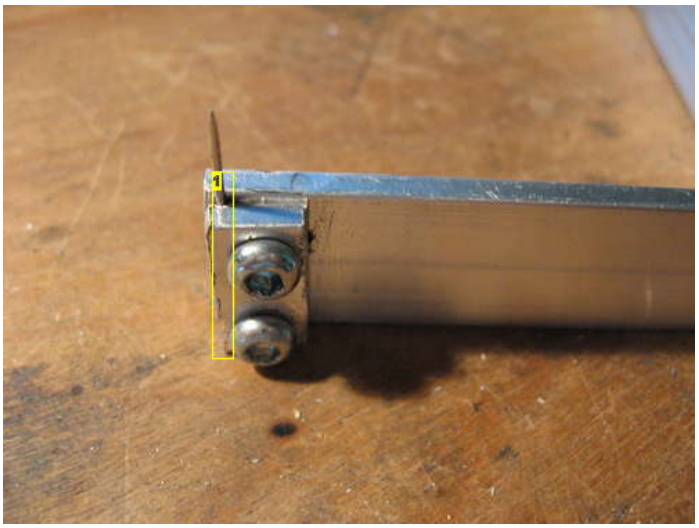


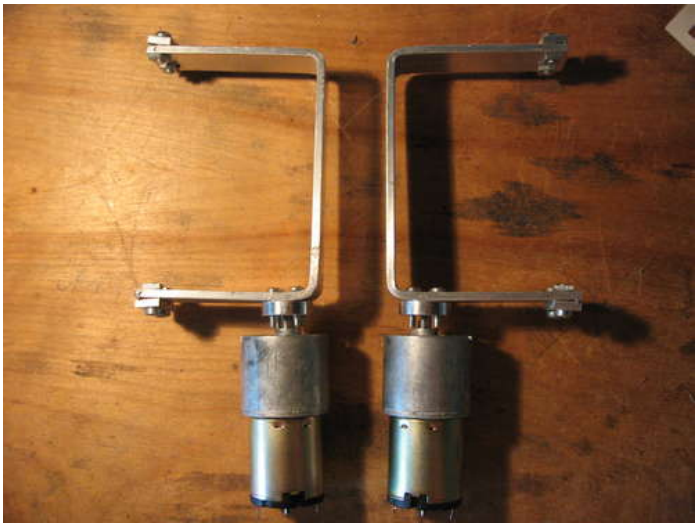
Image Notes
1. It is difficult to see from this picture, but there is a small groove down the clamp which holds the pin at a right angle to the leg.

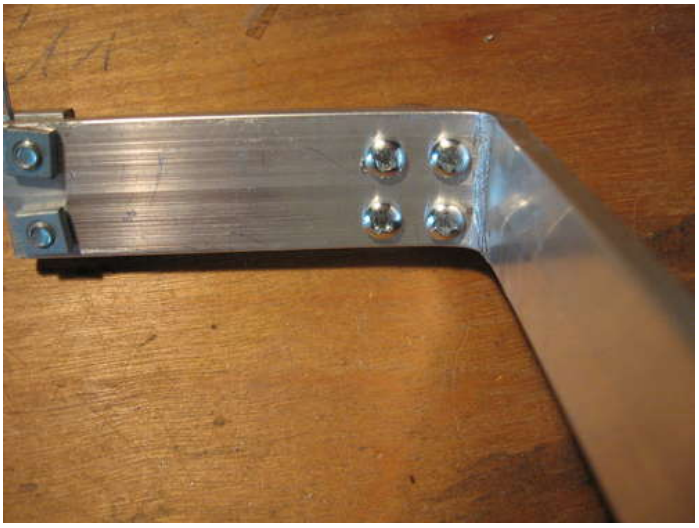
Image Notes
1. This is approximately how the legs will be positioned on the robot

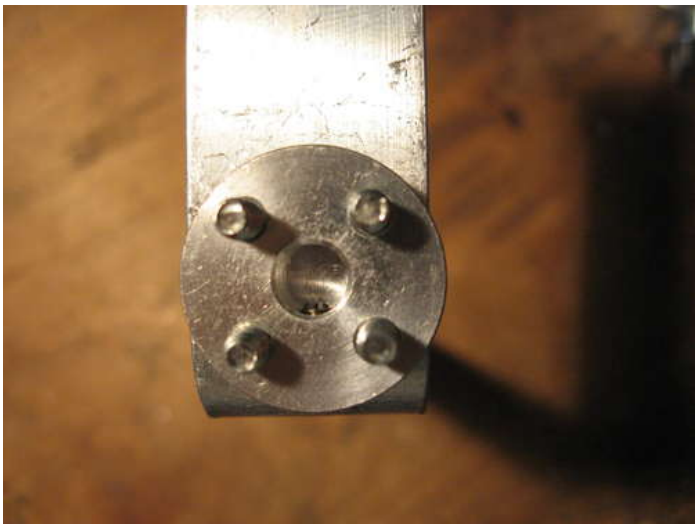
Step 8: Motor Hubs

The next step is to couple the legs to the motors. I found these handy 5mm mounting hubs , which were perfect for the job. I drilled four holes in a square on one side of each pair of legs, and screwed the hub to the legs with 4/40 screws. To fix the motors to the legs, you simply line up the flat side of the motor shaft with the screw in the hub, and tighten the screw.

Image Notes
1. I ended up switching these out for shorter screws







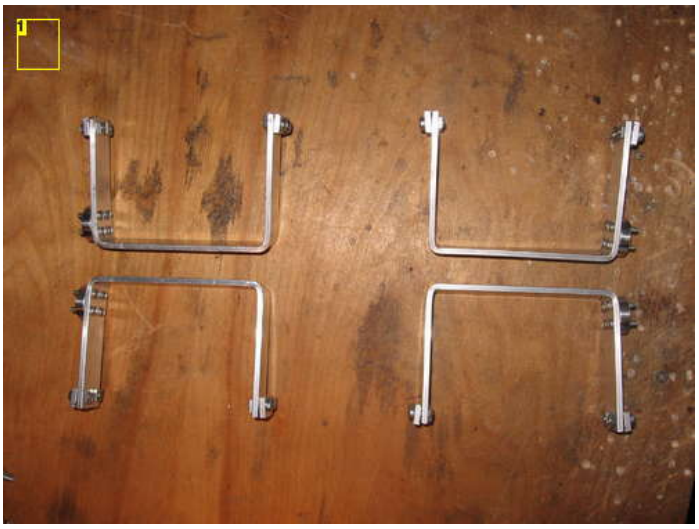


Image Notes
1. Approximate final layout of the legs

Step 9: Building the Frame

With the legs finished, the next step was to build a frame to hold the motors and legs together and in place. I started the frame by making a plate out of aluminum to hold the motors together. I drilled the plate to fit the screw holes of the motors and the gearbox shaft. The motors are held in place by 3/32 screws.

Next, I made a matching plate for the opposite side of the leg assembly. This plate holds the legs straight while they turn. I drilled holes through the legs, opposite to the motor hubs. Then I bolted the legs through the plate with washers and a locknut to hold them in place and let them spin freely on the bolt.

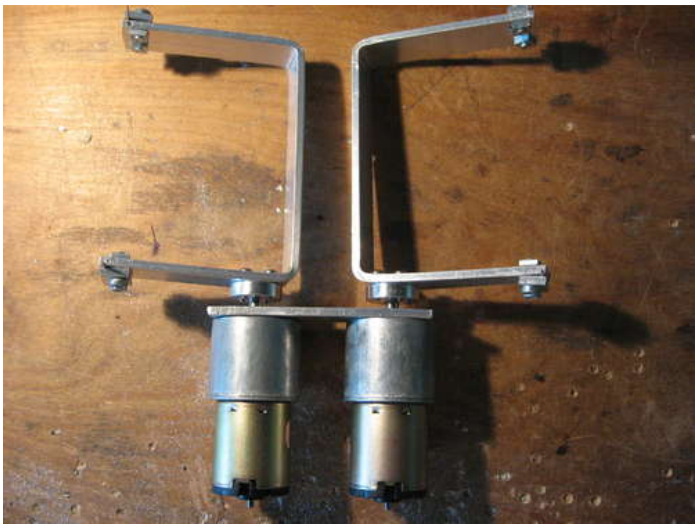
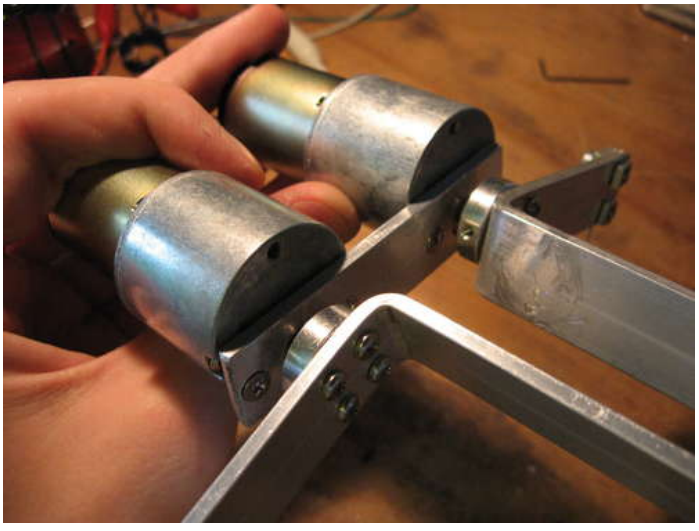
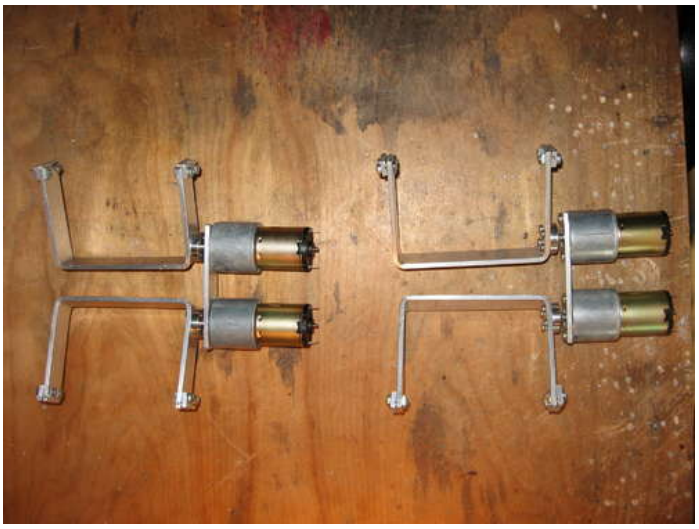


Image Notes

1. I countersunk the screws, because the heads were beveled.





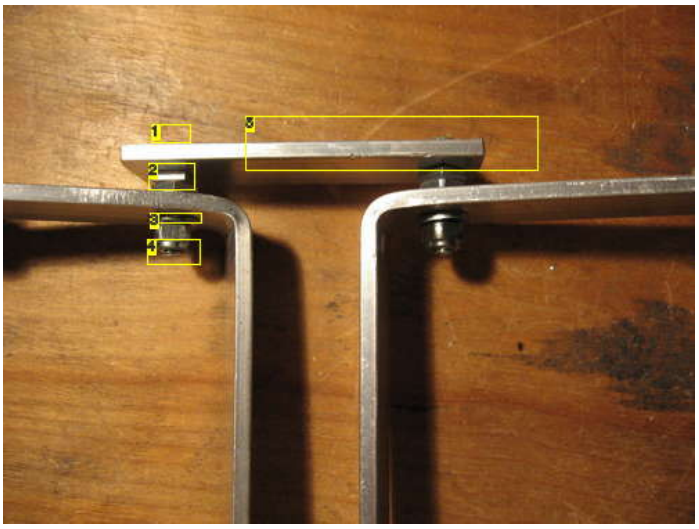
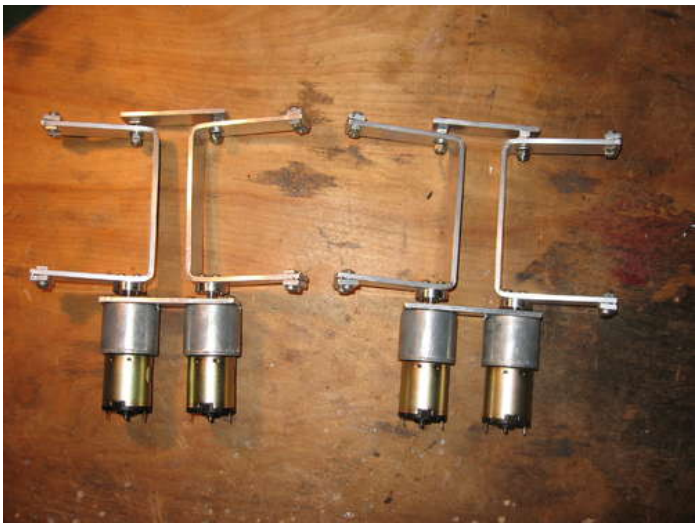


Image Notes
1. Bolt head
2. Nuts fasten the bolt to the plate
3. Washer
4. Locknut holds the washer on
5. Second plate



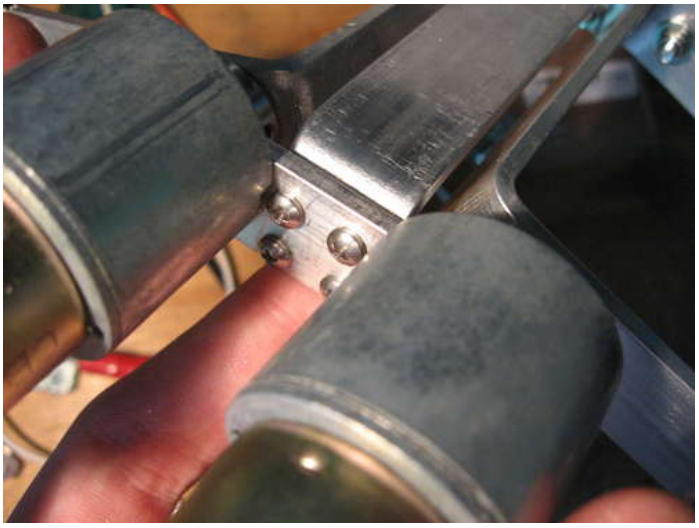
Step 10: Frame, cont.

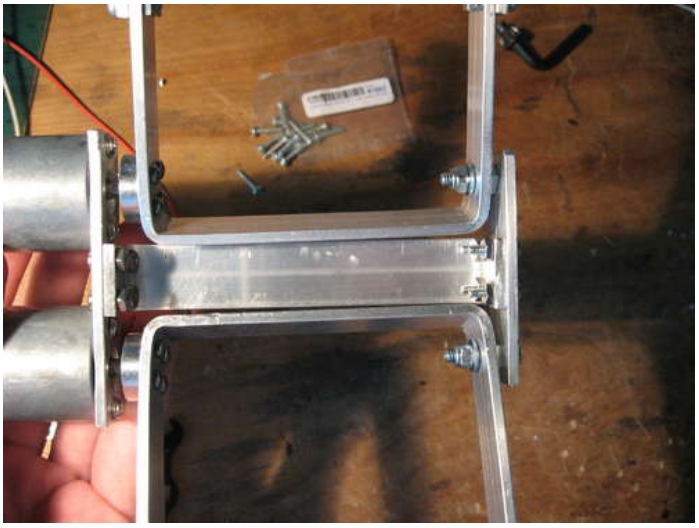
Next, I made a piece out of aluminum to fix the two opposite plate together. This piece sits between the pair of legs on each assembly, and is the primary structural support of each segment. As well as holding the robot together, it provides a place to mount the electronics and other components later on.

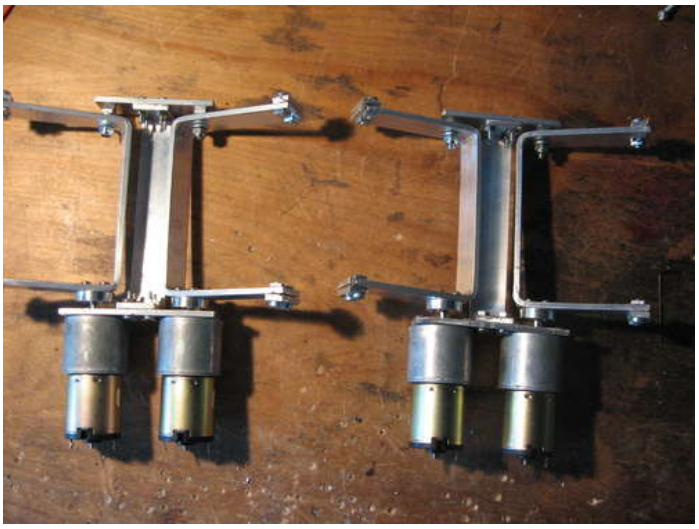
I bent the aluminum at right angles using a c-clamp, and drilled four holes in each end. I drilled matching holes in the motor plate and the opposite plate in each leg assembly, and then bolted everything together with 4/40 screws.

Once both segments of the robot were built and structurally sound, I could test their tree-gripping ability by hooking the motors up directly to a battery. Fortunately, they worked quite well, or I would have had nothing else to share.

Image Notes
1. 2 of these







Step 11: Electronics Platform

To hold the electronics, I cut an approximately 6" x 3" piece of acrylic. I drilled six holes in it and screwed standoffs into the holes, to support the Arduino and the motor controller. Then I drilled four holes in the top of the leg assembly, and bolted the acrylic to the assembly, with spacers to lift it up above the motors. Finally, I screwed the Arduino and motor controller into the standoffs.

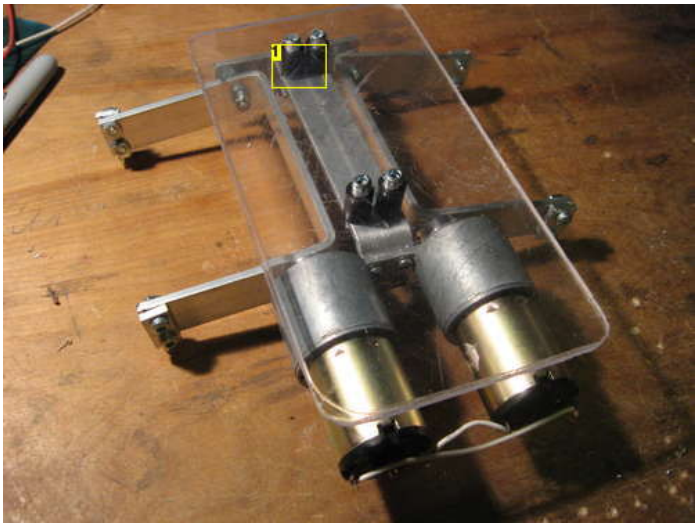


Image Notes
1. I actually eventually had to ditch these spacers. Read on.

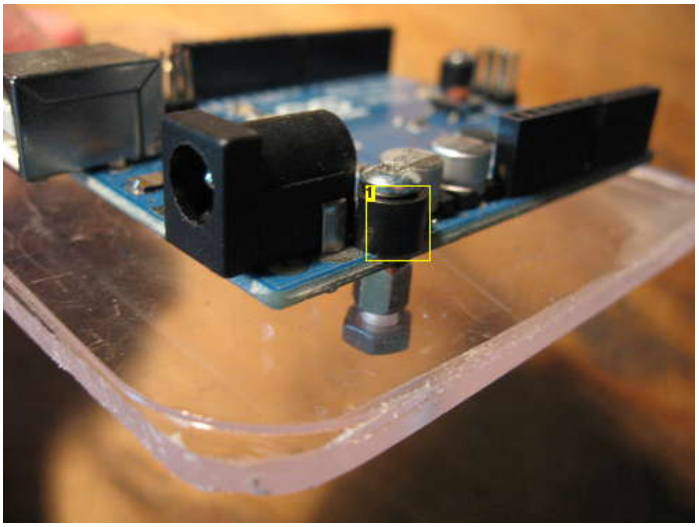
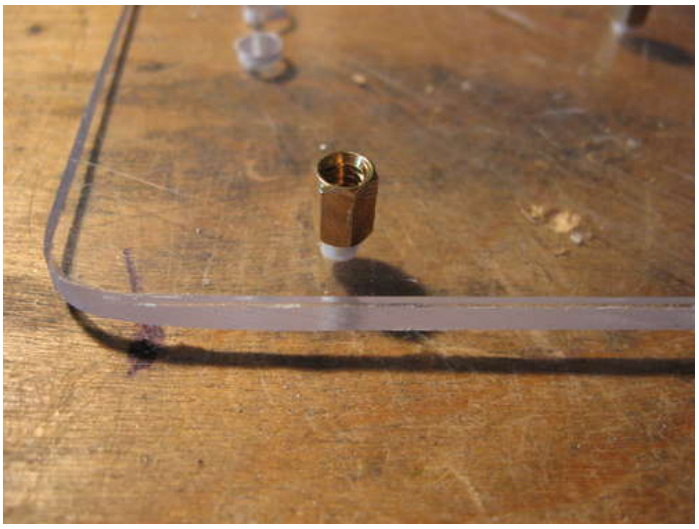
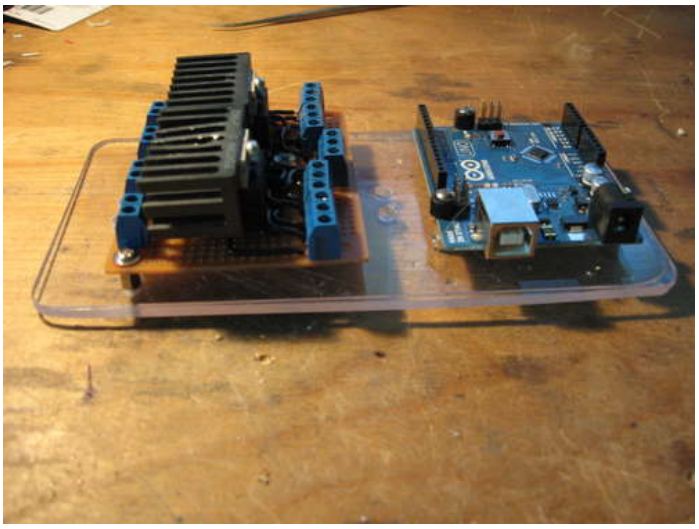
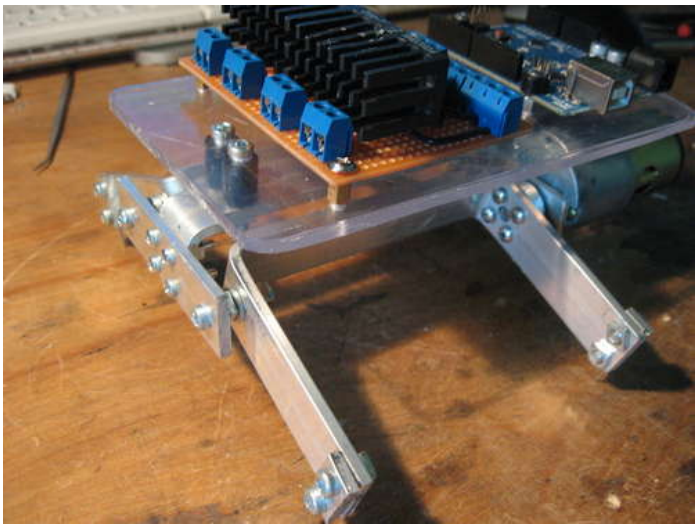


Image Notes
1. I used a spacer so the head would not short anything







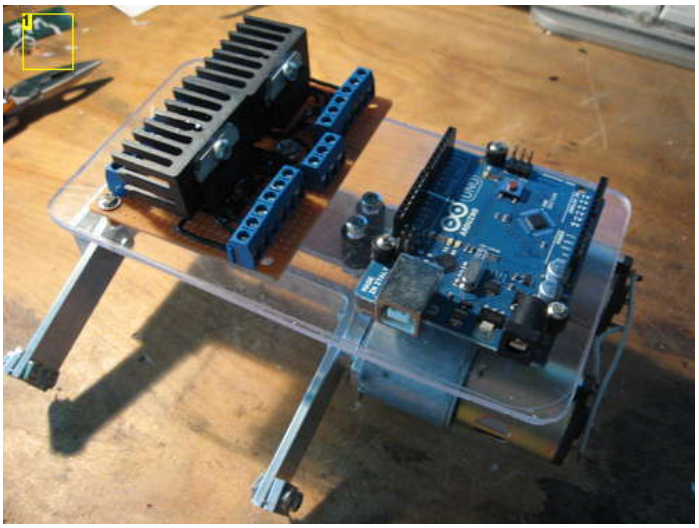


Image Notes
1. Starting to look like a real robot!

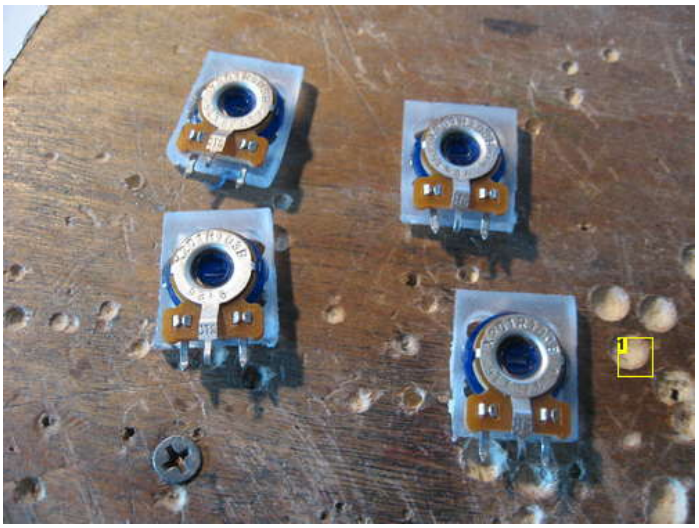
Step 12: Rotation Sensors

Rotation sensors are key to the operation of this robot. It has one rotation sensor per motor, so the robot knows the exact position of each leg at all times, allowing for precise control of the legs. For my rotation sensors, I used four very thin trimpots I had lying around. Pots are extremely easy to interface with the microcontroller, and are plenty precise for my purposes. They were not, however, very easy to interface with the hardware of my robot.

While designing and building the leg assemblies, I neglected to build in an easy way to connect the potentiometers to the legs. In the solution I came up with, one side of the pot is fixed to the inside of the leg by the protruding screw heads. The other side of the pot is fixed to the locknut on the end of the bolt that holds the leg in place. When the leg turns, the side of the pot fixed to the leg turns, while the side fixed to the locknut is held in place.

To interface the pots and the legs, I first sanded the plastic side of the pots flat. I took four squares of acrylic, approximately 3/4" on each side, and drilled four holes in each, corresponding to the four screw heads in each leg. Then I glued a potentiometer to the center of each acrylic square.

To fix the opposite side of the pots to the locknut I had to get even more creative. First, I glued metal standoffs scavenged from a PowerMac G5 case to the metal side of the pots. Then I glued the plastic shaft from a Bic pen to the metal side of the pots. The other ends of each pen were cut to fit within the metal legs. Then the pen shaft was forced over the square locknut and epoxied to it.



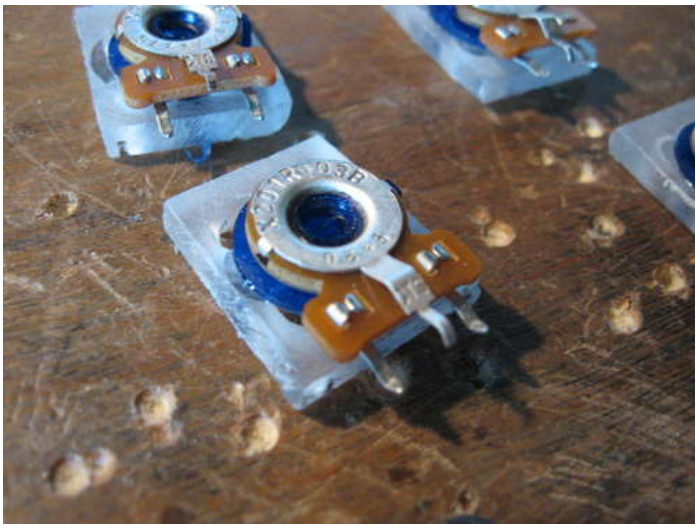


Image Notes
1. My poor desk!

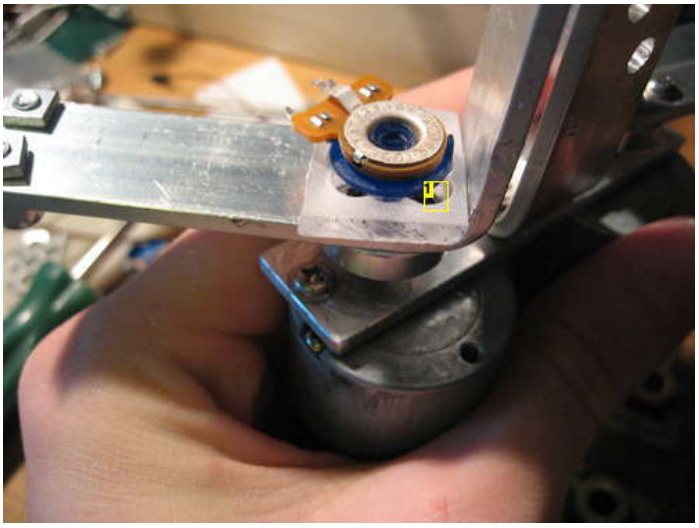


Image Notes
1. Screw heads are under there

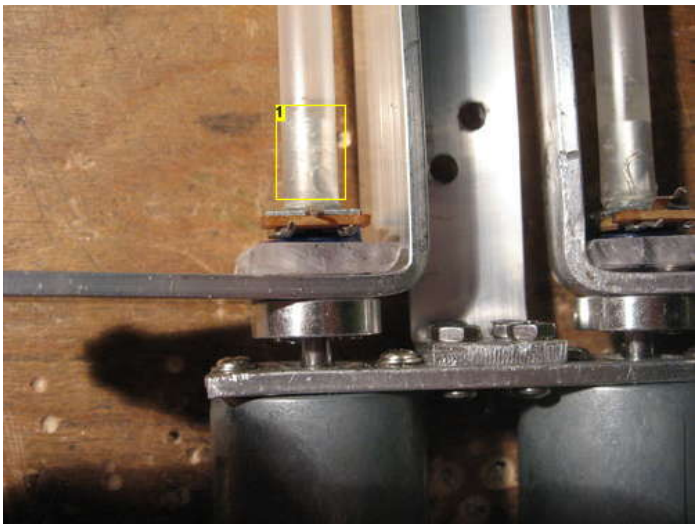
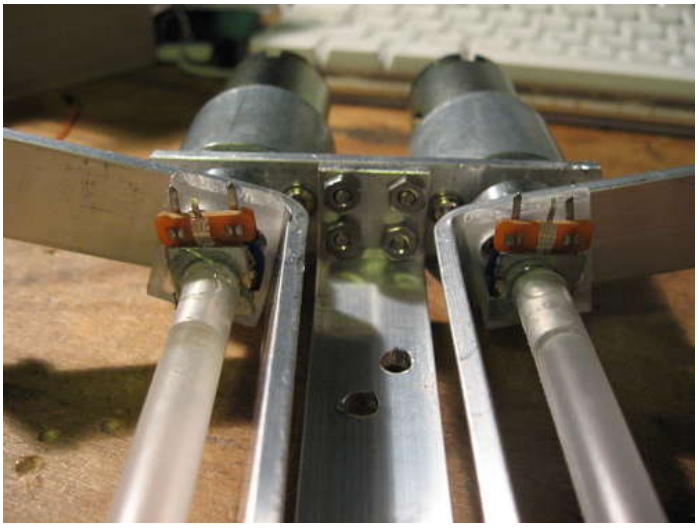
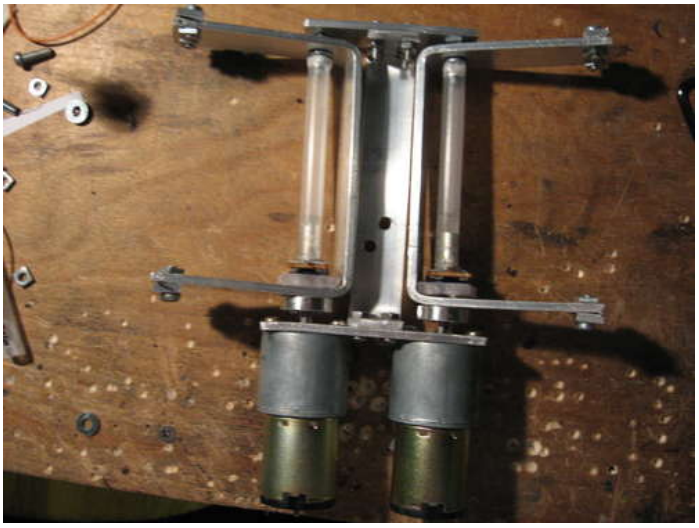


Image Notes
1. The standoff is in there

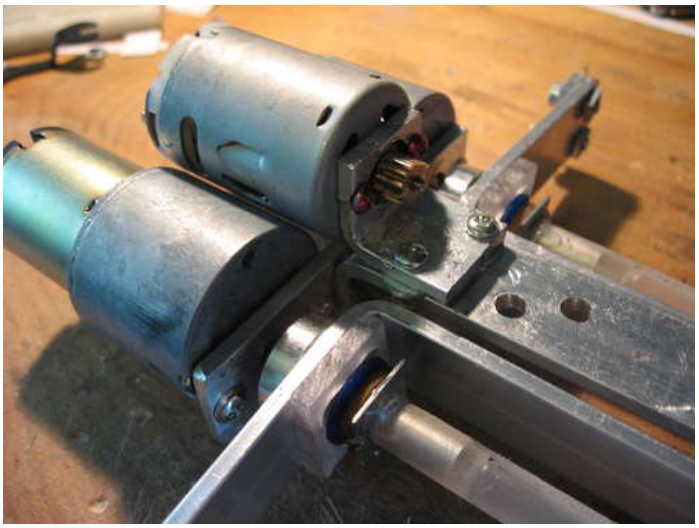


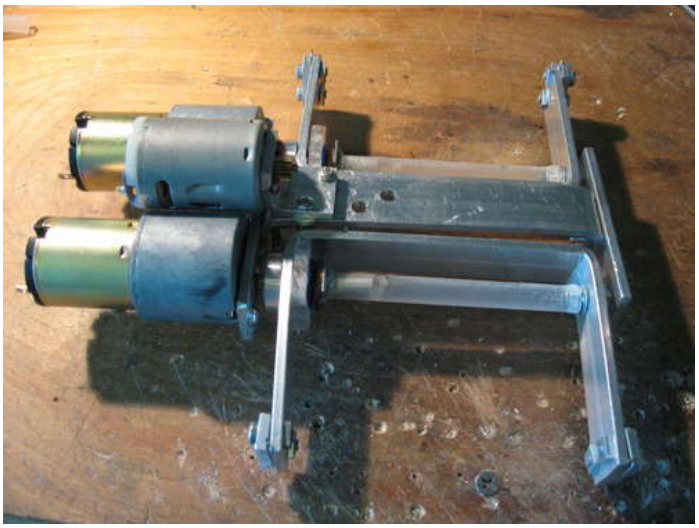


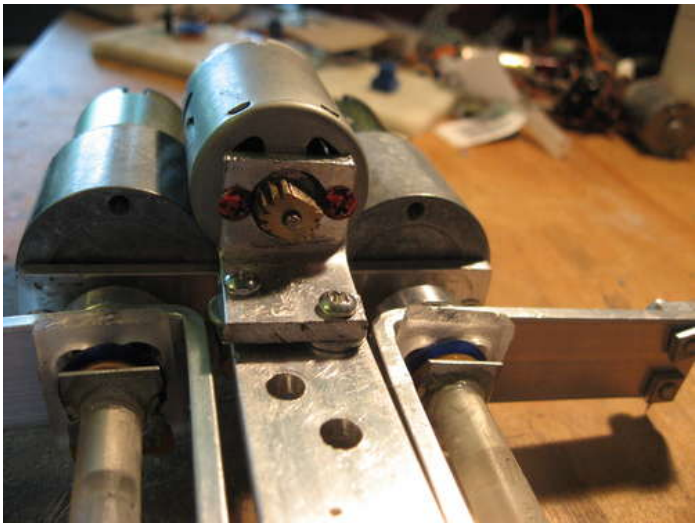
Step 13: Backbone Motor

To move up and down a tree, the robot extends and contracts by spinning a threaded rod that is fixed to the top segment. When the rod is spun clockwise, the two segments are pulled together, and the are pushed apart when it spins counter clockwise. To spin the rod, I needed a relatively high-torque low-speed motor that would run at 12V, and I happened to find just such a motor in my box of parts. This particular motor came fitted with a brass gear. To assist with coupling the motor to the threaded shaft, I filed two sides of the gear flat.

To mount the motor to the robot, I bent a short length of aluminum to an "L" shape. I drilled a large hole out of the center of one of the faces (for the motor shaft and gear) and two small holes in both faces for bolting the motor to the metal and bolting the metal to the robot. I drilled corresponding holes into the back of one end of the segment of the robot without the electronics, so that the motor was positioned between the two legs.



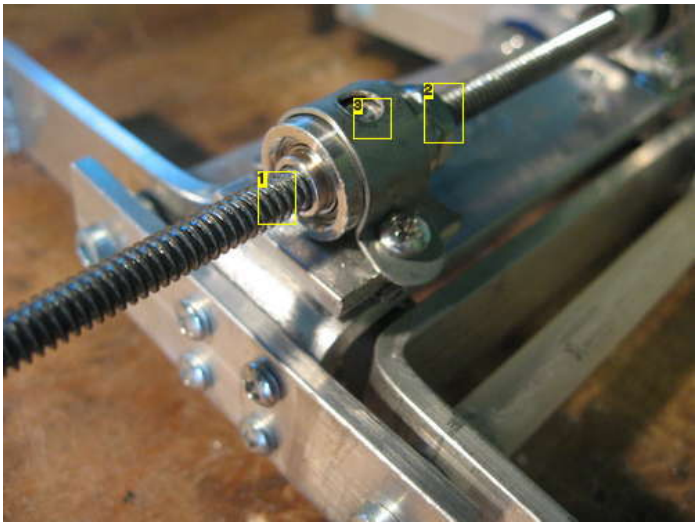




Step 14: Mounting the Spine

There are a couple problems with getting a threaded spine to spin smoothly. First, it must be coupled to the motor well, and second it must have some sort of bearing fixed to it on which it can spin. One of the first things I found while trying to couple the motor shaft to the threaded rod was that the connection should be flexible for smooth operation. I made my coupling out of two segments of clear nylon tubing. A wider diameter segment fits tightly over the gear attached to the motor shaft, while a thinner segment fits into the larger tubing and tightly over the outside of the threaded shaft. The coupling is secured with a zip tie.

With only the coupling, the threaded shaft still can not bear any load, because it would just pull off the motor. To support load, I made a bearing for the shaft out of an old hard drive read/write head bearing. I drilled out the center so that the threaded rod could pass through it. I then fed the rod through it, and fastened a nut on each side of the bearing, to hold the threaded rod in place. I then bolted the bearing down to the back of the robot's frame.



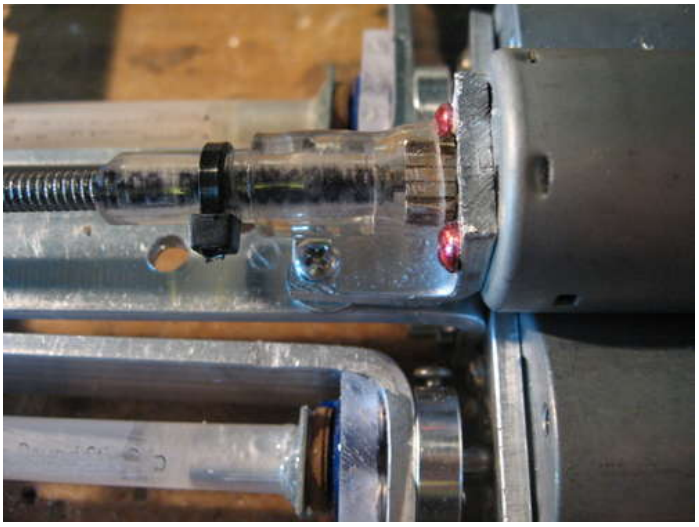


Image Notes

1. A nut will go here
2. Nut
3. This is a bit of scrap Meccano metal

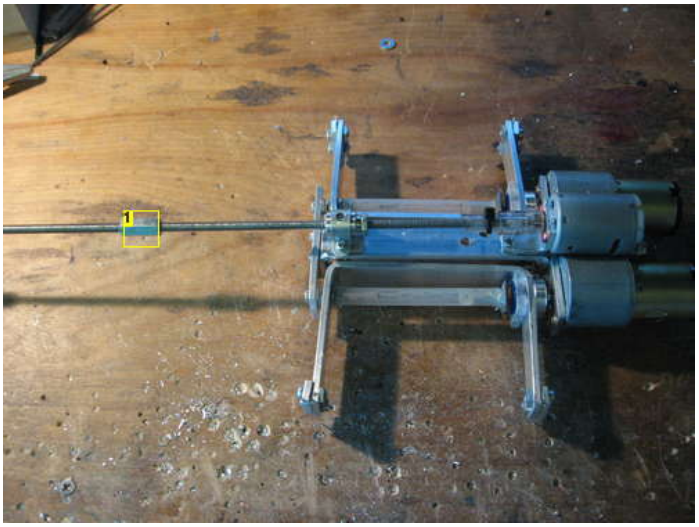


Image Notes
1. The other segment will be fixed to this nut

Step 15: Mounting the Spine, cont.

For the spine to work, it must pass through a nut that is fixed to the other segment of the robot. For ease of mounting, I used a large 1/2" long nut. To fix it to the segment of the robot, I cut two ~4" lengths of aluminum, drilled them to match the bolts that hold the acrylic piece, and mounted them through the screw holes, across the frame. These will later become supports for linear slides. I then drilled four more holes around the center of the back of the robot, with enough space between them for the nut to fit. I cut a piece of aluminum to run along the back of the robot, and drilled it to match the holes. I then placed the nut on the back of the robot, placed the bar of aluminum on top of it, and bolted through the bar, to sandwiched the nut between the two pieces of aluminum. Finally, I threaded the rod through the nut.

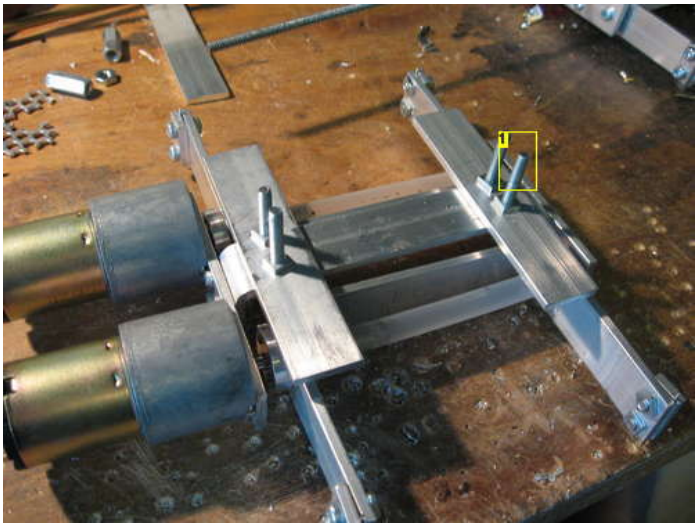


Image Notes
1. These will go through the holes in the acrylic

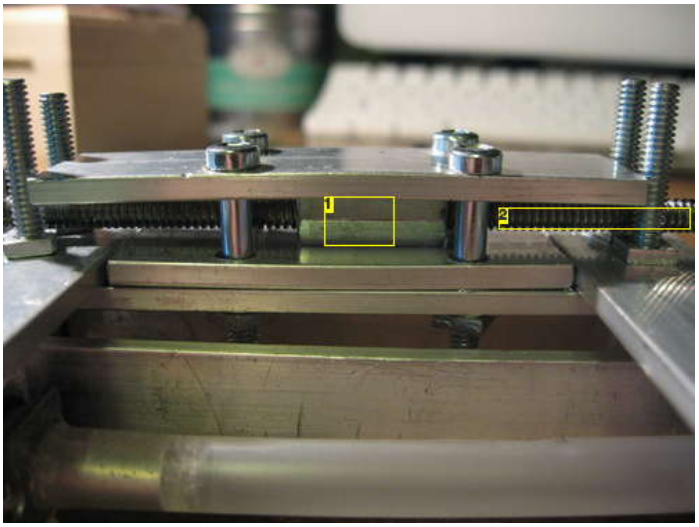
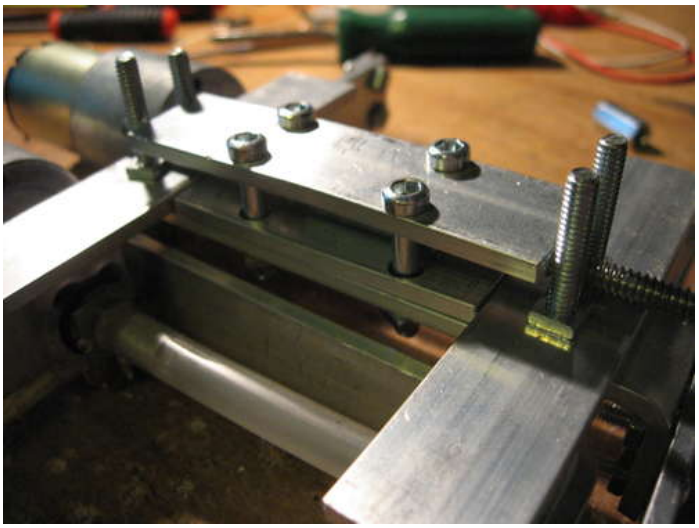


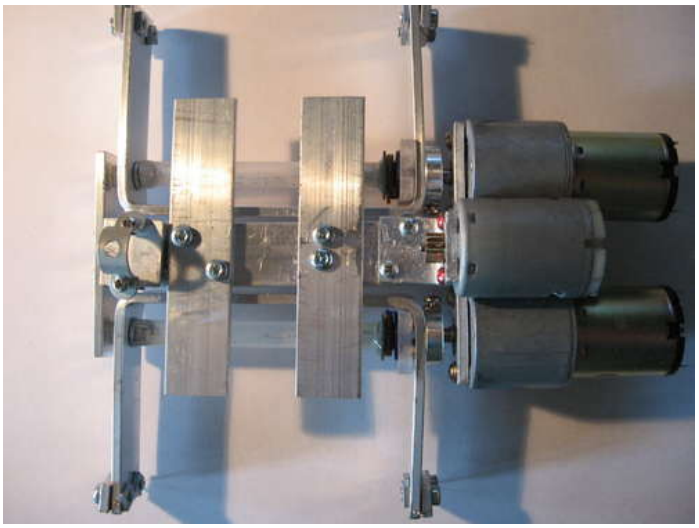
Image Notes
1. Nut
2. Threaded rod



Step 16: Linear Slides

Without something holding the two segments of the robot in the same plane, the top segment would turn when the threaded rod turned, instead of moving up or down. To keep the two halves of the robot in the same plane, I built linear slides out of two steel rods and brass tubing.

First, I added a pair of aluminum bars to the segment of the robot without the electronics, to match the pair on the other segment. To mount the steel rods and the brass tubing to these, I made a clamp system similar to the clamps holding the feet in place. To do this for the large diameter rods, I first clamped two 3/4" squares of aluminum together. I then drilled a 1/8" hole down the intersection of the squares, and then took them apart. I drilled two holes in each square, and corresponding holes in each supporting arm. Then I repeated the process four times. To get the slides perfectly parallel to the threaded rod, I had to bend up the supporting arms on the non-electronics segment of the robot.



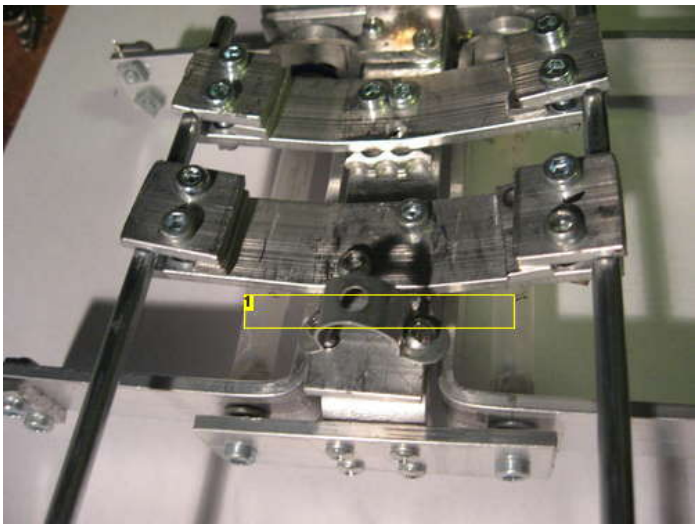
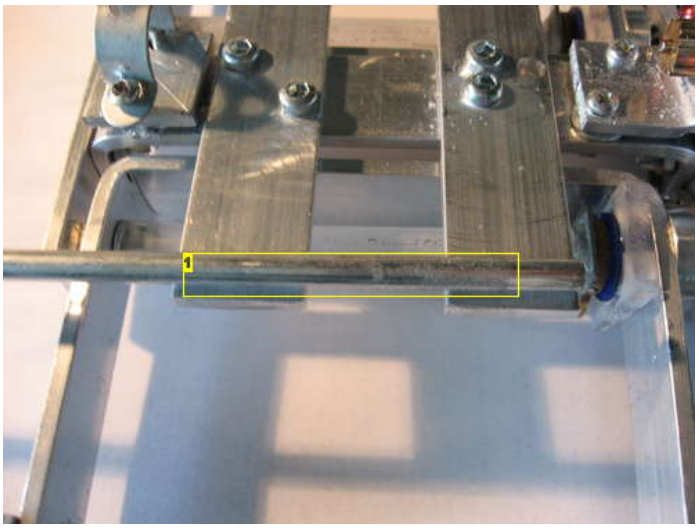


Image Notes
1. You can see the bend in the arms



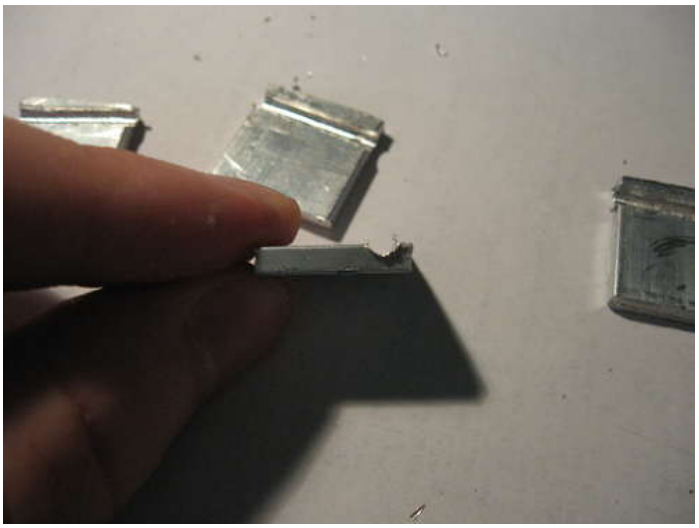
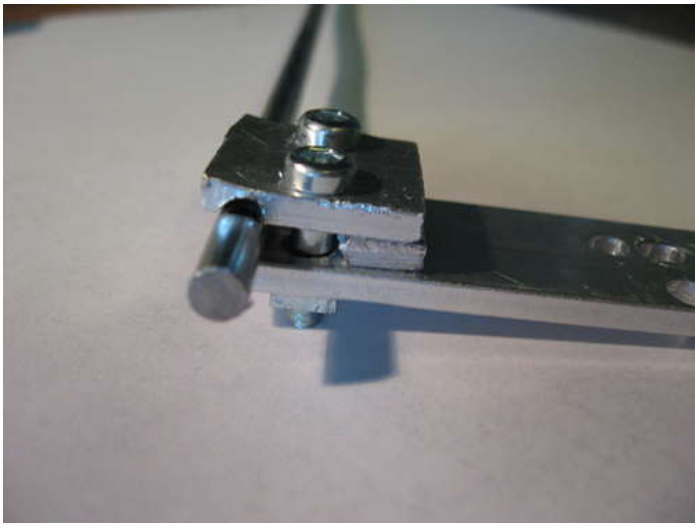
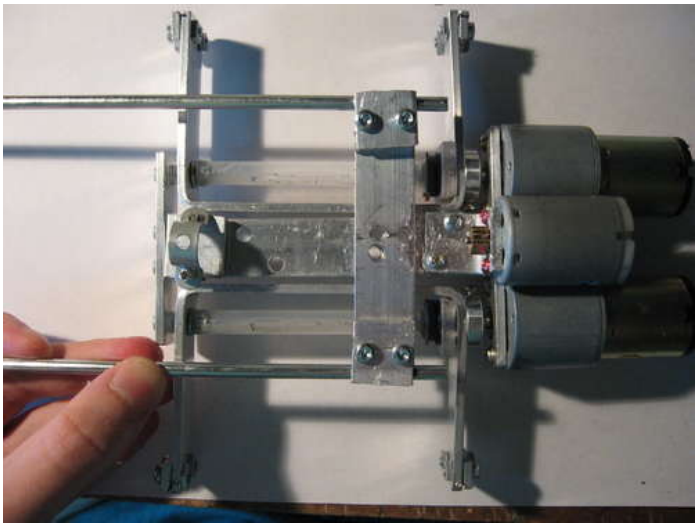
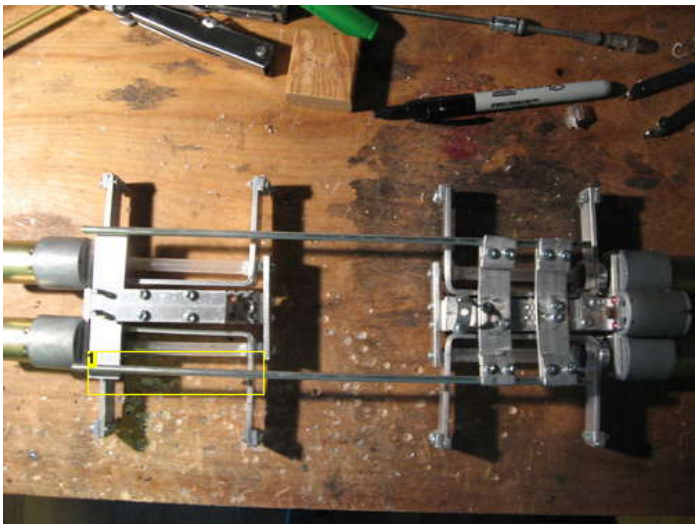


Image Notes
1. This needs to be fixed in this position







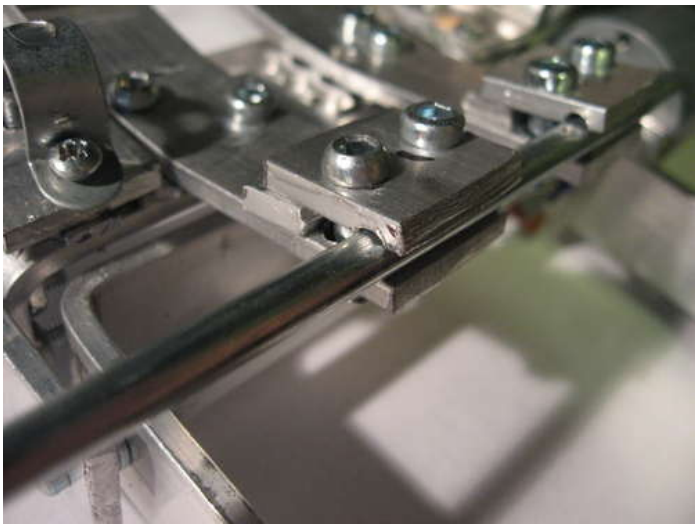


Image Notes
1. Brass tubing will go right here

Step 17: Wiring the Robot

The next step is to wire all the electrical components of the robot together. I started out by soldering long wires to the contacts on the motors. I twisted the wires together by chucking one end in an electric drill and holding the other end with pliers (a trick I learned from the Ben Heck show). Next, I wired together the pots on the legs. I did this using segments of ribbon cable from an old IDE cable. I wired the pots so that they all had a common ground and input voltage. The input voltage was connected to the +5V pin on the Arduino, and the four signal wires were soldered to headers and then connected to analog inputs A0 - A3 on the Arduino.

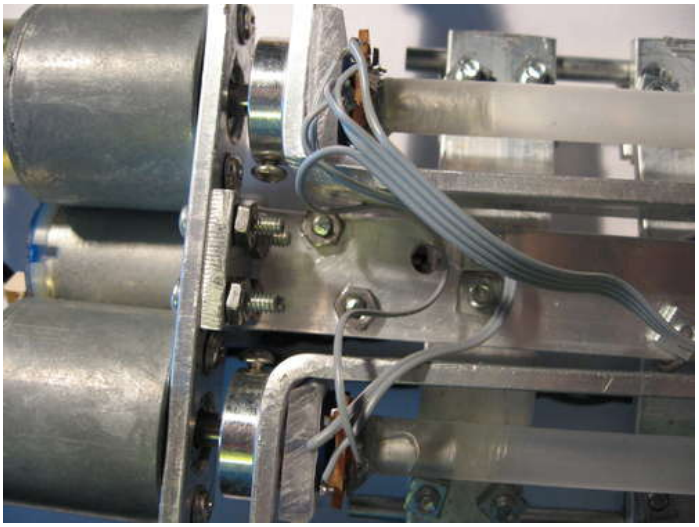
Because this robot is autonomous, I needed a method for controlling the robot's actions so that I could get it to release from the tree. For this, I just used a simple slide switch connected to a digital input on the Arduino

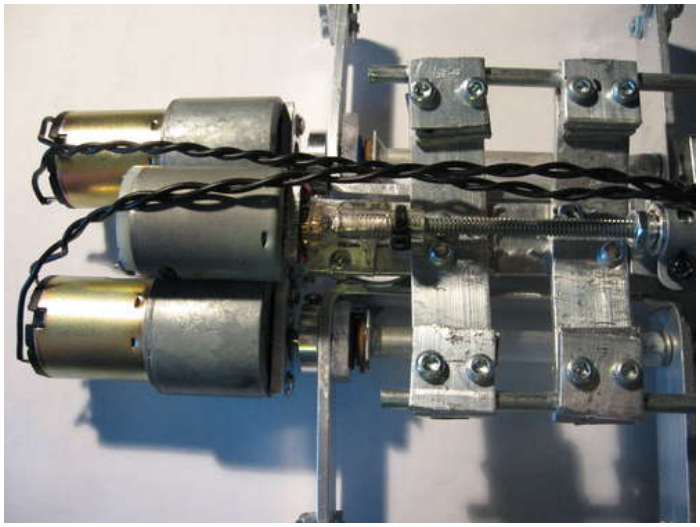
Next, I wired the digital output pins on the Arduino to the inputs on the motor controller. First, I connected all the motor enabling pins on the motor controller to eachother. The rest of the wiring went as follows:

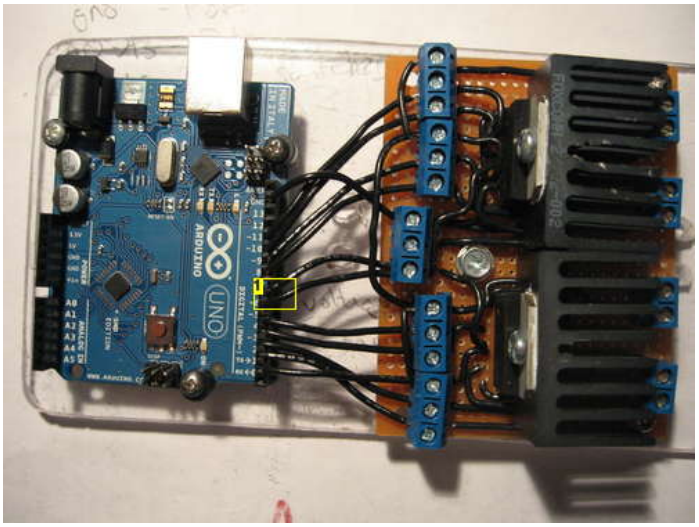
1. Enable Motors
2. Motor 4 Input 2
3. Motor 4 Input 1
4. Motor 3 Input 2
5. Motor 3 Input 1
6. Control Switch
7. empty
8. Motor 2 Input 2
9. Motor 2 Input 1
10. Motor 1 Input 2
11. Motor 1 Input 1
12. Motor 5 Input 2
13. Motor 5 Input 1

I then connected the motor's leads to the terminal strips on the motor controller, and connected the motor voltage terminal to the 12V battery pack, via a toggle switch. I connected the 5V regulator to the logic voltage terminal, via the same toggle switch.

I collected the umbilical cord of wires running between the two segments of the robot into a bundle, and fastened them together with zip ties and electrical tape, to keep them organized.







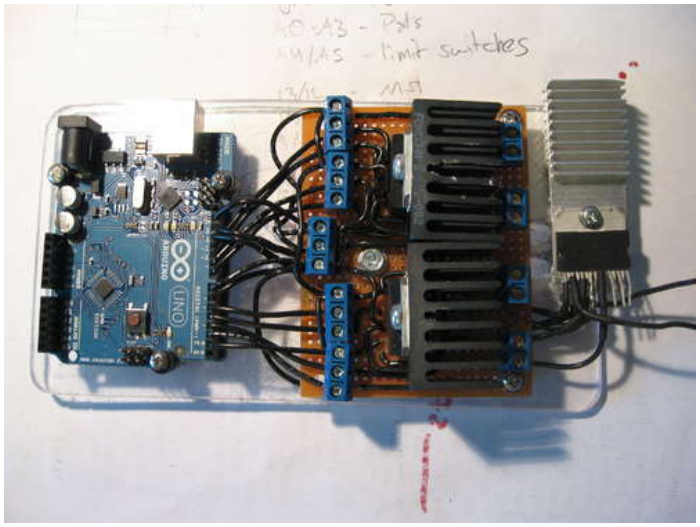
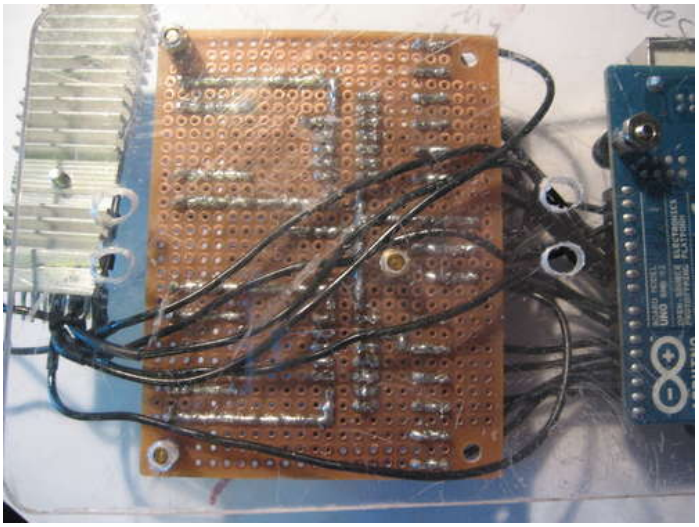
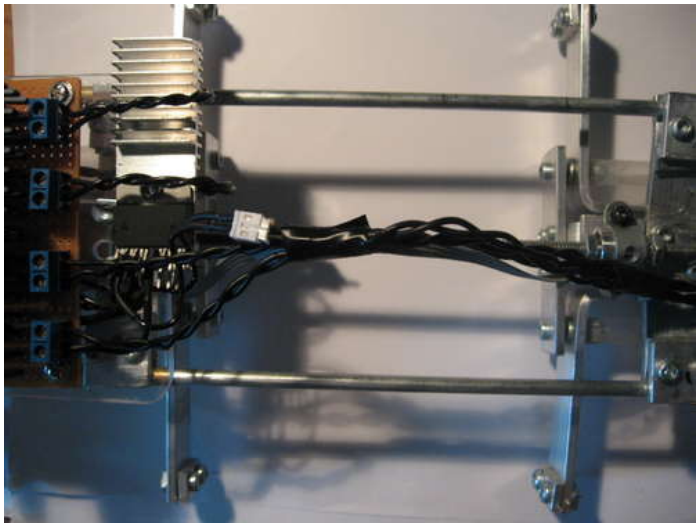
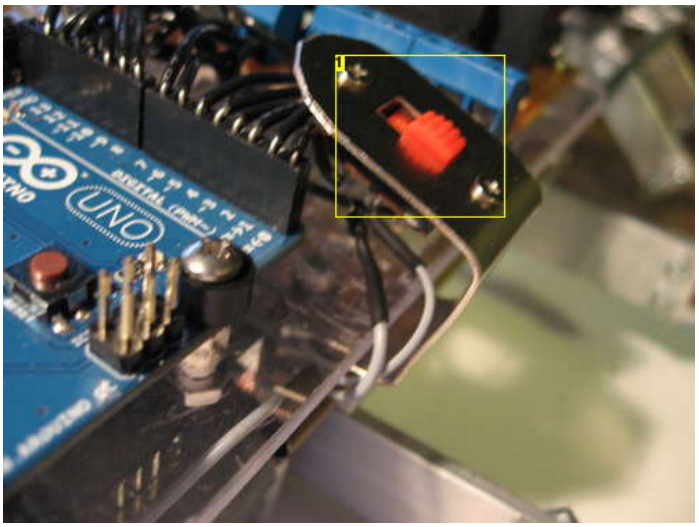


Image Notes

1. Ignore these. I removed them when I though to power the logic voltage off the







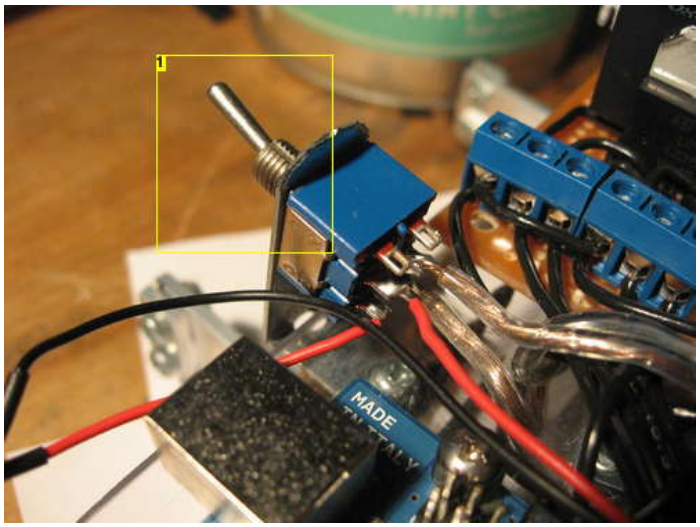


Image Notes
1. Control switch

Image Notes
1. On/Off Switch

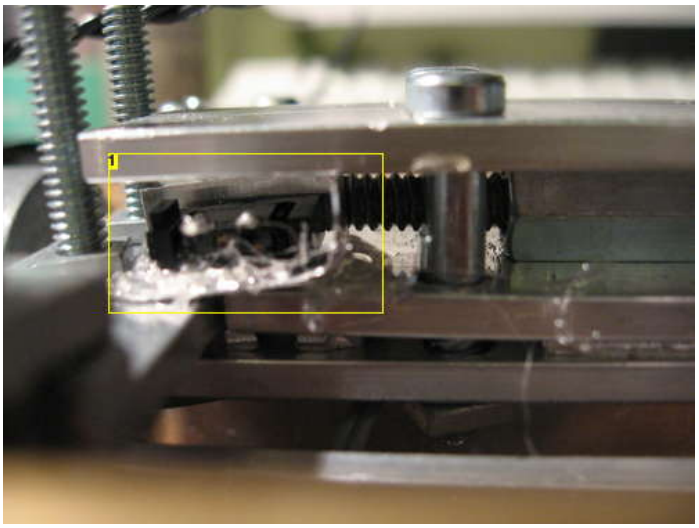
Step 18: Limit Switches

Because I used a regular DC motor instead of a servo or a stepper to spin the threaded rod that is the spine, the robot can not know the degree of extension of the spine at all times. Therefore, limit switches must be used to prevent it from extending or contracting too much.

The spine has two limit switches. One is pressed in when the two segments of the robot are pulled close together, and the other becomes un-pressed when the threaded rod retracts past it. The latter is a switch like this glued parrallel to the threaded rod, on the segment of the robot with the electronics. When the spine retracts, it pushes down the lever of the switch, and when it retracts, the switch opens.

The second limit switch is a push button switch that requires very little force to actuate. I mounted it on a strip of aluminum from the front of the electronics segment.

Both the switches are connected to the same 5V and ground lines as the potentiometers on the legs, and their signals go to inputs A4 and A5, which the Arduino is set to read as digital inputs rather than analog.



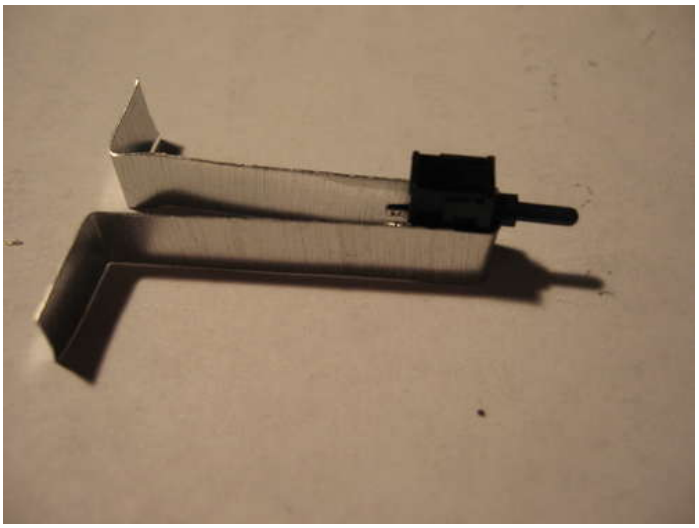
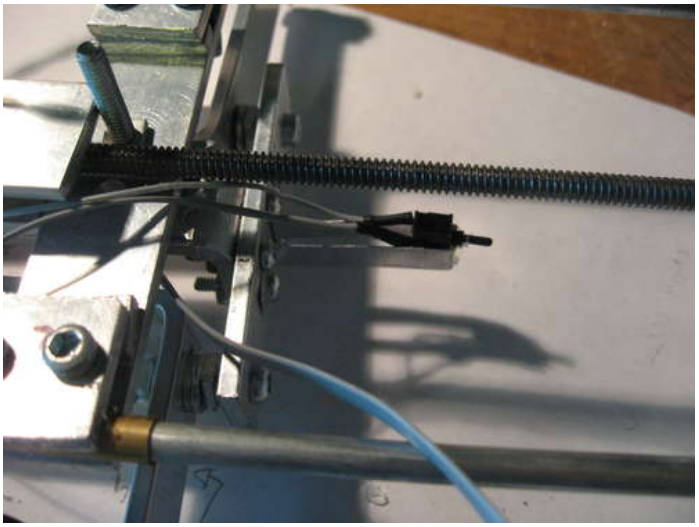
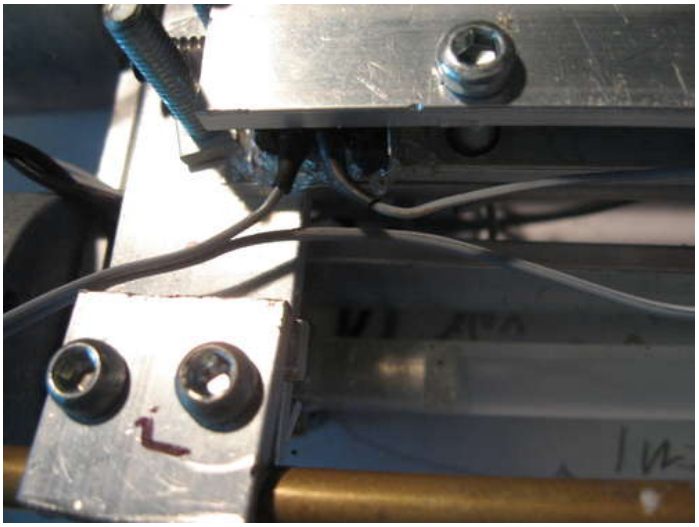


Image Notes
1. Switch



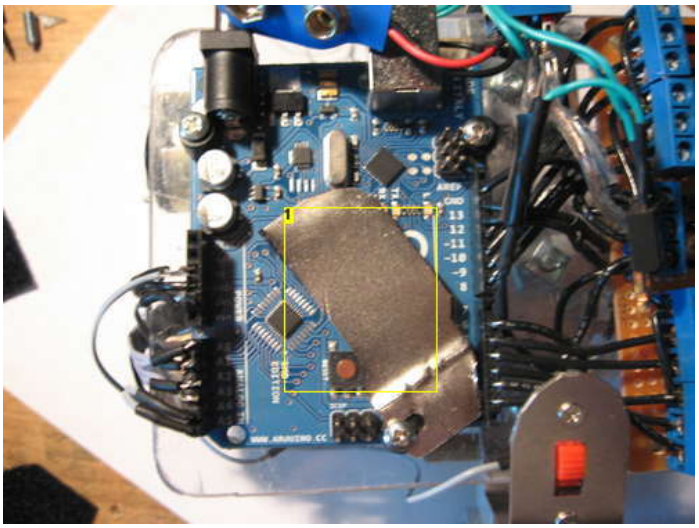


Step 19: Battery Holders

The last mechanical part of this project was to create a way to hold the batteries, while making sure that they are easy to remove for replacement or charging.

The perfect place for mounting the 9V battery was right above the Arduino, so I created a mounting system for it out of some scrap metal. A piece of metal (with an electrical tape insulated bottom) screws on above the Arduino through one of the standoffs. On top of the metal is a bit of stick-on velcro. A piece of metal bent into a "U" shape clips onto the 9V battery, and then sticks to the velcro above the Arduino board, holding the battery in place.

To hold the larger battery pack, I cut two brackets out of some soft plastic angle bar I had lying around. These brackets screw into the arms that hold the linear slides. The battery stays in mostly by friction, but a bit of velcro on one side helps to stop it from slipping out.



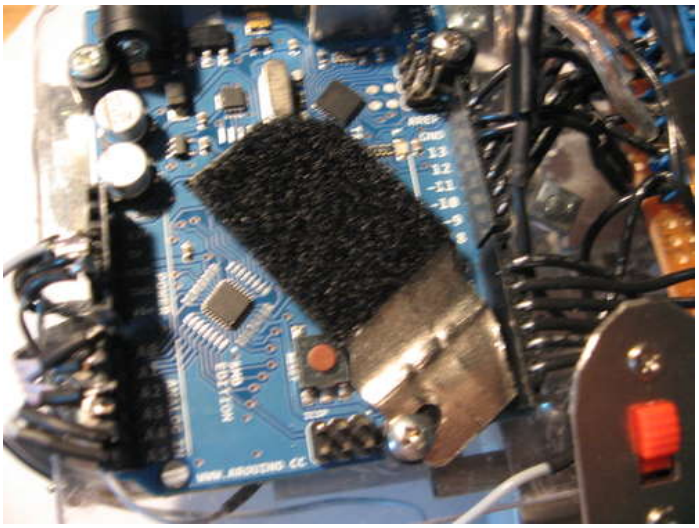
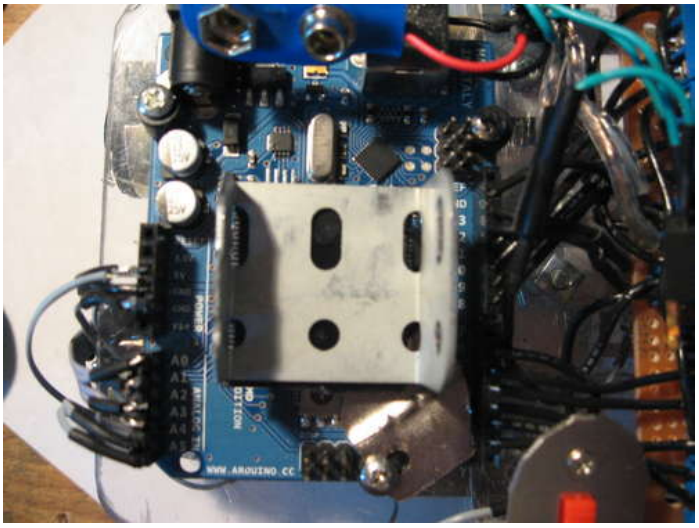
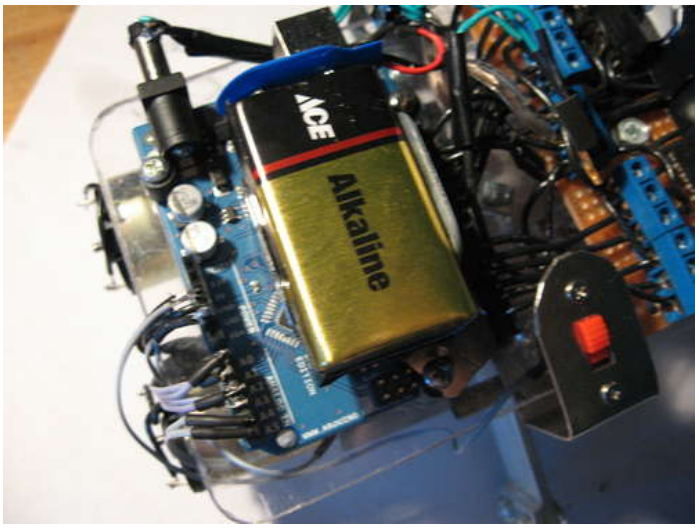
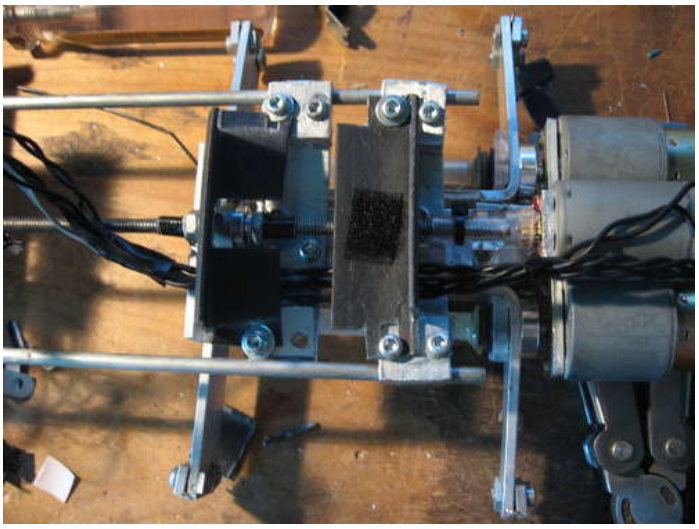
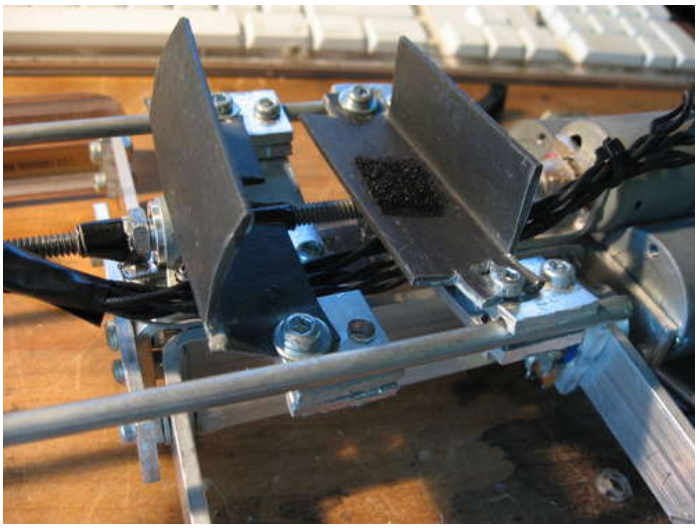


Image Notes
1. The bottom is covered with electrical tape to prevent shorts









Step 20: Programming

To climb up a tree, the robot goes through a simple series of motions. First, the top segment grips the tree and the bottom segment releases form the tree (if necessary). Then the spine contracts, pulling the bottom segment up towards the top segment. Next the bottom segment grips the tree, and afterwards the top segment releases from the tree. Finally, the spine extends, pushing the top segment upwards, and the cycle can start over again. For ease of programming, I wrote a function corresponding to each basic motion. These are as follows:

- closeTop
- closeBottom
- openTop
- openBottom
- Lift
- Push

By combining these functions in the proper order, the robot can be made to ascend or descend trees.

Opening the legs is very simple. The legs turn outwards from the tree until their rotation sensors reach a point set in the program. Then power is cut off to the motors. Closing the legs on the tree, however, is a little bit more complex. Since trees vary in diameter, the legs need to be able to grip a wide variety of diameters without reprogramming the robot for each size. To figure out when to cut off power to the motors, the controller first calculates the speed at which the legs are moving towards the tree. It does this by sampling the position of the legs' potentiometers every .05 seconds. It subtracts the previous value of the potentiometer from the current value to find the distance traveled by the legs over the time period. When the distance travels becomes close to zero (I used 1 in my program), it means that the legs have gripped into the tree and are beginning to slow down. Then the controller cuts of power to the motors, to prevent them from stalling out, or damaging themselves, the motor controller or the gearboxes.

The last piece to the programming puzzle is the method of controlling the robot's actions. If you look at the above movement cycle, you will notice that the robot is gripping the tree at all times. This makes it difficult to remove the robot, so I programmed the control switch to manually control the behavior of the robot. While the switch is off (circuit open), the robot keeps its legs open. Once the switch is turned on, the robot begins its climbing cycle. To remove the robot from the tree, the switch is turned back to the off position, and both sets of legs release.

If you liked this project, please vote for me in the Epilog contest! What would I do with a laser cutter? Well, I could use it to make parts for even more robots and machines (after I finished etching every electronic device I own, of course). Not having to manually cut, file, bend, and grind every component of my robots would let me significantly increase the complexity and variety of what I can build, and would also significantly cut down on construction time, so that I would be able to build even more interesting things.

Update

This project was featured on Hack A Day! Thanks for the great article.

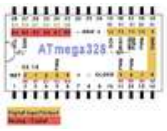
File Downloads

tree_robot.pde (5 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'tree_robot.pde']

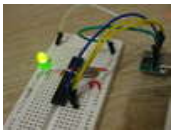
Related Instructables

Computer Controlled Musical Christmas Lights by dnicky2288

Line Follower Robot by nbibest



Ubuntu and the arduino. by Computothought



Arduino on a Breadboard (Photos) by ThatCaliforniaGuy

robot with microcontroller (Photos) by www.robotscience.

Arduino Controlled Line Following Robot (video) by earthshine

Belvedere - A Butler Robot by wolffan

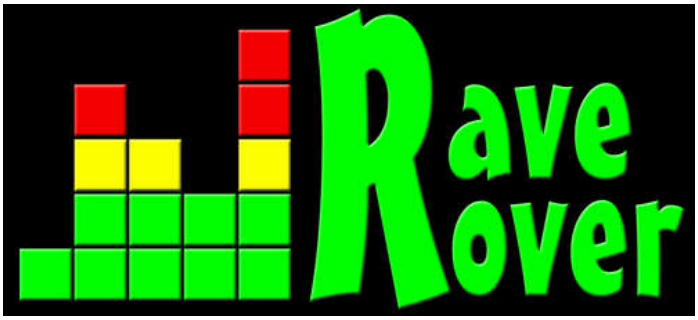
simpleWalker: 4-legged 2-servo walking robot by edwindertien

Rave Rover - Mobile Dance Stage

by **cwilliamson8** on September 10, 2011

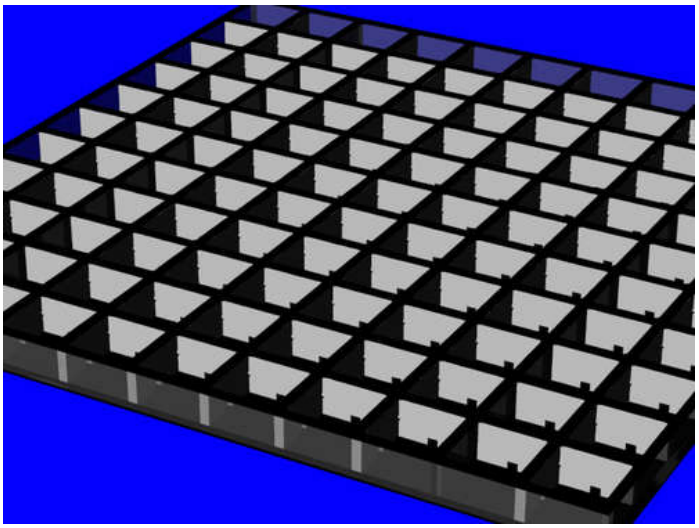
Intro: Rave Rover - Mobile Dance Stage

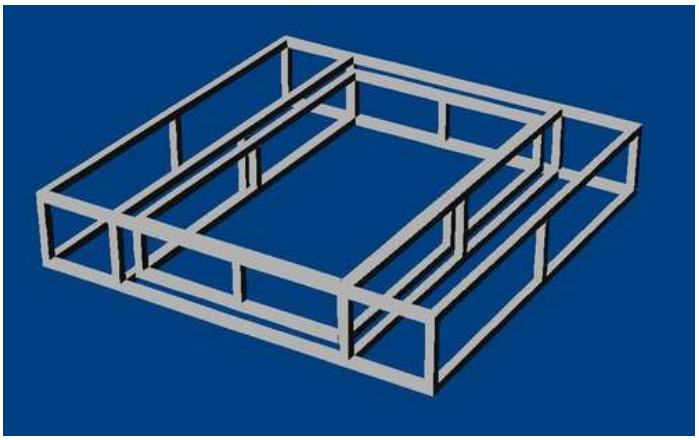
Rave Rover was designed and built to be a portable dance platform for parties, raves, and any other trouble we can get into! I will go into as much detail as I can explaining the entire build process, and where to find parts and other accessories. Be sure to check out more information, including party galleries on our website at www.raverover.com

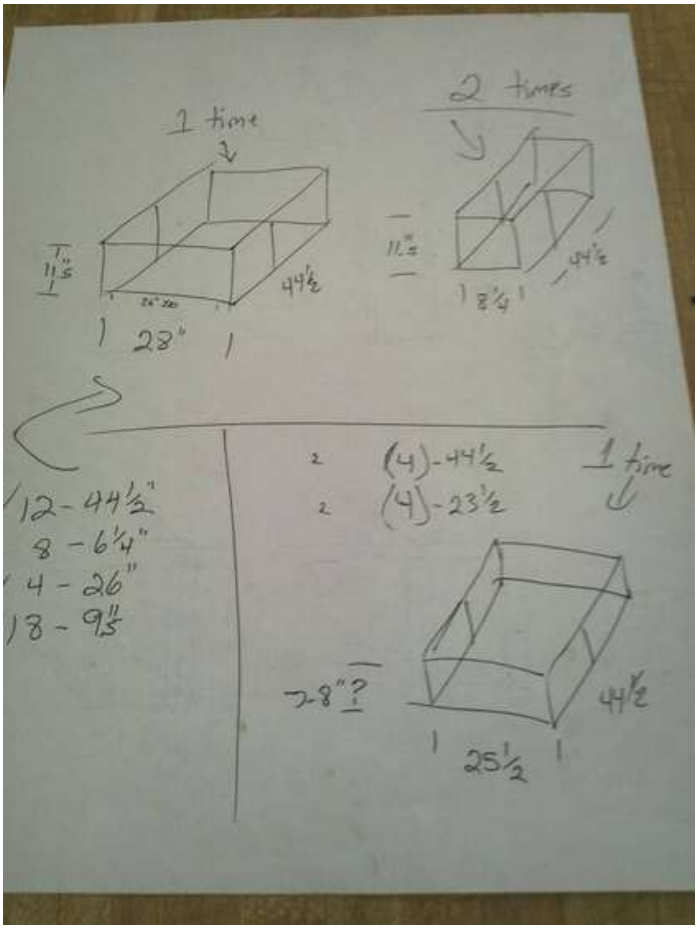


Step 1: Starting the Build

Before doing any work on putting something together, I always like to sit down and think about the development and how something should go together. CAD is a great resource for this, so I designed most of the layout before spending any money.







Step 2: Cutting Parts

After designing and seeing how things were going to start to fit together, I decided it'd be a good idea to start cutting parts. Luckily at work, I have access to a 5 foot by 10 foot CNC router, where I'm able to cut any types of plastics up to 2" thick.

From the CAD models, I was able to cut out the exact frame so that everything slides together and locks. I was also able to cut the top out of very thin ABS plastic sheet, which will give the 'rounded square' look once the LEDs are installed and lit up.

The reason for using black plastic is to try to keep this project as light as possible, while at the same time not allowing any light to go between boxes.

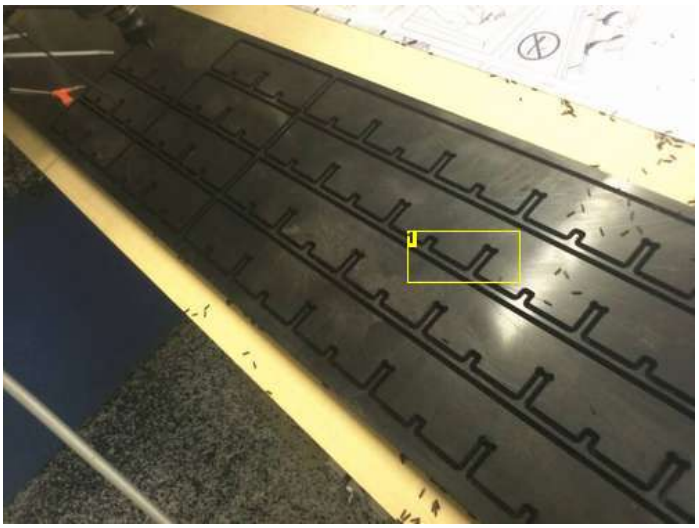


Image Notes
1. Interlock section and a groove for wires

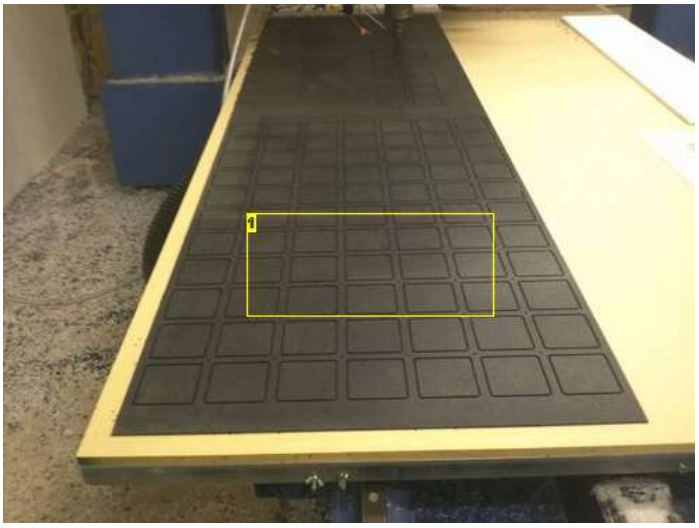


Image Notes
1. Top panel grid for the main portion of the stage.

Step 3: Fitting the floor

Once all of the floor pieces were cut out, assembly began of all the rails to check fitment and make sure enough pieces were cut. There are three main sections, the center of the stage, plus the two pieces that fold up.

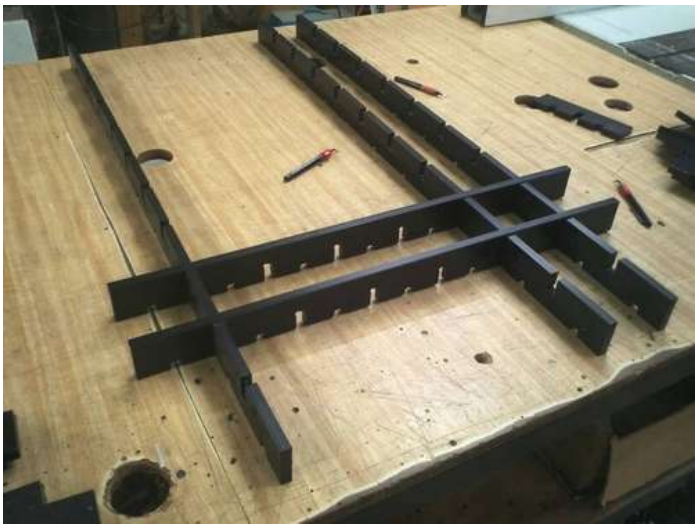


Image Notes
1. Making sure everything lines up correctly

Step 4: Getting LEDs ready

After cutting all of the parts, it's time to assemble LED modules for the floor. These specific LED modules have three SMD5050 RGB leds per module, and they are able to be controlled over an SPI interface. This makes for being able to change any module to any color at any time, and allows the most control for some really cool displays!

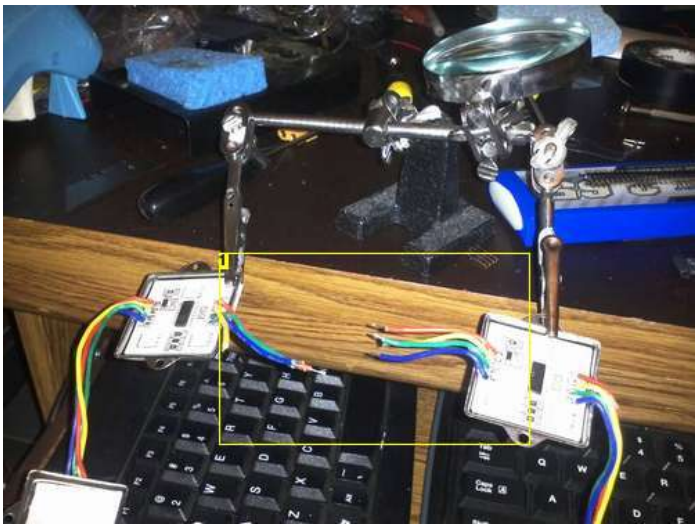


Image Notes
1. Four Wire interface for connecting, Power, Ground, Clock, Data

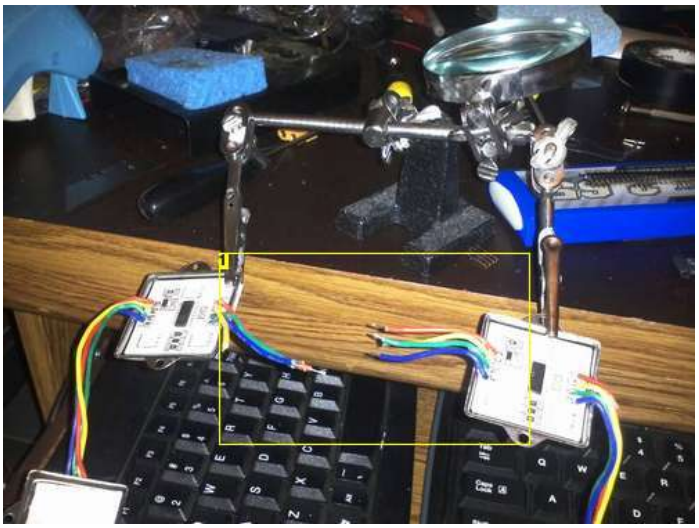


Image Notes
1. Light up test to make sure each strand works before installing!

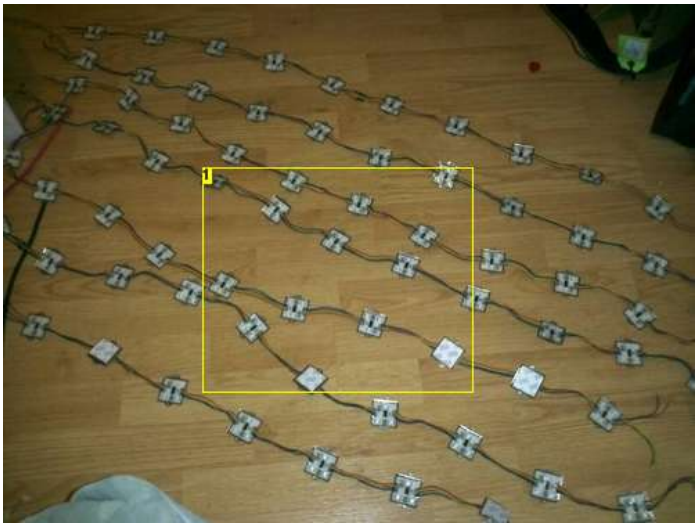


Image Notes
1. Almost all the strands complete

Step 5: Installing the LEDs

Once all of the LED strips were set up (the matrix was 11 x 11, so we built 11 strands of 11 LED modules), it was ready to start installing the modules. Luckily, these modules have 3M double sided tape on the bottom, so positioning them were very easy, but we did come back with some hot glue to make sure they stuck to the bottom panel.

The have to be wired all in series, so each row has to follow the row before it, and the 'flow' has to be correct, else you'll not be able to light up some of the LEDs. The way these LEDs work is by sending them a serial string of data, the first LED takes it's data off the top, and then sends the rest of the packet down the line, basically bit shifting the data stream. You can't individually address the LEDs, but knowing where they are in the data stream, you can change their data in the stream itself.

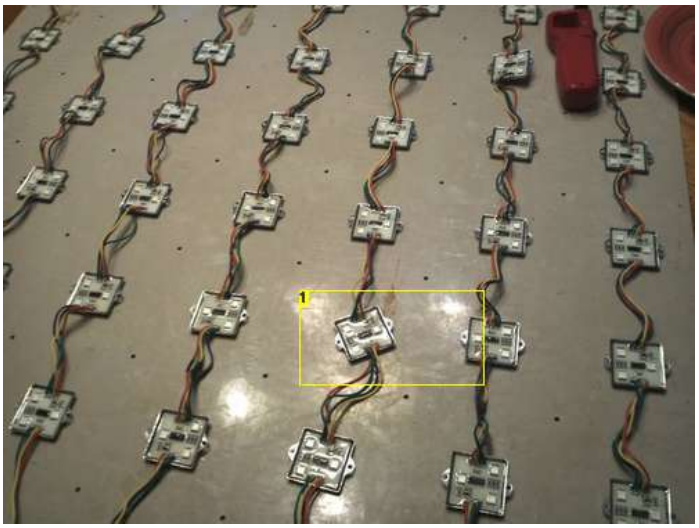


Image Notes
1. Sometimes 3M tape doesnt always stick, always best to go back and make sure they stick with Hot Glue or another adhesive!

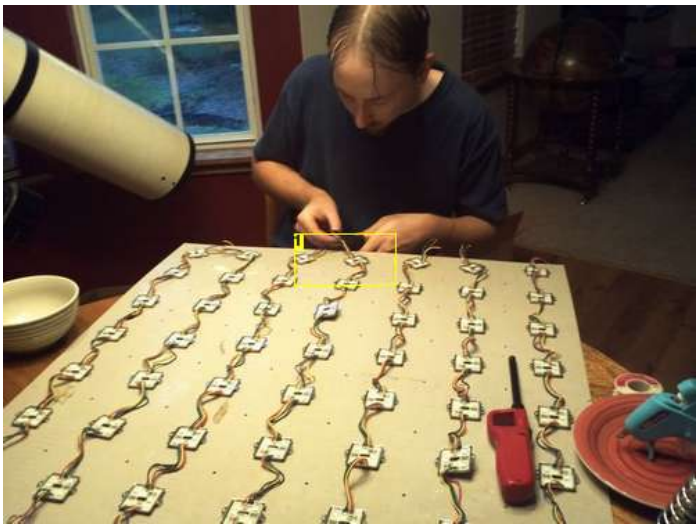


Image Notes
1. All of the LED ends have to be wired together, so once the strands are installed, they still have to be coupled together.

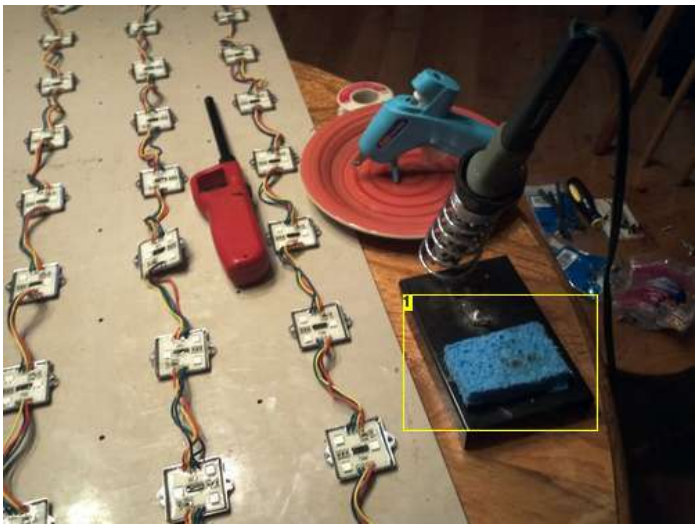
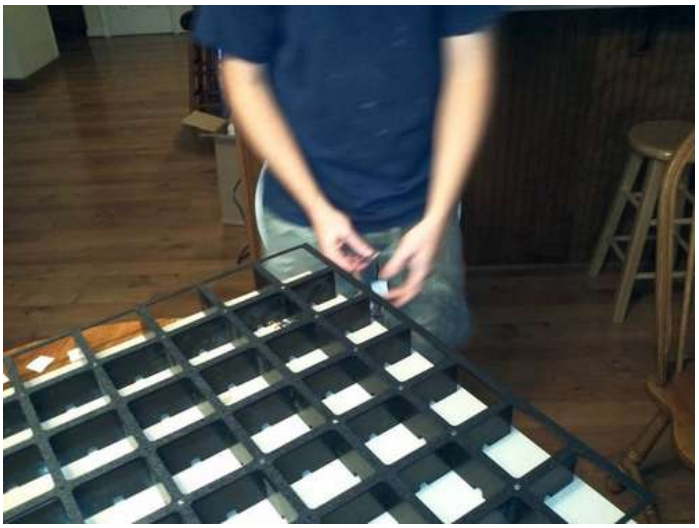


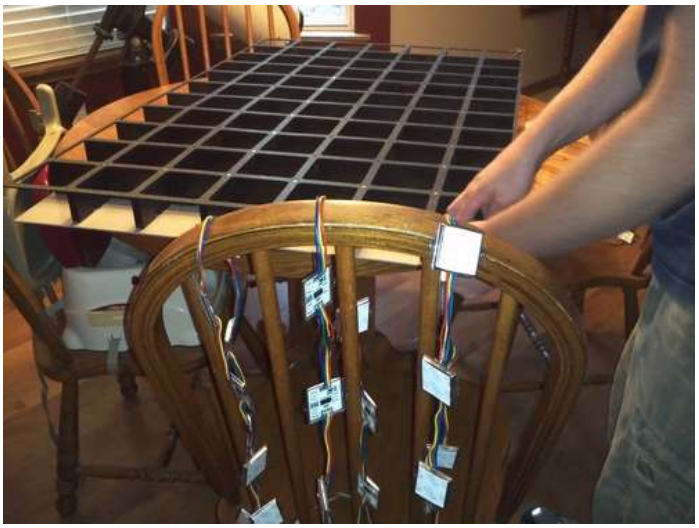
Image Notes
1. The tools of the trade, Soldering Iron, HeatShrink, a lighter for the heatshrink, and a glue gun!

Step 6: Adding the Frame

Once the LED modules are all planted on the bottom panel, it's time to over lay the cut frames to set up the light 'boxes' or 'pixels'. The holes in the bottom panel (and top panel) allowed us to be able to screw the top and bottom overlays to the rails themselves, and make the entire structure much more solid and to keep it from sliding apart.

You can see how the slots in the rails we cut out are perfect for running the wires between the boxes.





Step 7: LED Color Check and Testing

Once everything was wired and completely set up, it was time to power it up and check for colors and tracking.

For driving the LEDs, we're using a simple Arduino by outputting data out of the SPI channel. Most of what you see is just random algorithms to make sure the colors are in working order and all of the pixels are working.

The top piece is white translucent plastic, works as a great diffuse panel as one is needed!

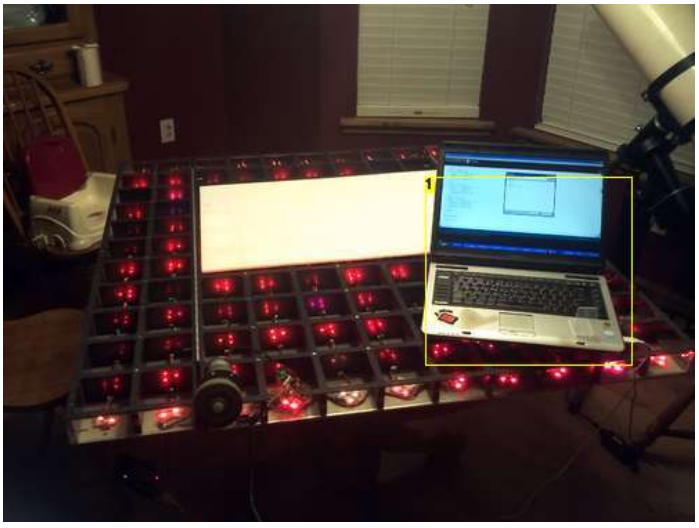
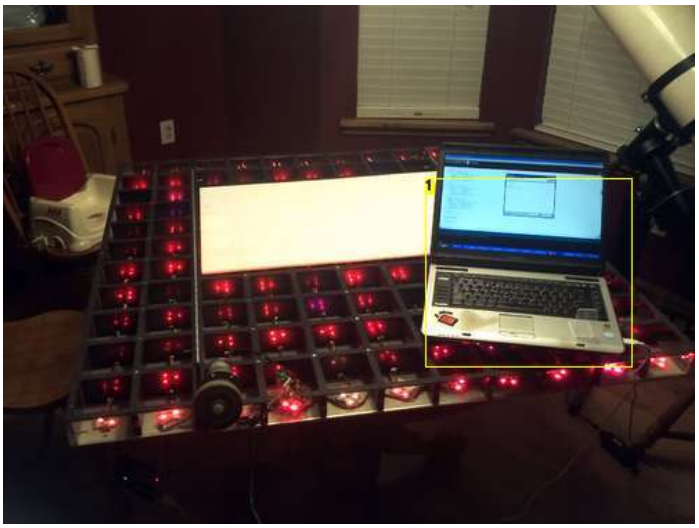
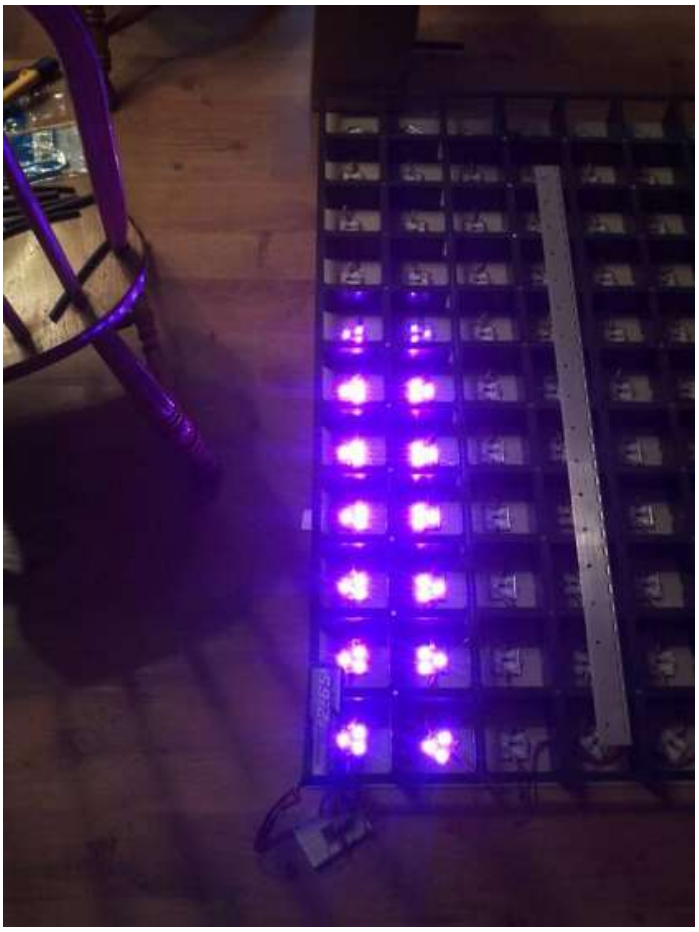


Image Notes



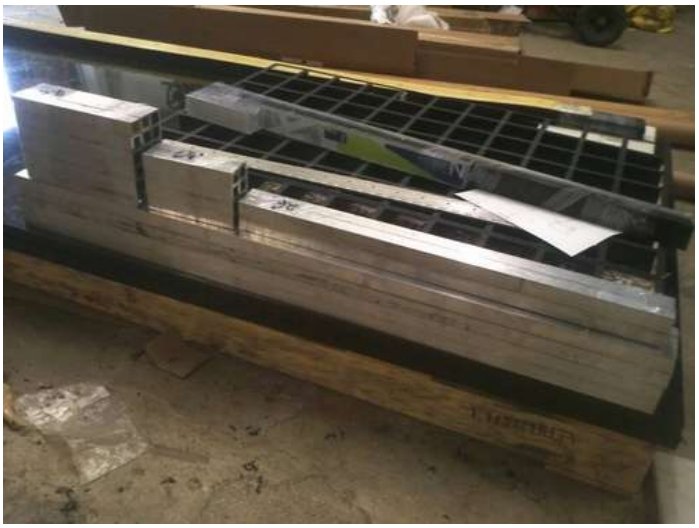
1. Arduino Programming



Step 8: Gathering More Materials

Once happy with the LED floor itself, it was time to start gathering materials to build the drive train and frames for mounting all of the rest of the electronics.

We picked up all of the aluminum (1x1x1/8" wall) tubing, and cut it up to the sizes that we needed for the frames. While doing this we also picked up all of the pneumatic components which I'll get into later in the build and explain WHY we need air cylinders on this project :)



Step 9: Frame Building

The goal behind the project this year was to be able to make the stage sit completely on the ground when 'in use'. The method to do that was to build essentially four frames. The main stage area frame was 28 x 44.5 inches, with a foldable wing on either side that were each 8.25" x 44.5". This allowed us to be able to drive through a standard 30" door opening once folded up. The drive system frame was built to fit inside the main stage frame, and by using drawer slides as linear rails, the entire stage could be lifted up or down and be allowed to be raised to drive around, and then lowered for stability for the dancers.

The next few steps will show the construction of these aluminum frames.

The original idea was to have all of the aluminum to be welded, but running out of time it was decided to use L brackets to bolt everything together. This seemed to be extremely strong and held together very well!

Image Notes
1. Main stage frame going together

Image Notes
1. Main Stage Frame Complete

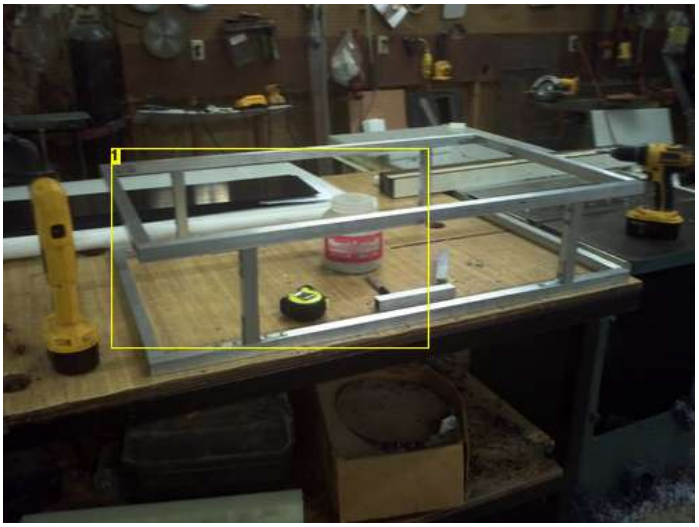


Image Notes
1. Drive train frame complete

Image Notes
1. Extra rails added on top of the Stage frame to support the LED frame and dancers.

Step 10: Getting frames to fit...

Once all of the frames were built, it was time to get the two main frames sliding together, so the linear rails (drawer slides) and the pneumatics started going together.

In the video you can see how hard it is to control the air, I am using a standard blow nozzle and just shooting air into the input to make sure the frames will move and not be locked together. In the final version, I fixed the flow by adding in a flow restrictor on the solenoid input, make going up and down very smooth.

Image Notes
1. Linear Drawer Rails/Slides that were 8 inches long

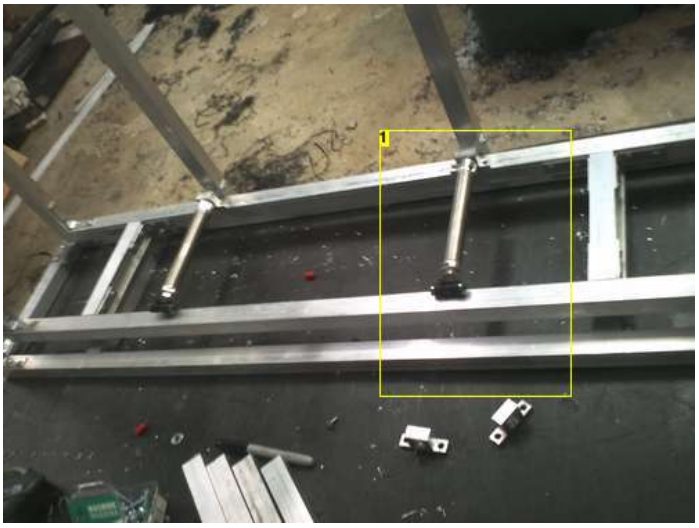


Image Notes
1. Air cylinders mounted and ready to be aired up

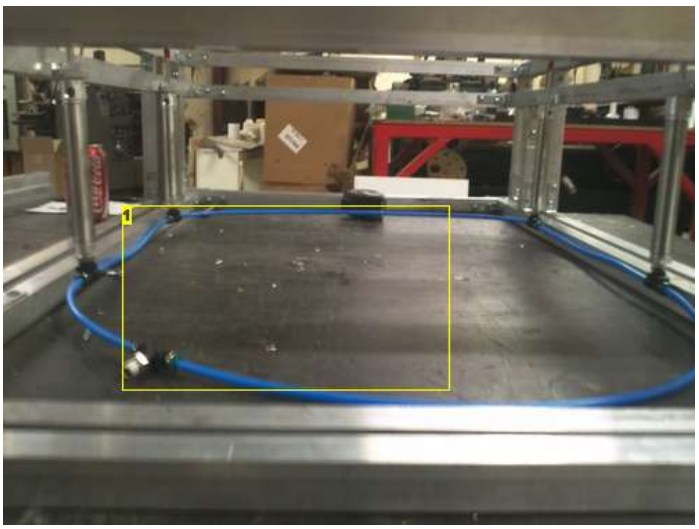


Image Notes
1. Air line installed for testing in the video.

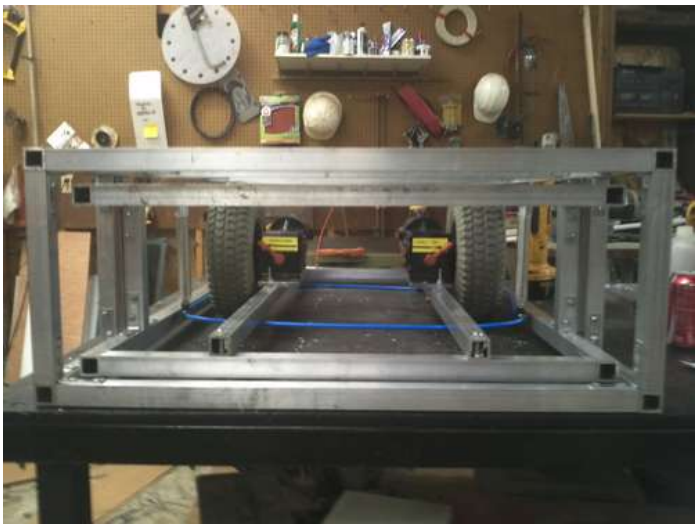
Step 11: Mounting Components

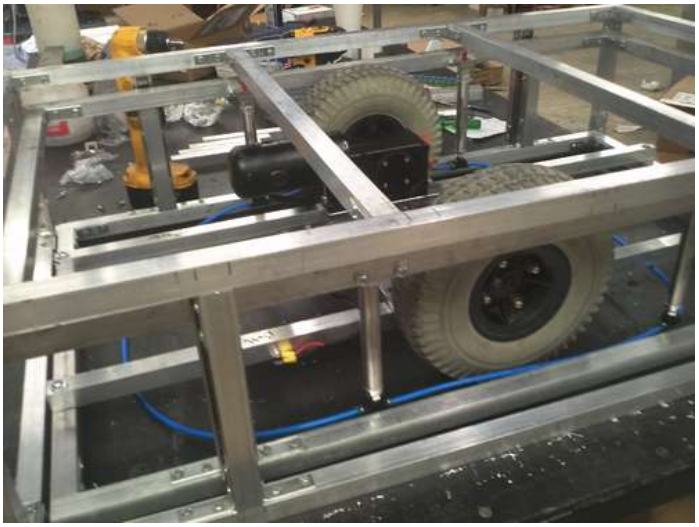
Now that we have the frames built and the air cylinders working together, it's time to really get down to business and start trying to figure out how to shove:

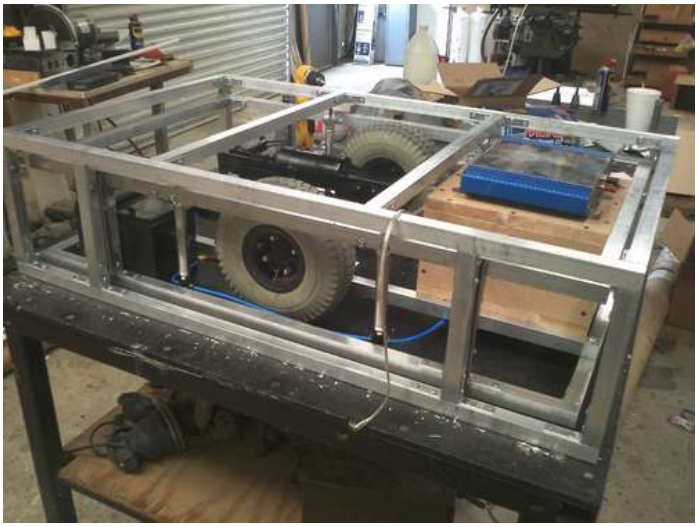
(2) Drive motors with 10" Wheels (From Electric Wheel chair)
Custom 10" Subwoofer Box
Amplifier for Subwoofer
Car Radio for powering mids/highs and taking computer input
Onboard PC
(2) 12v 35Ah SLA Batteries
Compressor
Air buffer tank
Electronics (Solenoid, drive speed controllers, Arduino, power switch, etc)

Now if you remember, the Main Stage frame was 28 x 44.5 inches, this means that the drive train frame was smaller, around 25x42" where all of this stuff has to fit. What are we waiting for?! Lets get to it!

We start installing by necessity. Obviously we need to drive around, so the motors and wheels get mounted first! Next is the batteries (can't forget those)..and then the next biggest item which was the subwoofer.







Step 12: More Mounting...

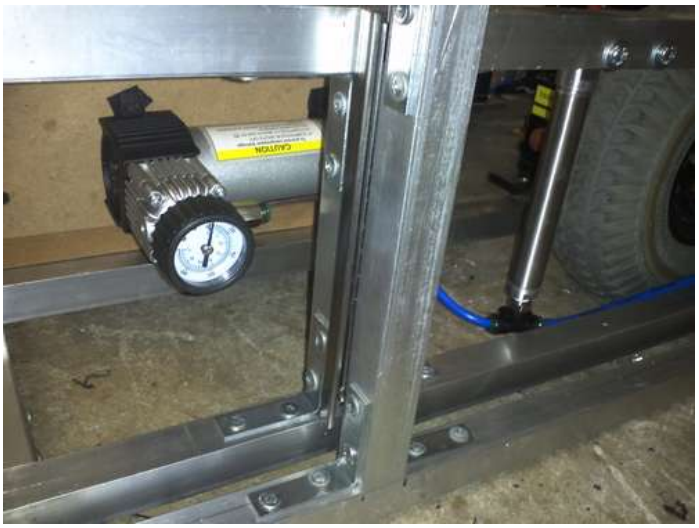
Once we had the big parts out of the way, it was time to start finding room for all of the smaller items. Also we can't mount anything in the center, because we have to leave room to be able to mount the pole!

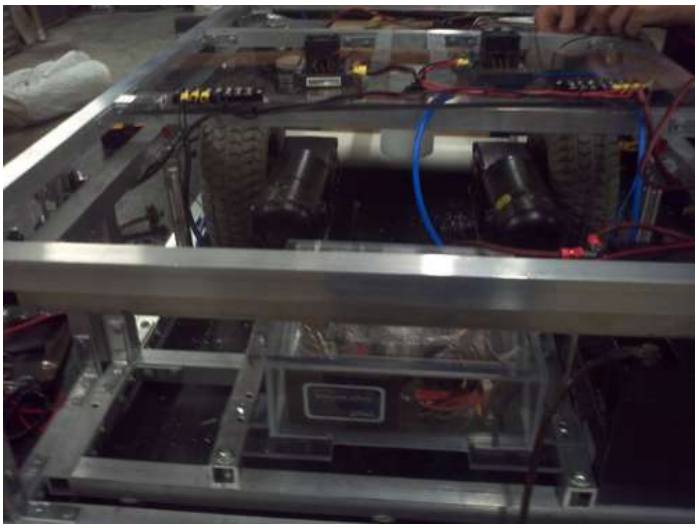
With the onboard computer, we're running Windows XP and custom software along side of RoboRealm, which is a high customizable robotic software. I came across a 10" vga LCD and decided it wouldn't be a bad idea to slap it onboard too just incase debugging was needed in the field.

The computer itself is a Zotac Mini ITX in a custom case with 2Gb Ram, and 32Gb Solid State hard drive.

Compressor was picked up from Harbor Freight, just one of the small simple compressors that you keep in your car incase of a flat, plugs into the lighter.

We decided to use Victor 884 motor speed controls, these deliver plenty of amperage for the wheel chair motors at 12v.





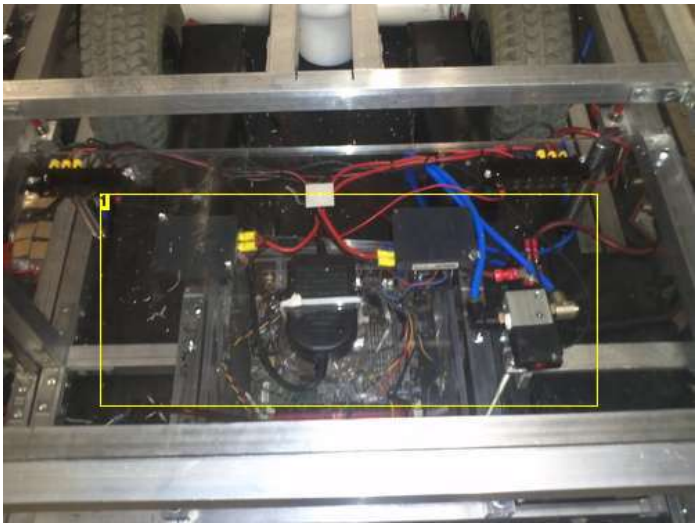


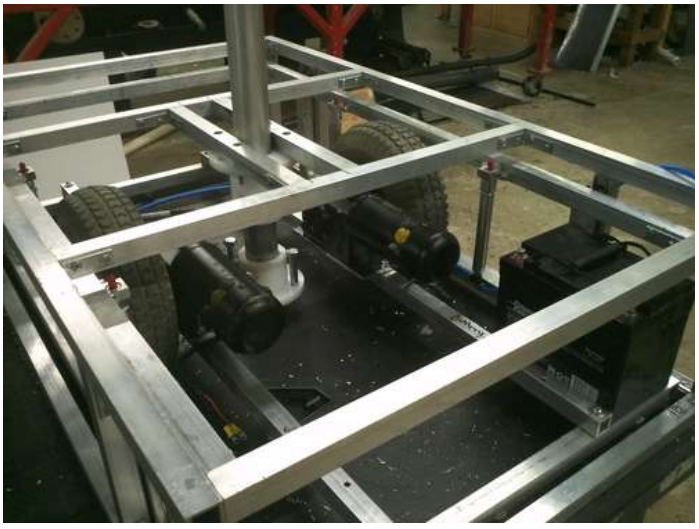
Image Notes
1. Zotac Mini ITX with 2Gb Ram and 32Gb Solid State Harddrive

Image Notes
1. Victor 884 Speed Controls 12v to 19v Converter for PC Standard 3 Position Air Solenoid Ground and Power Strips

Step 13: Pole Mounting

We can't forget about the pole! The pole is a two piece design, where around 20" is always mounted inside the Rover, and the other 70+ inches slides into a custom fabricated mount system.

The pole is mounted to the Stage Frame at the top and bottom for added stability.



Step 14: Finishing the Electronics...

After installing all of the drive parts, it was time to install the arduino, start securing some wiring, and run air lines to get ready for a drive test!

We also mounted the WiFi router which allows us to access the onboard PC by just a simple HTTP server. This allows any cell phone with a browser to go to the Rave Rover webpage being served, and change music, the LED modes, and in the future more stuff!

For Audio control, we feed the Audio line out into the AUX line in on the Car Radio. Using the Car Radio saved a lot of time as it has a built in 4 channel amplifier which was perfect for mounting the four external speakers.

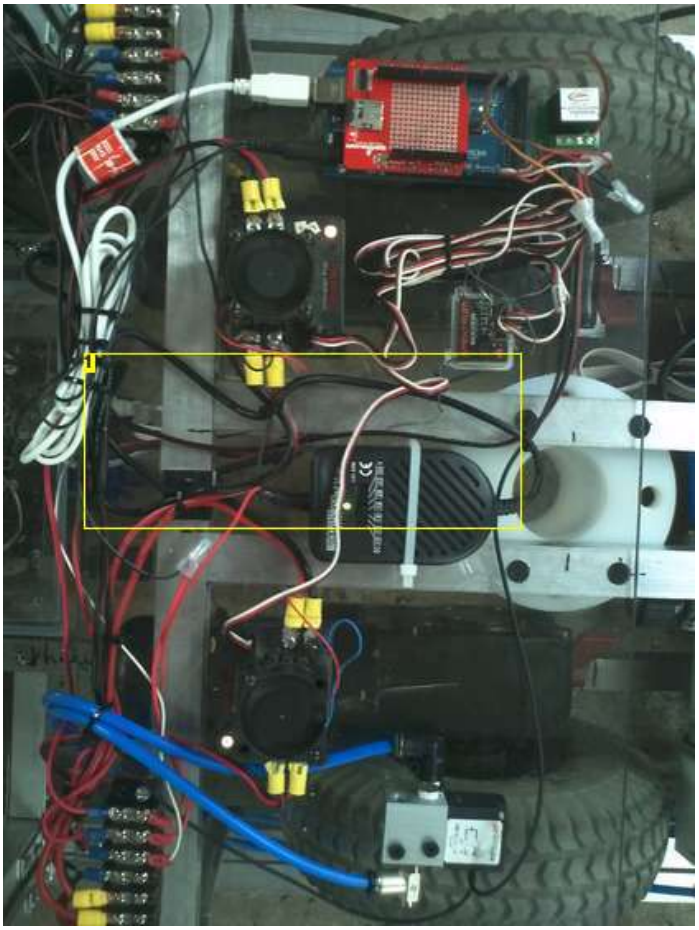


Image Notes
1. Electronics mounting complete!

Image Notes
1. Zoom Player is great for its TCP interface to be able to control it over a network.

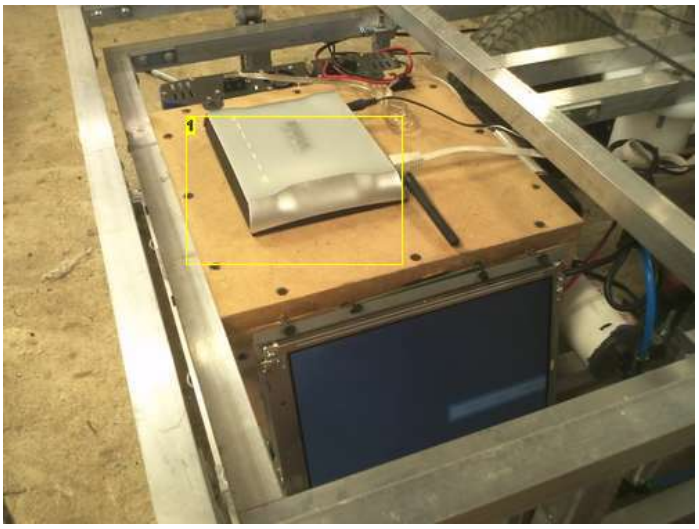


Image Notes
1. Wireless Router Mounted

Step 15: Drive Test!

Once things were secured, a power check was in order, and then a drive test. The control system is a standard Spektrum DX6 Radio Transmitter and Receiver, you can find the newer models from places like http://www.robotmarketplace.com or http://www.towerhobbies.com

Step 16: Installing Floor

Now that the frames were built, everything mounted, and we were able to drive, it was time to mate the pieces together that we built earlier. Time to install the LED floor and get some music going! I'm getting anxious to party!

The floor fit perfect, and just using some simple self tapping screws, it was a super fast job of attaching it to the aluminum frames.

Hinges were mounted on either wing to allow that section to fold up to retain the size of being able to go through a standard 30" door.

Image Notes
1. Aluminum Piano Hinges for Folding

Image Notes
1. Power Test, it works!

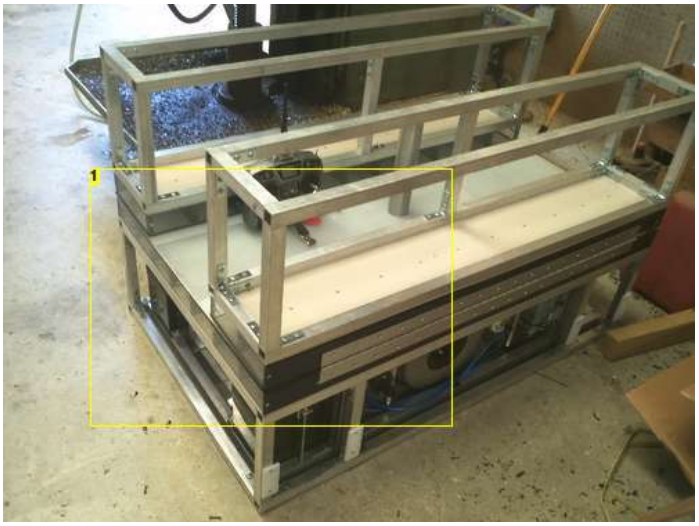


Image Notes
1. Once folded, it can easily be tucked away in a corner.

Step 17: Final touches

You guys ready to party? I know I am, but first....

Gotta make it pretty!

It was time to cut some side panels, and install the speakers. For the front and rear panels, we wanted to make sure everyone knew 'Rave Rover' when they saw it.

We took some acrylic mirror as a backer, installed some stand off rails, and a front clear piece of acrylic with a vinyl decal with the Rave Rover logo cut out. Trying to find a diffuser element was tough, but tissue paper worked perfectly for us. The inside of this box was lined with green LED strips so that when power was on, the logos lit up bright!

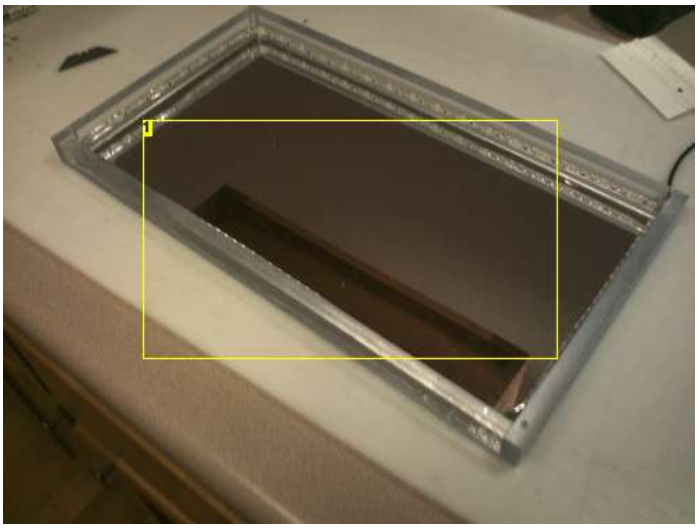


Image Notes
1. Mirror background, LED strips around the edges.

Image Notes
1. Vinyl decal with letters removed.

Image Notes
1. All layered together

Image Notes
1. Lit up with front panels overlaid.

Step 18: Speaker Install

Time for speakers! Simple install of just screwing them in, and wiring them to the car radio. For this build we used Fusion 5.25" 2 - Way Car door speakers, cheap enough not to be worried about being kicked.

Step 19: Finally Done!

Everything is finished, and we're ready to party....

First party is Dragon*Con 2011, the next step will show you some pictures that we got from the event! For now, check out the modes of Rave Rover below.

Yes, the audio IS coming from the rover itself....

Step 20: Where to find parts...

Just wanted to give you guys a source list for some of the parts that I managed to pick up:

www.ebay.com -> Sourced Wheel Chair Motors
www.surpluscenter.com -> Sourced Air Cylinders, Pressure Swtich
www.arduino.cc -> Best source for Arduino References
www.robotmarketplace.com -> Anything Robotic!
www.hobbyking.com -> Sensors, Radio Control Gear and More
www.towerhobbies.com -> Radio Control Gear
www.pololu.com -> Sensors
www.sparkfun.com -> Arduino!
www.tigerdirect.com -> Sourced Onboard Zotac Mini ITX Motherboard and Harddrive
www.interstatebatteries.com -> Sourced Two 12v 35Ah SLA Batteries
www.mscdirect.com -> Sourced Air line, bolts, nuts, and other small hardware
www.harborfreight.com -> Sourced 12v Air Compressor!

I'll add more as I remember them!

Step 21: Party Time!

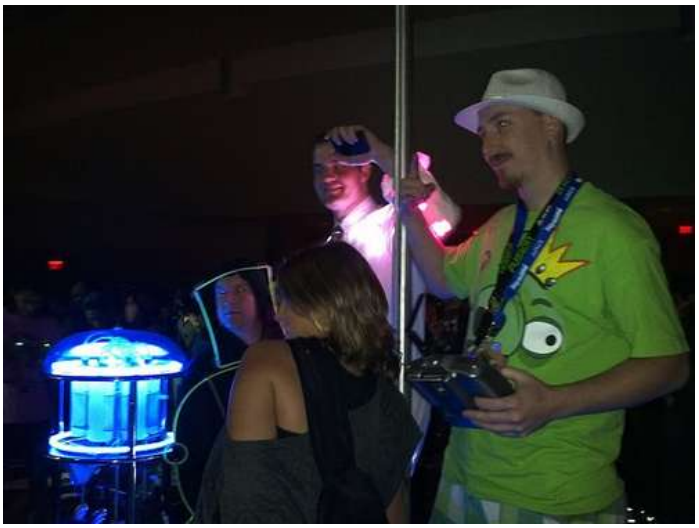
Here's just a few images from Dragon*Con 2011, for more, check out our website at http://www.raverover.com

You may notice some of our friends in the background...Yep, that's Bar2D2 and Marc DeVidts, be sure to check out their webpages too!

http://www.uiproductions.com

http://www.jamiepricecreative.com/bar2d2.html

We hope you enjoyed our instructable! This project was a lot of fun to do, and we can't wait for the next event.



Related Instructables

Easy Autochanging RGB Dome Lamp by powerman666

Cyber Mask Mod by neologik

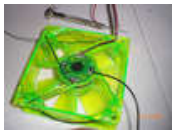
Installing 12V Color Changing RGB Lighting by Oznium

Carlitos' Projects: Wireless Speech-Controlled Arduino Robot by RobotShop

How to Make a Super-Bad Rave Mask (LED) by gardnsound

Sound Activated 4 X 7 RGB LED Matrix by jwflammer

The Lightning Simulator/Breatha Equalizer - Arduino Powered by alinke



Raving out your Computer by itstemo1

Type Case, the making of a low-resolution display

by **Martin Bircher** on July 16, 2011

Author:Martin Bircher author's website

I am an artist, son, electrician, student, lecturer and uncle.

Intro: Type Case, the making of a low-resolution display

What this text is all about

What I describe here is how I got from an idea to a presentable art piece and I don't want to write up a construction manual: How to build your own "Type Case" installation.

Abstract

The installation "Type Case" is a low-resolution display with 125 rectangular pixels of different sizes. These are formed from the reflecting light of digitally controlled LEDs, embedded in each section of a European printers' type case. Due to the standardized fragmentation of its compartments, the density of visual information is decreased towards the objects' centre. Viewed close by, it is nearly impossible to recognize more than a flicker – however after moving some distance away, it becomes distinguishable, that the lights and shadows give a representation of the latest headlines.

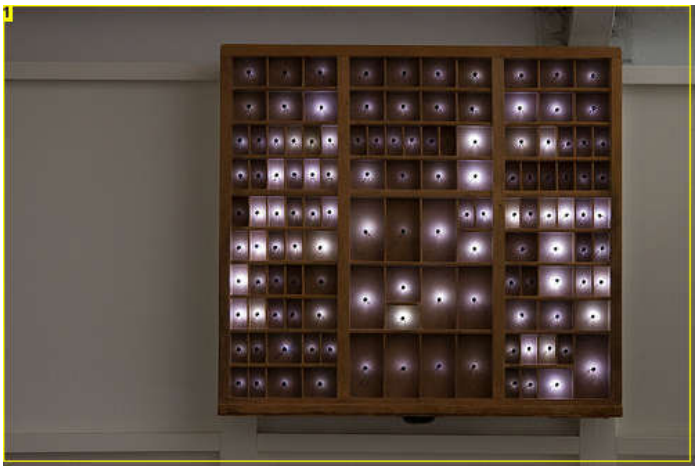


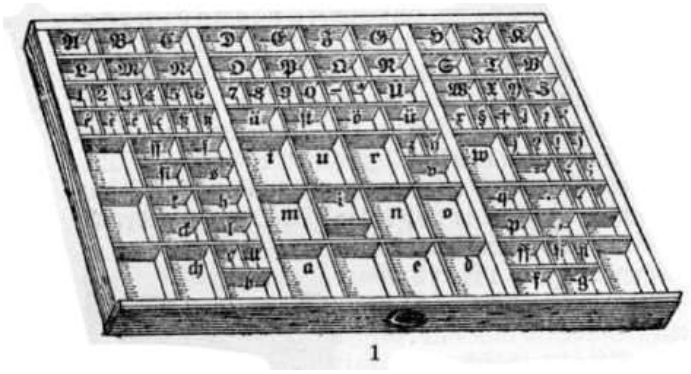
Image Notes

1. The ready "Type Case" installation, displaying recent newspaper headlines.

Step 1: The idea

On a late winter day last year, I looked at a picture of a European type case. I was fascinated of its odd segmentation and got interested whether it was possible to use the case as a display with different sized pixels.

Most of the ideas I get, I forget as fast as they came, but I immediately had a good feeling about this one.



Step 2: Simulations

I've learned it the hard way a few years back: It is better to make some mockups and visual tests before jumping off. Simulations have many benefits. First of all they can be changed more easily then the real deal and therefor are better suitable for experimentation. It can be seen faster whether something might work or not and after everything is right, a simulation is a proof of concept and can help to get funding for a project.

I did some compositions in Photoshop to see what the 125 pixels are be capable of. It became clear that it was not possible to recognize images, with so few pixels – text on the other hand worked somehow. Since it is not so interesting to display static letters I needed to get going with an animation. I love to use Processing for all kinds of things and therefor it was the tool of choice.
My first test in Processing was a conversion of a movie to the 125 pixels. Here is a flick which shows a Betty Boop cartoon, the Processing simulation and a later recording of the real thing.

(The original Betty Boop cartoon from 1933 is in public domain and not copyright protected anymore. Source: archive.org)
I think that this shows quite well the limitations due to the very few pixels.

After that I moved on to animate text. The simulation fetched resent headlines from an online newspaper and displayed them as scrolling text. The bigger part of the Processing code was then later used for the real object, which after this tests, I was confident to build.

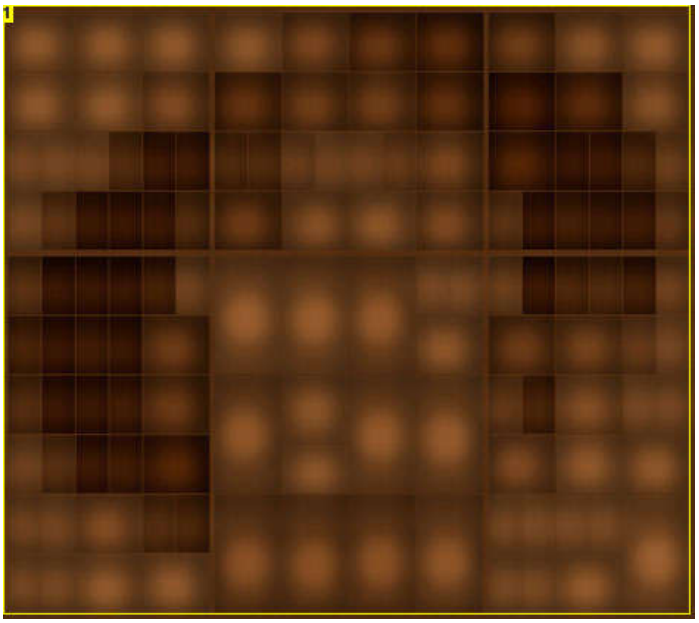


Image Notes
1. Photoshop simulation: Movie star with hat.

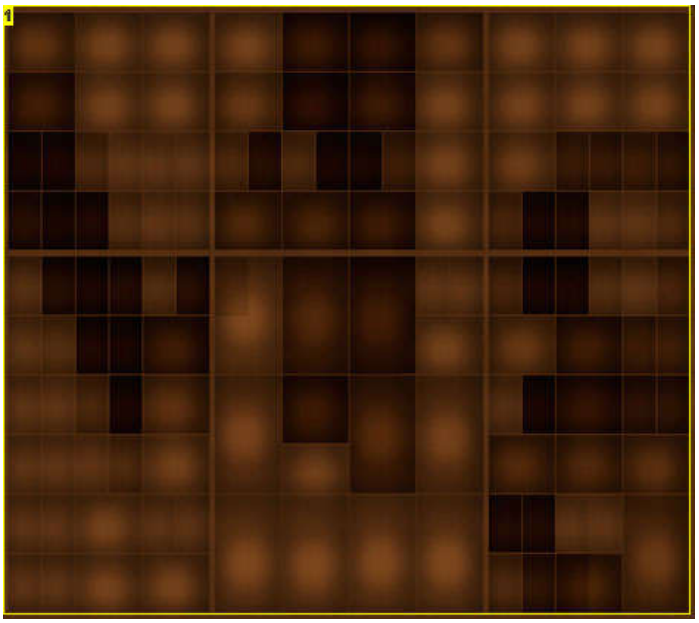


Image Notes
1. Photoshop simulation: Some simple text.

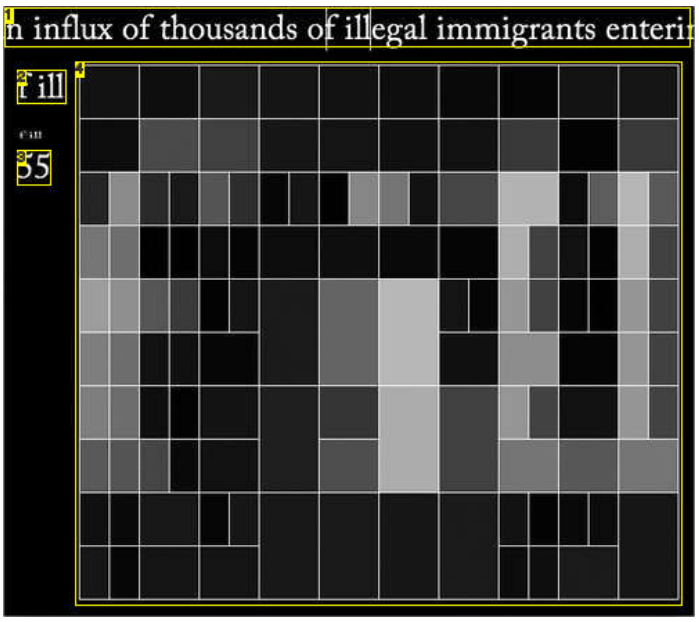


Image Notes
1. The headline.
2. The actual displayed part of the headline.
3. The resent frame rate.
4. The brightness values of the according compartments.

Step 3: Development = solving problems

After having the visual proof of concept, I had to solve a huge amount of problems. To be still standing at the beginning and being confronted with so many questions can be discouraging and stressful. I usually try to break down the big problem into smaller ones, which aren't so complex anymore. After this process I ended up with the following partial questions:
How will I be able to use the compartments of the case as a pixel?
What light sources will I use?
How can I control this light sources and their brightness?
How will I distribute the task between computer and microcontroller?
How will I mount the hardware on the case?
How will I mount the case?

Fortunately I did not have to find answers to everything yet and started with the electronic: I decided to use Arduino and TLC5941 LED drivers. After some tinkering and reading thru many different forums I had a working solution.

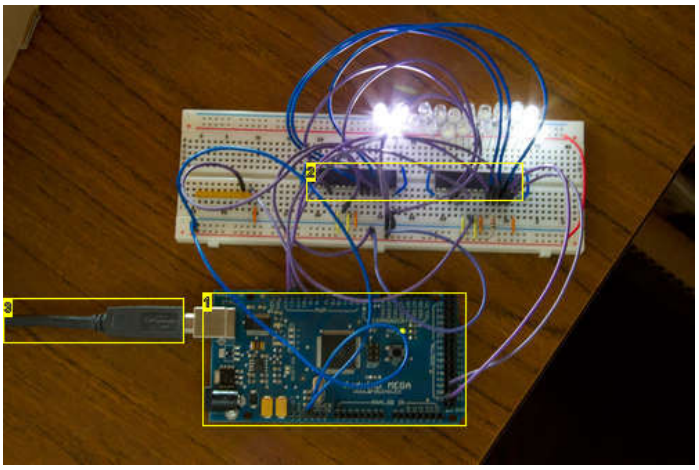


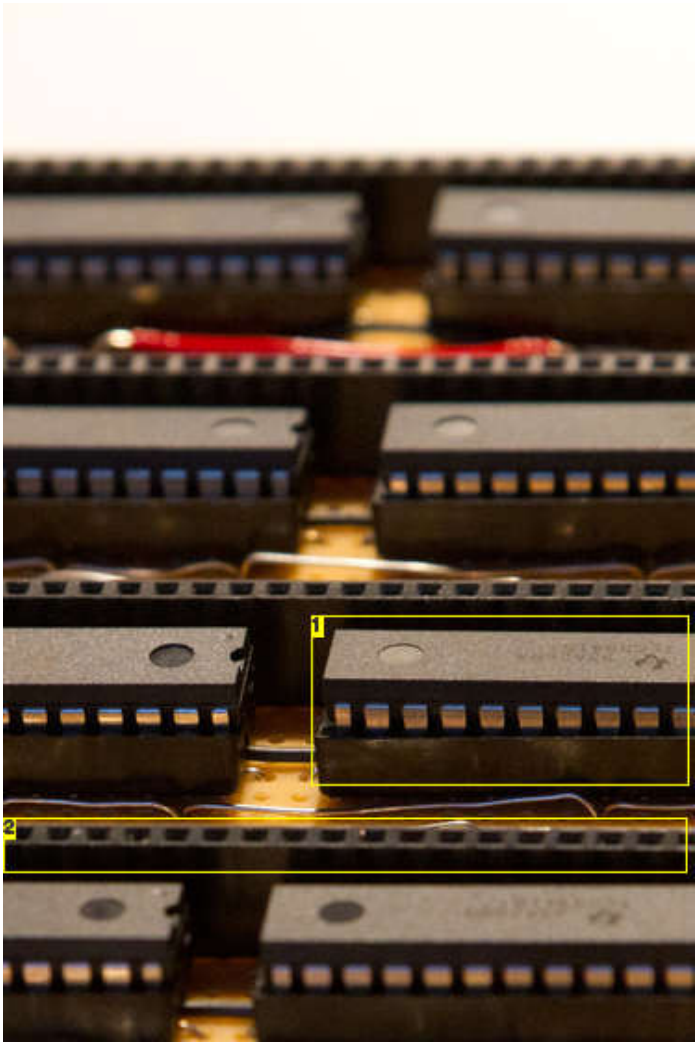
Image Notes
1. Arduino MEGA
2. 2 TLC5941
3. Serial data from Processing

Step 4: The build

Building the object was then just a question of time and endurance. Looking back I calculated, preparing, mounting and soldering one LED took about 20 minutes. Adding this up for 125 pixels we are talking a Swiss working week of 42 hours.

Here a small movie about the assembly process:

Now it was finished!



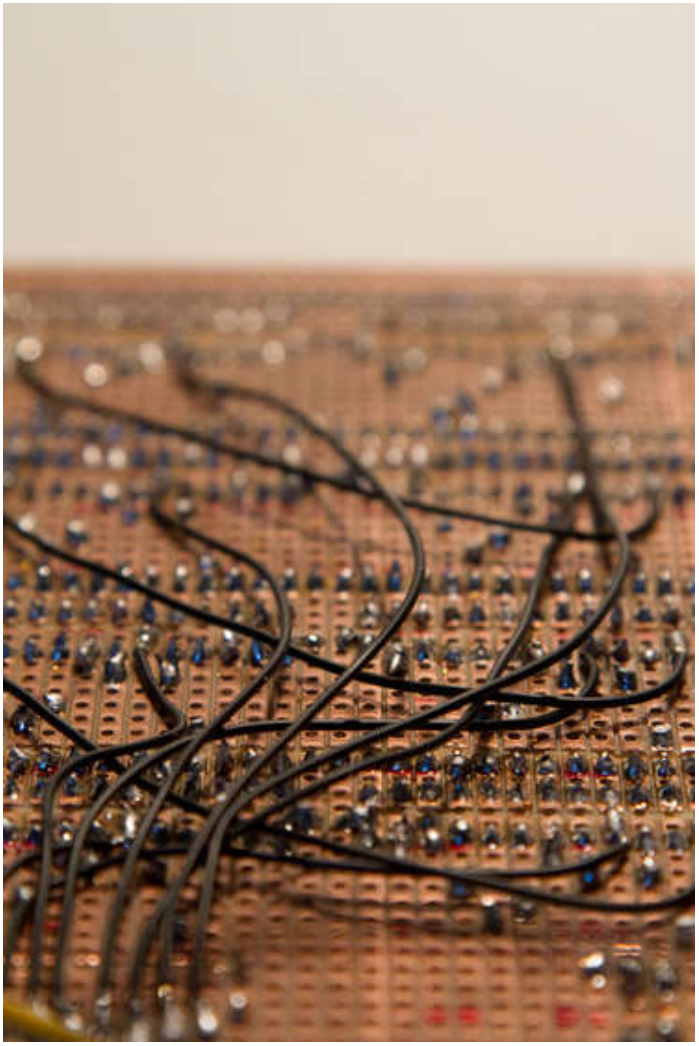
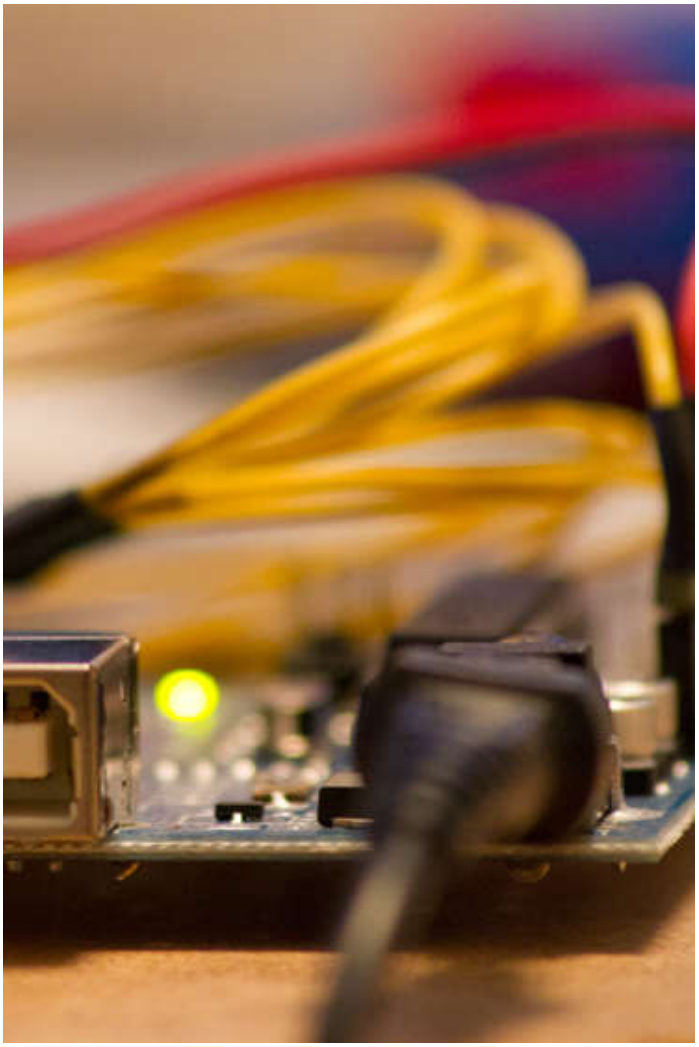


Image Notes
1. TLC5941 LED driver.
2. Outputs to LEDs.



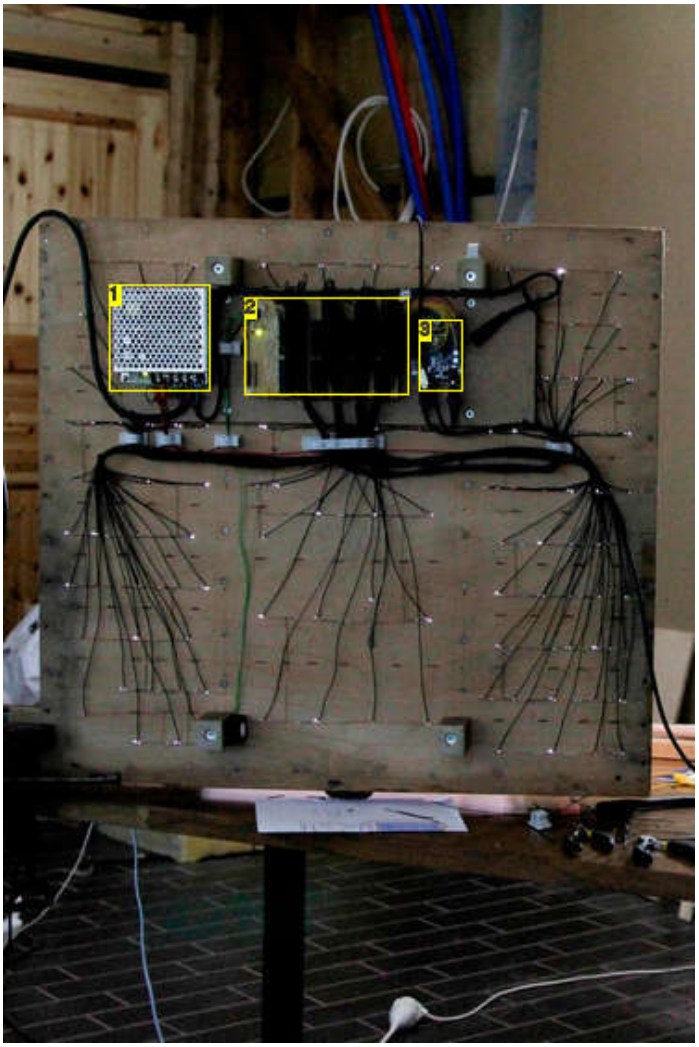


Image Notes
1. 5V power supply.
2. Driver electronic.
3. Arduino.

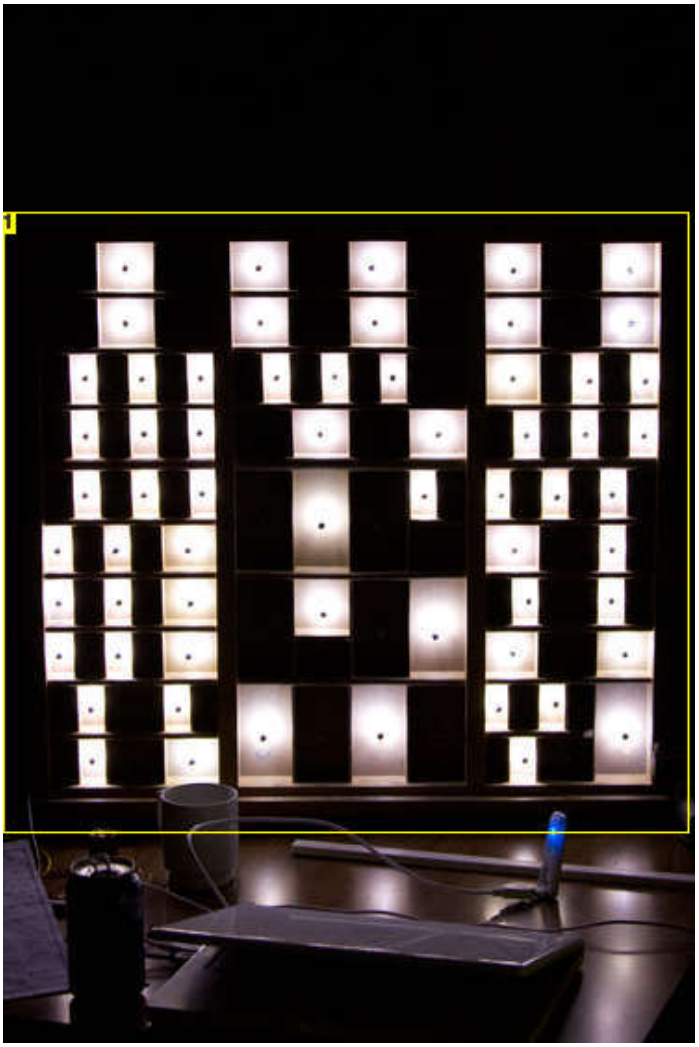
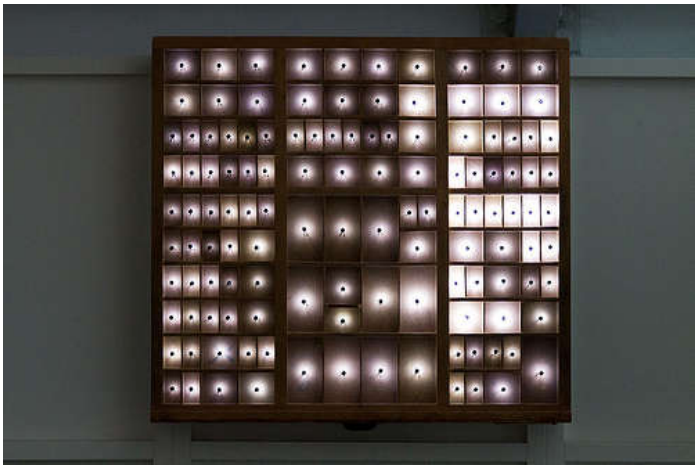


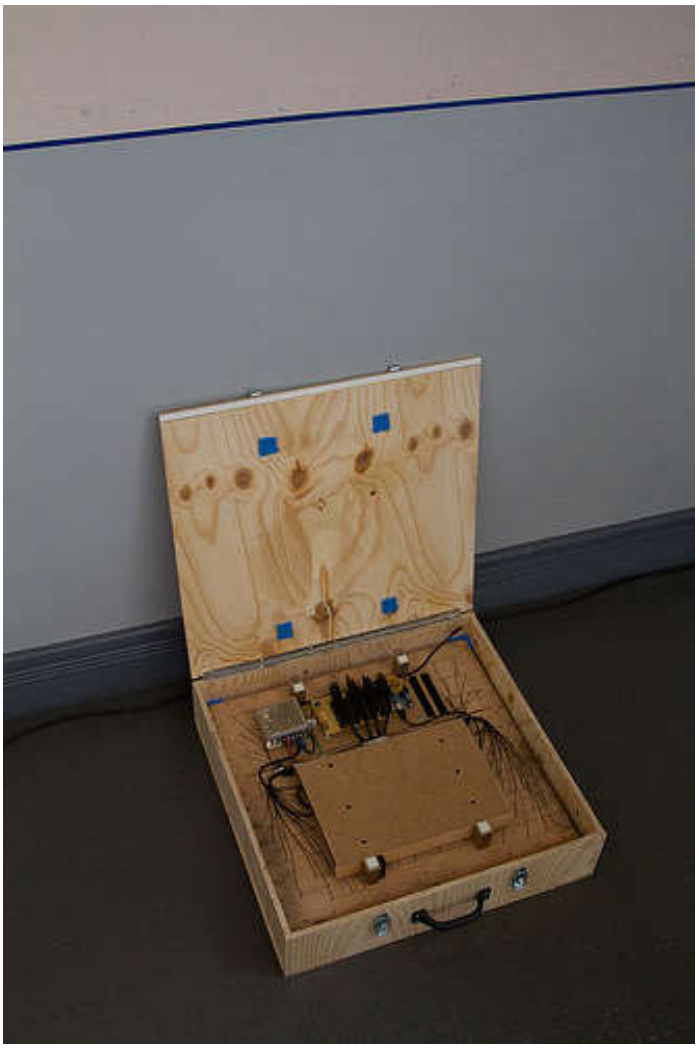
Image Notes
1. Running a test pattern.

Step 5: The documentation process

As important as not just talking – but doing something, it is to be talking about what you do.
I took loads of pictures during the process and after it was done, I took movie clips and photographs for almost a day. I wrote texts and edited the small film which can be seen in the beginning of this instructable.
Last thing to do was to update my own website and upload files to Vimeo , YouTube and flickr .




13
175
8
Post-Versal



Related Instructables

Foot Tap Amplifier by mkontopo

TurtleArt Turtle by IT_Daniher

Smallsword Choreography Shirt by grossmr1

How to have fun with Arduino (and become a Geek in the process) by john otto

How to get started with Eclipse and AVR by andy

Public Window with HTML5 by PublicWindowHSL

Office Phone by bddbbd.b

Sigh Collector by mkontopo

Sigh Collector

by **mkontopo** on March 2, 2009

Intro: Sigh Collector

Sigh v. i. [imp. &p. p. {Sighed}; p. pr. & vb. n. {Sighing}.]
1. To inhale a larger quantity of air than usual,
and immediately expel it; to make a deep single
audible respiration, especially as the result or
involuntary expression of fatigue, exhaustion,
grief, sorrow, or the like.
[1913 Webster]

Description:
These are instructions for building a home monitoring system that measures and 'collects' sighs. The result is a physical visualization of the amount of sighing, for personal use in a domestic environment.

The project is in two parts. The first part is a stationary unit, which inflates a large red air bladder upon receiving the appropriate signal. The second part is a mobile unit, worn by the user, which monitors breathing (via a chest strap) and communicates a signal to the stationary unit wirelessly when a sigh is detected.

Assumptions:
1. You have a basic understanding of construction and fabrication techniques,
as well as access to the appropriate tools and facilities.
2. You have a working knowledge of physical computing (reading circuit diagrams)
3. You are overwhelmed with the anxiety of living in a failing state, and frustrated
that most of your household objects address only physical rather than emotional health.

Step 1: Material Needed

Here is an overview of the materials that will be needed.
Each individual page has more details and links on where you can purchase some of these materials.

Physical Materials:
> 1, 4x8 Sheet of Plywood. I used a piece of shop-grade maple ply.
> 2, 2x2 for the structural frame
> ~2 yards of red nylon strap fabric
> Some loose red fabric from a fabric store
> Latex tubing (Inner Diameter: 1/8", Outer: 1/4")
> Wood Screws (5/16, 3", 4")
> 1 Rechargeable battery powered air pump (Coleman Rechargeable Quick-Pump)
> 1 unidirectional "Check Valve"
> A piece of a garden hose
> Liquid Latex & Red Pigment, or a large red balloon of some kind.

Electronics, Misc:
> 1, 20cm Stretch Sensor
> 1 red RCA cable, Male and female headers
> 1 10K Potentiometer with large sized knob
> 1 3-way toggle switch
> 2 Arduino Microcontrollers (Diecimille or newer)
> 2 9V battery clips with 5mm (center positive) male jacks.
> 2 xBee wireless modules
> 2 xBee shiels from LadyAda
> 1 FTDI cable for programming the xBees
> 1 LMC662, "rail-to-rail" OpAmp chip
> Misc Electronics components (see circuit diagrams for details).

Step 2: Build and Program Circuit. Hack into Air Pump

I like to start by getting the electronics working first, usually with a prototype of what I want to build (made from cheap exterior plywood, or even cardboard and hot-glue).

The electronics are divided into two parts. This part is the receiving end. It will receive a wireless signal from the wearable unit and use that signal to turn an air pump on for ~2 seconds and then turn it off.
Between the pump and balloon, is what's called a check valve , which lets air pass one direction but not the other.

The air pump is a Coleman Rechargeable "Quickpump" . I like it because of the rechargeable battery, and the different sized nose attachments.

Open up the pump and rework the toggle switch, so that it's bridging between the battery and one terminal of the motor. The other terminal of the motor will run to the collector of the TIP120 transistor. To do this, you'll have to de-solder the black wire from the second motor terminal, and also de-solder the lead coming from the battery charger and going to the other end of the toggle switch. Be sure to common ground the motor's battery with the arduino's power supply.

Build the circuit in the diagram below. There is also a PDF attached for higher resolution.
Program the arduino with the code supplied in the text file. You'll need to install this library .

If you don't know how to work with Arduino, here are some references so you can learn:
> Main Arduino Website
> Freeduino -- Repository of Arduino knowledge and links
> NYU, ITP's in-house physical computing site with tutorials and references.

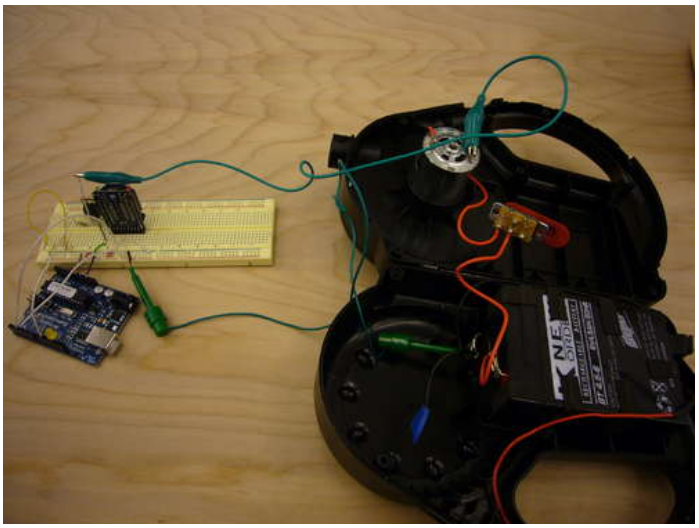


Image Notes
1. One end goes to the motor, from the TIP120's collector pin.
2. common ground with the arduino's power supply!

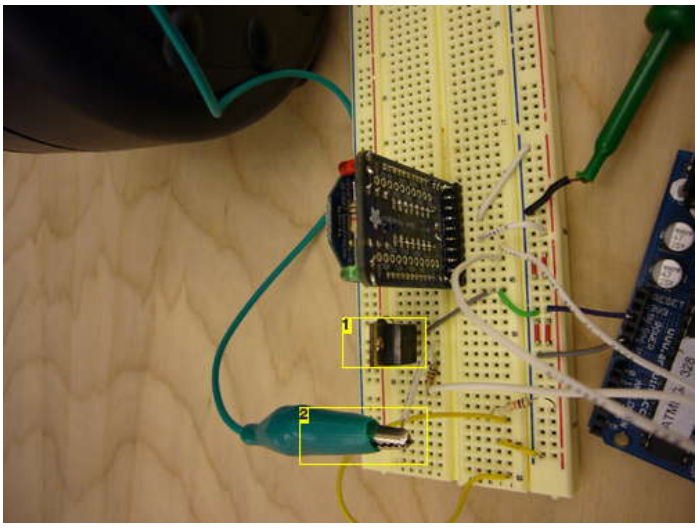


Image Notes
1. TIP120
2. To motor.

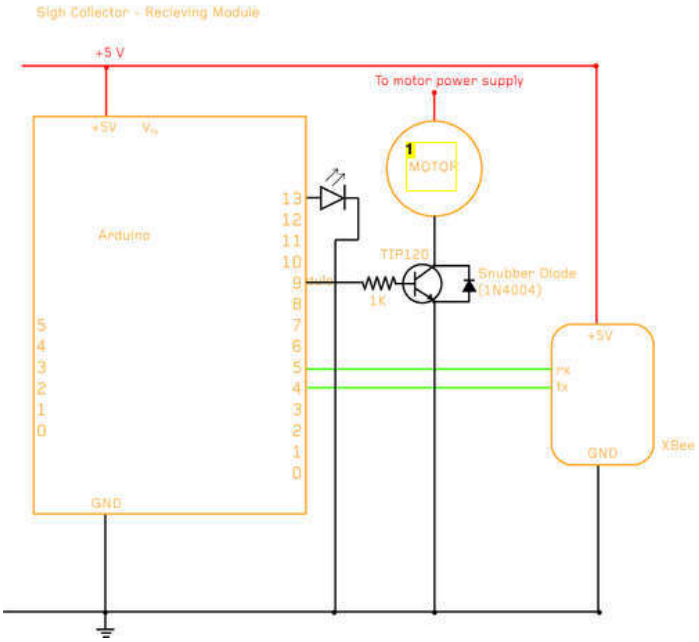


Image Notes
1. Pump.

File Downloads

circuit-sigh-recieve.pdf ((425x425) 241 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'circuit-sigh-recieve.pdf']

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

Step 3: Build the Sigh Collector main unit

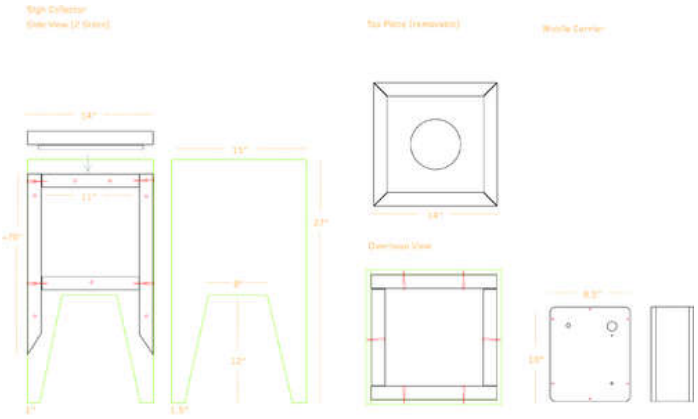
For the sake of brevity, I will not detail every step in the process of building the main unit. Suffice it to say that it can be as simple or complex as you wish; anything from cardboard and hot glue to custom fabricated or more advanced materials.

I have designed mine this way, which isn't to say it's the only way it could be done. If you care to follow or elaborate on my instructions, see the diagram below. Again, a higher resolution PDF is attached. On the diagram, you will find exact measurements and specifications on how to build the unit pictured below.

As stated in Step 2, I built mine out of shop-grade Maple plywood. It has a nice grain and cuts well. I left the surface raw.

A couple design notes:
I decided to drive all the screws in from the inside so that you wouldn't see them from the exterior. It can be tricky to sneak a drill inside of the unit, so I recommend building it in sections. I angled the bottom edges of the 2x2 frame, so that they would look a little sleeker when visible.
The top piece with the mitered corners and circular opening is removable, for easy repair of inside parts. The pump and electronics will sit inside the box, on a shelf that is held up by two of the 2x2's on the inner frame (see diagram).

The reason I built it on a frame is so that the corners would stay square. Otherwise, plywood can tend to warp. This way, also, everything can be held together by screws and therefore broken down easily into pieces.



File Downloads

sight-collector-diagram.pdf ((684x432) 231 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'sight-collector-diagram.pdf']

Step 4: Make the air bladder

I wanted a more organic, fleshy texture of my air bladder, so I cast it out of liquid latex. Liquid latex of many different sorts can be bought in a craft store, prop shop or easily on the internet. I mixed the latex with red pigment to color it, and painted it, in layers, onto the outside of a large balloon. The many layers built up to form a big, floppy fleshy balloon, with the texture I created with the brush.

A simple balloon, beach ball or even a garbage bag could replace. Check out this website for different types of large-sized balloons.

Step 5: Combine electronics with main unit. Install Check Valve and Pump

Place the air pump and circuit inside of the main unit, on the lower shelf. Now it's time to make a connection between the air pump, and the air bladder/balloon, which will sit on the surface.
We only want air to go one way, and not come out the other direction, so we use something called a "check valve" . The basic principle is that a hinged door, rubber diaphragm or ball is displace by air going one way, but then prevents the air from going back.

I bought my check valve on McMaster Carr's website; More specifically it's called a PVC Swing-check valve. I'm using the 1" diameter one. This one was attractive to me because of it's extremely low "cracking pressure", or the pressure needed to displace the barrier. < 0.1 psi !!

I used a simple garden hose to run from the pump, to the check valve, then from the other side of the valve into the balloon. The fittings are coupled and sized properly, and I used some glue to further secure them, and prevent any air leaks...

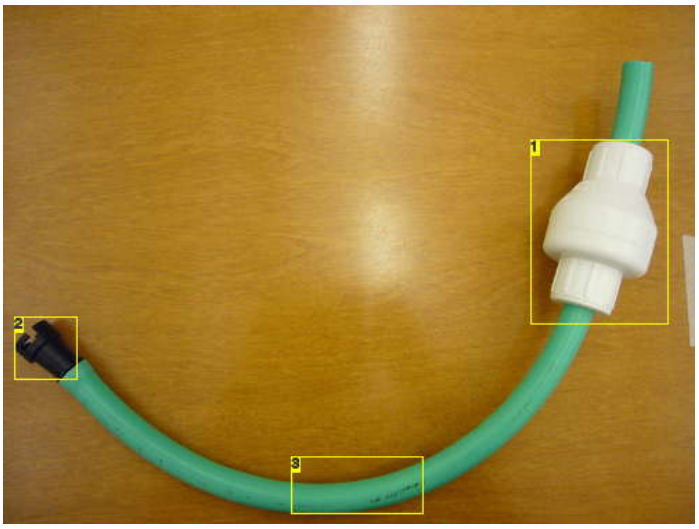


Image Notes
1. Low psi Check Valve, purchased from McMaster Carr.
2. Pump Attachment.
3. Typical garden hose from a hardware store.

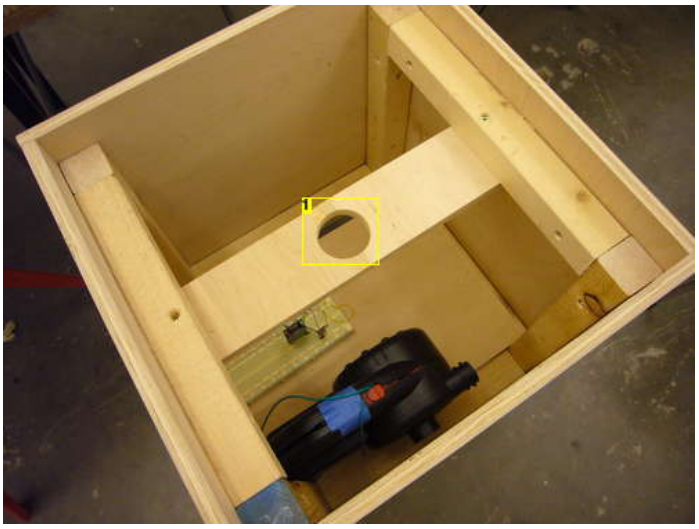


Image Notes
1. Added this piece at the end, to secure the check valve in place.

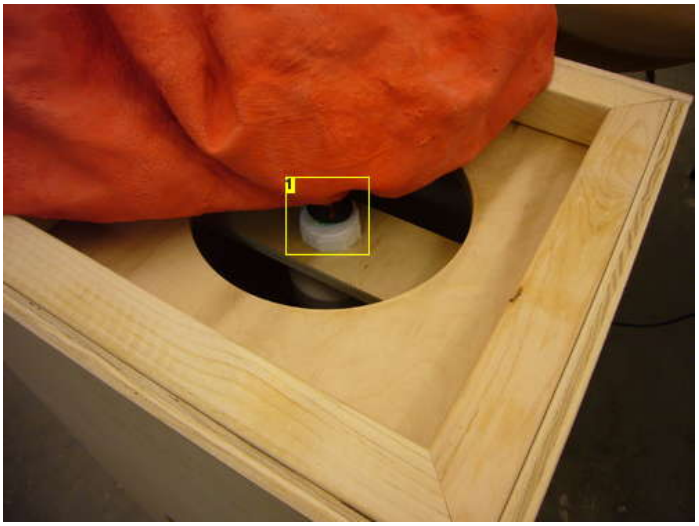


Image Notes
1. Secured the air bladder to the hose with a couple of zip ties..

Step 6: Build carrying case, Sew handle.

Sighing is monitored by a chest strap that you will wear. To hold the electronics and power supply, you must build a "carrying case". This will be mobile and will attach to the chest strap. You will carry this around with you while you perform your daily tasks and it will monitor your sighing activity. When a sigh is detected, the mobile unit will send a wireless signal to the main unit.

Again, you may follow the diagram I've provided and find measurements on how to build the carrying box. Or you may choose to make your own, unique version, or improve upon my own. I modeled mine after various kinds of medical, patient monitoring devices .

Notes:
I spliced an RCA cable in between the circuit and the sensor/chest strap (Steps 7 & 8) so that it can easily plug in and out of the box. I chose RCA cable because it's a simple way to have two stranded wires, packaged nicely with an easy to plug/unplug header. I slipped the RCA cable into a length of latex tubing, for aesthetic reasons.

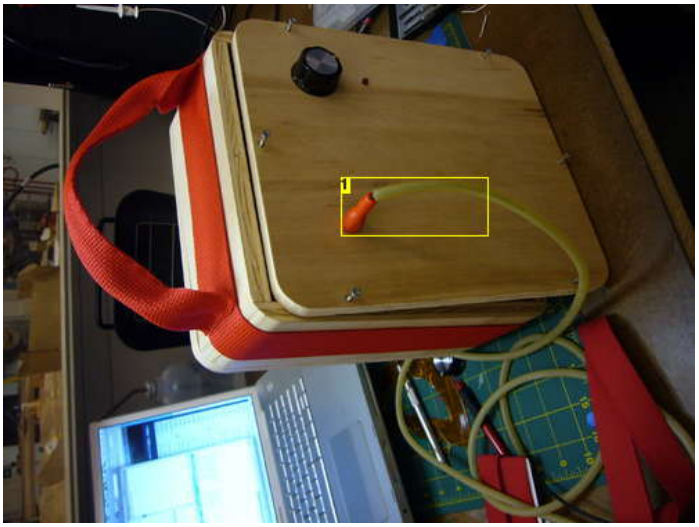


Image Notes
1. Sensor connects through an RCA cable. The cable is encased in latex tubing.

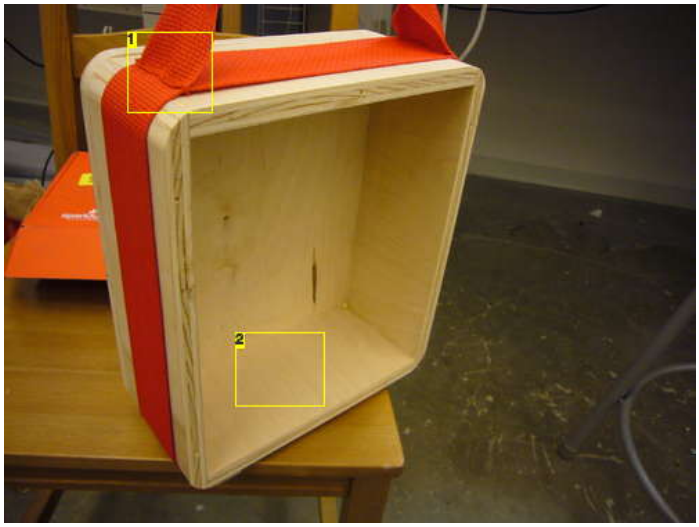


Image Notes
1. I sewed the nylon strap so it runs around the box and attaches to the handle.
2. The circuit will go in here.

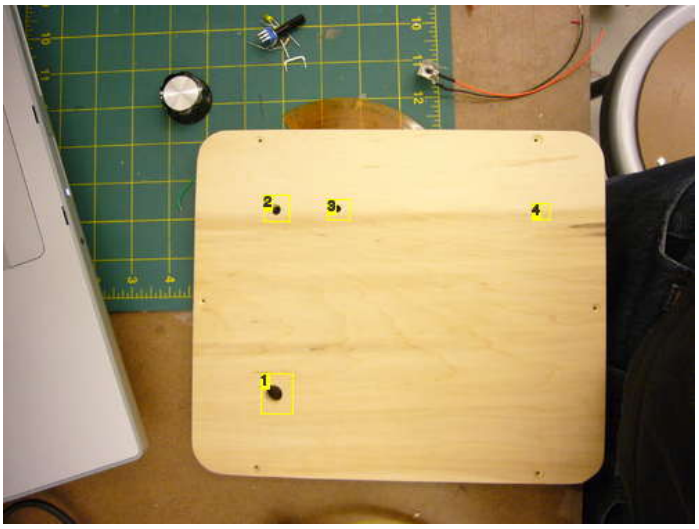
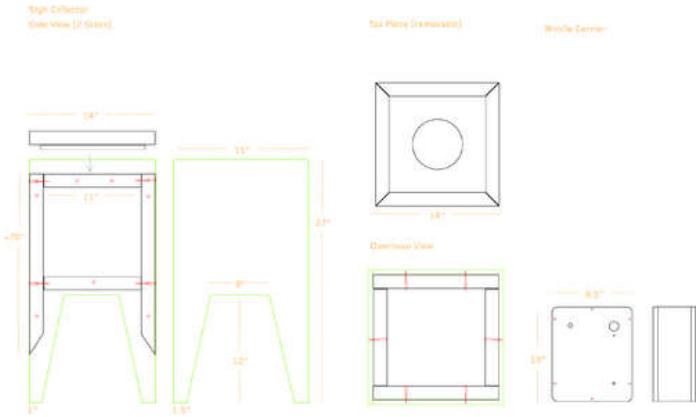


Image Notes
1. Sensor will attach here, via a hacked RCA cable.
2. A hole for the potentiometer to poke through. We'll attach the knob from the front.
3. Hole for the LED indicator light. This LED will illuminate when a sigh is has been detected.
4. A hole for the power switch will be here.



File Downloads

sight-collector-diagram.pdf ((684x432) 231 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'sight-collector-diagram.pdf']

Step 7: Build and Program circuit for sigh detection. Assemble electronics into carrying case.

Follow the circuit diagram below. A higher resolution PDF is also attached.
Program the Arduino with the provided code.

To monitor breathing, we will be making a chest strap that is outfitted with a stretch sensor. The expansion and contraction of the chest will provide us with data that we can use, in code, to extrapolate what normal breathing is, and therefore determine with a larger than usual inhalation (followed by large exhalation) is. A 10 or 20K potentiometer will be used to dial in a threshold value, which will represent how large of an inhalation is associated with a sigh.

I purchased my stretch sensor from Merlin Robotics , a company in the UK. They stock a variety of sizes. I'm using the 20cm sensor.
In my circuit, i'm amplifying the signal from the sensor with a resistor bridge and an OpAmp chip (see diagram). This is the method suggested by the manufacturer. You can find the datasheet on the internet. Note: I imagine a similar idea could be done with pressure sensor instead of a stretch sensor. You'd could attach the pressure point on the sensor to some kind of tubing and wrap that tubing around the chest.

Drill holes in the front face of the carrying case and attach the potentiometer, indicator LED, power switch and stretch sensor attachment (RCA, female) to it from the back before screwing the box back together.

I'm powering the Arduino with a 9V battery. I've got 2 of them wired in parallel so i'll get the same voltage, but double the amperage (it'll last longer).

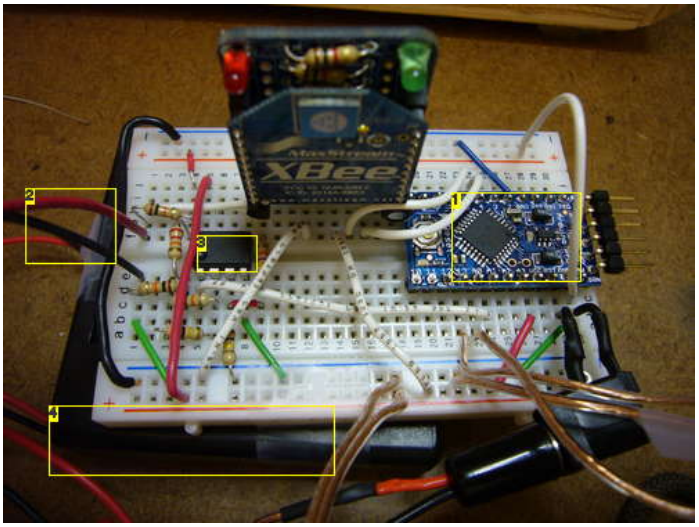
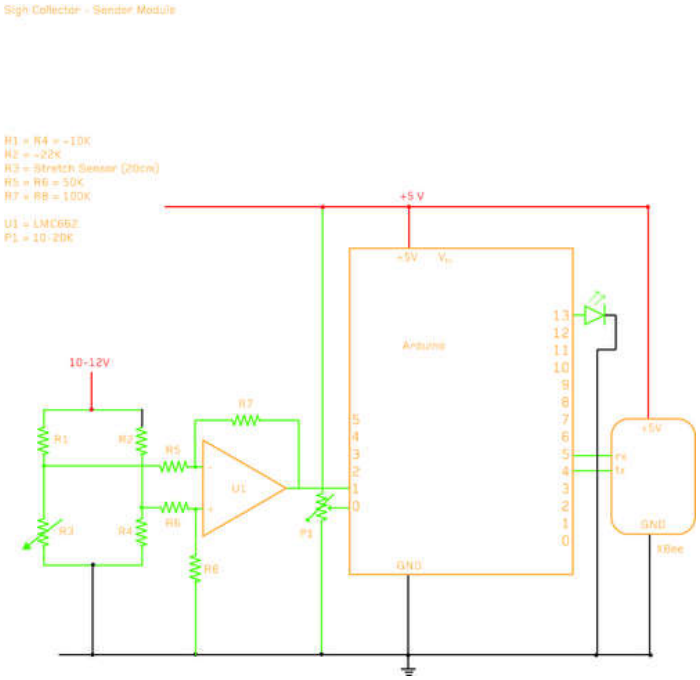


Image Notes
1. Arduino Pro Mini (just a smaller Diecimille)
2. Stretch sensor
3. Op-Amp
4. Powered by 2 9V batteries, wired in parallel for extra amperage (but same current).



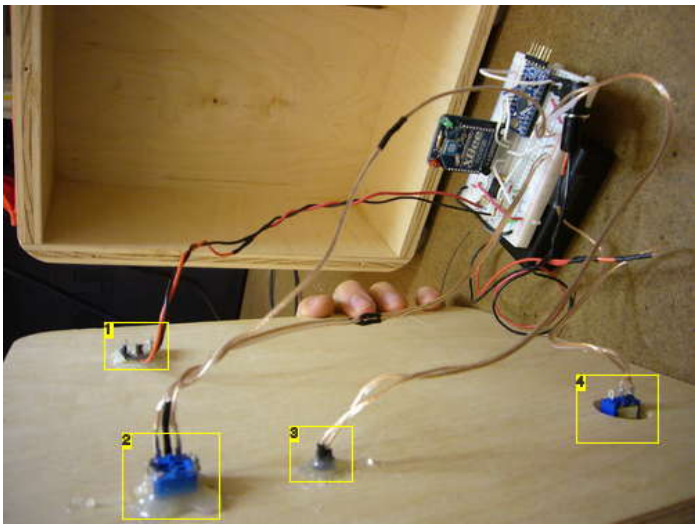


Image Notes
1. Stretch sensor connect
2. Potentiometer.
3. LED
4. Power switch.

File Downloads

circuit-sigh-send.pdf ((500x500) 247 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'circuit-sigh-send.pdf']

sc-sender.txt (2 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'sc-sender.txt']

Step 8: Cut and Sew chest strap and attach the stretch sensor.

The basic idea here, is that a fabric strap is wrapped around the chest by the lower ribs (where the most motion occurs). The stretch sensor bridges a small gap in the chest strap, the rest of which is not stretchy, so breathing, subsequently deforms the sensor as needed.

You'll have to measure the length of strap to your individual body type. I sewed an extra strip of fabric around the strap, so the wires can safely sit inside. In the front, where the stretch sensor connection is, I sewed a 'sleeve' of fabric that would loosely cover the sensor so it wouldn't get rubbed or damaged.

In the back of the chest strap, I made a simple shape (like on a backpack) for tightening and loosening the strap. I had the shape laser-cut out of clear acrylic (see image), but you can make it any way you are able to.

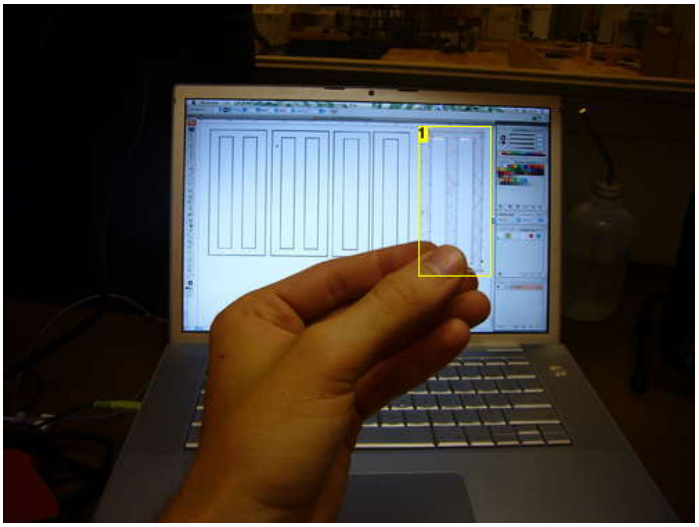
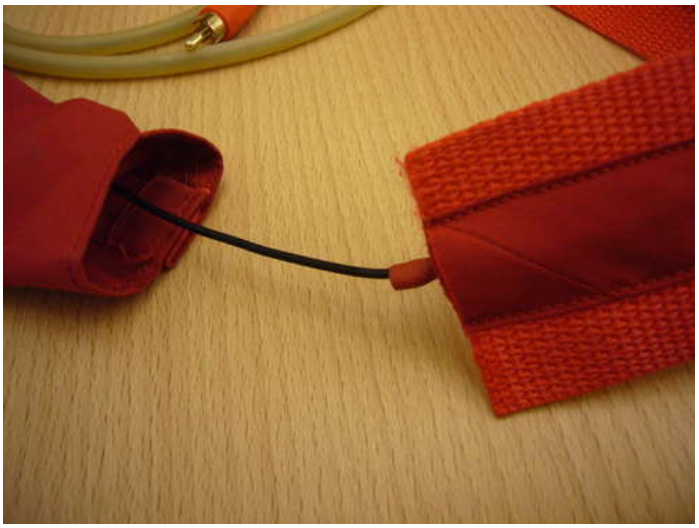


Image Notes
1. Final laser-cut piece, made from the background Illustrator diagram. This will be the strap tightener for the chest-strap.

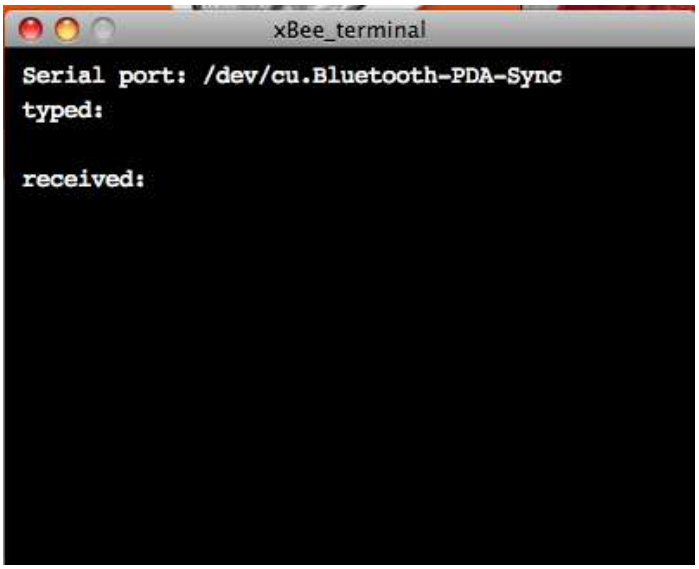


Step 9: A word on Wireless

One thing I haven't talked about yet, is how the wireless communication is being achieved. I am using xBee wireless modems. xBee's are an easy way to make a wireless point-to-point connection, or create a mesh network. To interface with my Arduino board, I used LadyAda's xBee adapter. It's inexpensive, easy to put together and there is a detailed instructional website explaining how to configure it.
Through a combination of this website, and a chapter on xBee radio's in the book "Making Things Talk " (Tom Igoe), I implemented, possibly what is the simplest use of these radios, which are actually quite powerful.

I got my adapters and xBees (+ the appropriate cable) from here .
Instructions on configuring the xBees are here .

The only thing i'm not going into is how to configure the xBees. I did it very easily (on a mac) by transcribing some code from Igoe's book that uses Processing to create a simple terminal for programming the xBee. That code is on page 198.



Step 10: Finished

Congrats! You're finished. You can now use your Sigh Collector to monitor your emotional health.

Related Instructables

Foot Tap Amplifier by mkontopo

Pacing Track by mkontopo

Smell Graffiti by numberandom

claymation improvement by shiboohi

Emotidora: Hats with Emotions by aiswaryak

Create A Laser Projector Show Without A Laser by Jixz

Cheapest storage box (Photos) by RebesaurioRex

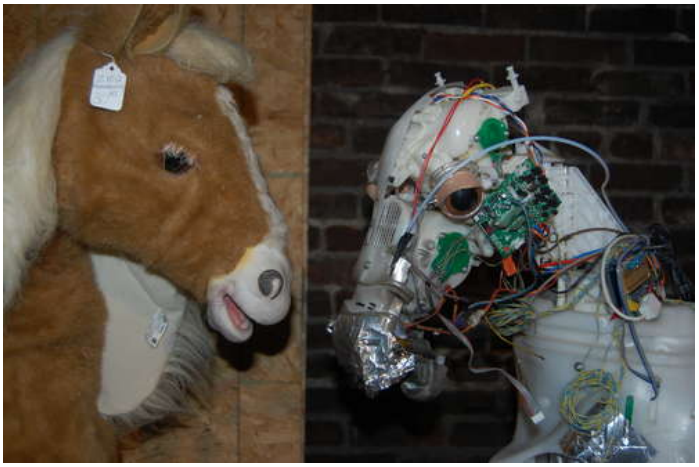
Custom Mechanical Biorhythm Computer, 3D printed by LabRat

Make a Fire Breathing Animetronic Pony from FurReal Butterscotch or S'Mores

by **lvl_joe** on November 12, 2011

Intro: Make a Fire Breathing Animetronic Pony from FurReal Butterscotch or S'Mores

For Maker Faire Detroit 2011, I displayed a hack I made to a FurReal Friends Butterscotch Pony. My fellow LVL1 Hackers and I had taken control of the motor control system of the toy and added a flame thrower to it. It seemed to go over really well with the crowd, so I am putting up the information for anyone to make there own. It was a blast to make and I hope everyone has as much fun remaking it. Just remember that this project uses Fire and should only be built and operated by no less then 2 adults with appropriate experience in fire safety and proper fire safety equipment on hand.



Step 1: Get it before you hack it

At one time, Butterscotch and S'more ponies both sold for around $300, but they seem to be discontinued. I would never suggest someone pay this much for something new, just to make it better. Thankfully, there is a fairly steady stream of them showing up on Craigslist and second hand stores.

I purchased my first Butterscotch off of Craigslist for $20. I have since picked up a second one for $25 from a peddlers mall. I commonly see them listed for ~$100, but with a little negotiation and/or patience you should be able to pick one up for dirt cheap.


raleigh craigslist > for sale / wanted > toys & games

Avoid scams and fraud by dealing locally! Beware any deal involving Western Union, Moneygram, wire transfer, cashier check, money order, shipping, escrow, or any promise of transaction protecti

Butterscotch Pony - $50 (Holly Springs)

Date: 2011-12-01, 6:45PM EST
Reply to:

Furr Real Friends Butterscotch Pony in good shape.

Location: Holly Springs
it's NOT ok to contact this poster with services or other commercial interests

PostingID: 2731113285

Copyright © 2011 craigslist, inc. terms of use privacy policy feedback forum

Step 2: What you will need.

The tools you will need vary in this project. I will try to tier it based on what you want to accomplish with your FurReal pony.

Hardware you will need:

- FurReal Butterscotch or S'More Pony
- Arduino Mega
- Wire 18g
- Solder
- Electrical tape
- Wii nun-chuck
- Wii Nunchuck breakout adapter
- 0.1" 16-pin strip male header
- 1/8th OD ptfe tube (trade name Teflon)
- Bowden cable (brake cable for the back wheel of a bike)
- Scrap PVC tube around 3" at about 1' long
- Scrap plexi glass

Tools you need

- Wire Strippers
- Razor blade
- Phillips head screwdriver
- Flat head screwdriver
- Multimeter
- Soldering Iron
- Computer to program the Micro Processor (Any OS)

Image Notes
1. Wire cutters
2. Wire Stripers
3. Wire
4. 3v (2 AA battery pack)
5. razor
6. Multimeter
7. Grill igniter
8. Soldering Iron
9. Solder
10. Hotglue Gun
11. Wii Nunchuck
12. Screwdrivers

Step 3: Removing the skin: Head first

Before you get into the really fun parts, you will need to skin your pony. I started at the head as it already has a zipper. Move the mane out of the way, and locate the zipper at the base of the neck. You will find that the zipper pull has a cap over it to prevent it from unzipping. Simply break off this cap and unzip the skin from around the head.

Step 4: Removing Skin: ENT

Now we are going to remove the skin from around the face. The face is attached in 4 areas: the ears, eyes, nose, and mouth. Pull the skin up the back of the neck so you can get inside the back of the ponies head.

The ears break off easy (one was already missing when I got to this point). The other broke off when I was trying to get the skin off. This is not a problem as you can glue them back on easily. If you wish to keep the ears attached, you can cut the cloth around the ear holes with a razor. If you are OK with taking them off, a hard tug should pop them right off. Once the ears are removed, pull on the fabric where it tucks into the head. This will rip the seams, freeing it up.

Roll the skin further down, and you will get to the eyes. The fabric is sewn into the top and bottom of the eyes. Simply cut the stitches here to detach the fabric. Try to not look your pony in the eyes when you do this, as you may start feeling bad about what you are doing.

Moving down to the ponies nose, there are 2 pegs holding the rubbery snout on. One in each nostril. These slide out without much trouble, away from the body in a parallel fashion. The rubber on the snout is thin, so try to get a grip on the hard plastic with your tool as not to tear it.

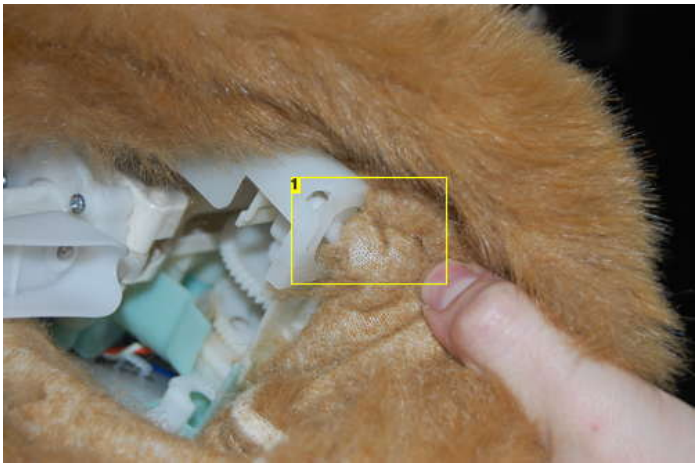


Image Notes
1. Fabric tucked in around the ears. I already removed the years at this point.

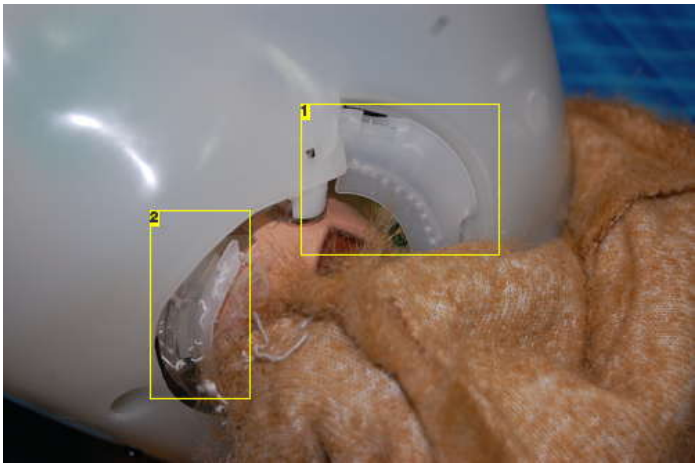


Image Notes
1. Fabric sewn into the plastick under the eye.
2. Fabric sewn into the top of the eye.

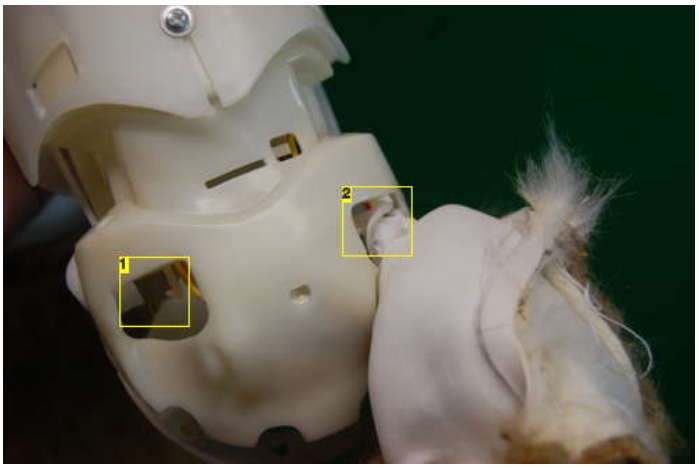


Image Notes
1. already removed
2. Slide out away from the body and it should pop right out. be careful not to beak the rubber on the skin.

Step 5: Remove Skin: Straight from the horses mouth

The last step of peeling your ponies face off is to removed the skin from the mouth. The the skin on the upper and lower jaw are both connected in their own way.

To removed skin from the upper jaw you just need to fold the face down until you see a horse shoe shaped piece of plastic around the mouth. It will have 4 pegs pushed up into holes with 3 legs that close around them. You will just need to bend 2 of the legs on each peg back and they will pop right out.

The bottom jaw will most likely have popped off by this point. We need to remove the jaw plastic from the rubber. It for the most part will pill off but it will take some time. I also had to cut some spots with a razor that where to fused.

After you get the lower jaw removed slide it back into the slot under the chin and glue it into place.

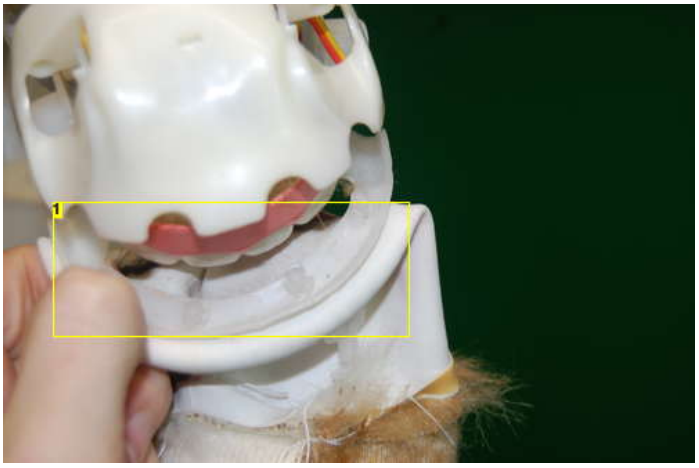


Image Notes
1. The 4 pegs holding the upper lip on.

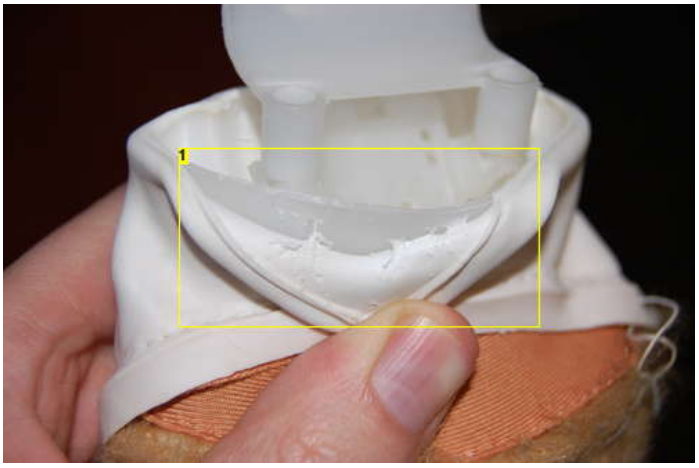


Image Notes
1. Removing the rubber from the plastic. pulling will do it for most parts but a few will have to be cut.

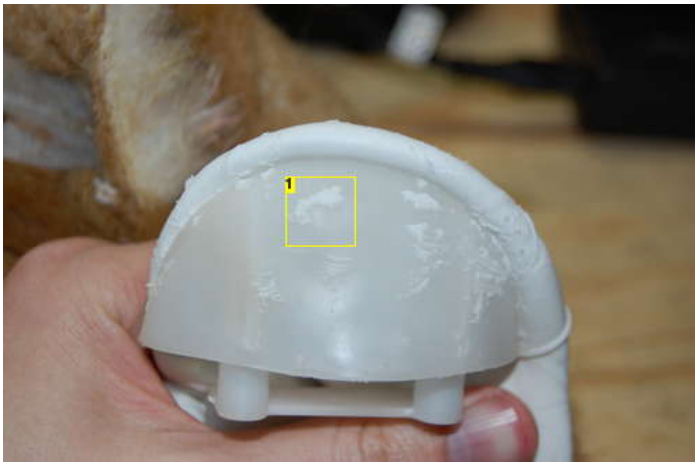


Image Notes
1. Part like have a little rubber on then is from where I had to cut it with a razor.

Image Notes
1. This part slip into a slot under the chin. It will not stay so you will have to hot glue it.

Step 6: Remove Skin: The body

Removing the skin from the body is a lot of the same. there is a few more places that they sew the fabric to the plastic. I followed the seem in most parts. just cutting a little bit of the thread then pulling it apart.

On the underside pull the velcro open. In the back side where the Velcro ends you will find a zipper leading up to the tail. There will be stitches at each end holding it together but no slider. cut the stitches and it will come unzipped. There will only be a small area holding the zipper area to the Velcro area. Cut this small bit of fabric and we will move to the front.

The frond end of the Velcro has a small stitch leading to a T intersection. Unstitch this area then unstitch along both sides of the T till you get to the legs.

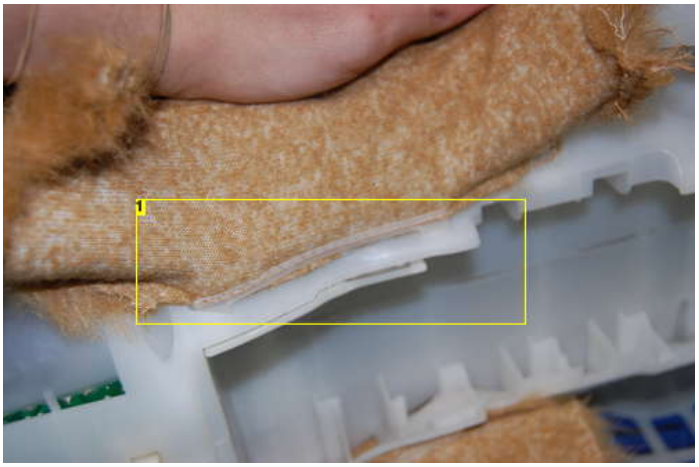
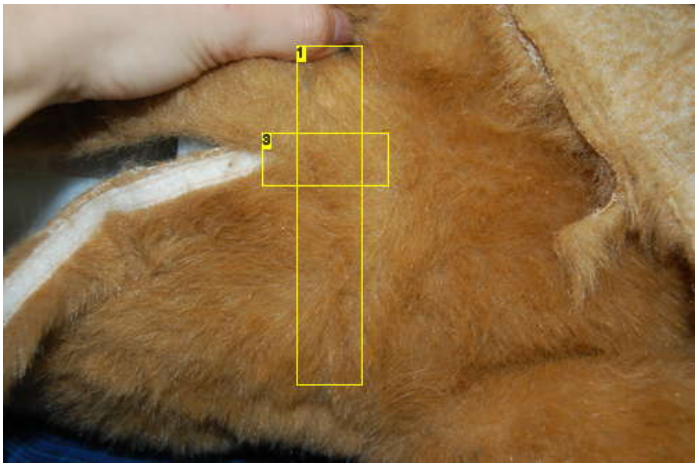


Image Notes
1. Fabric sewn to the underside of the pony.



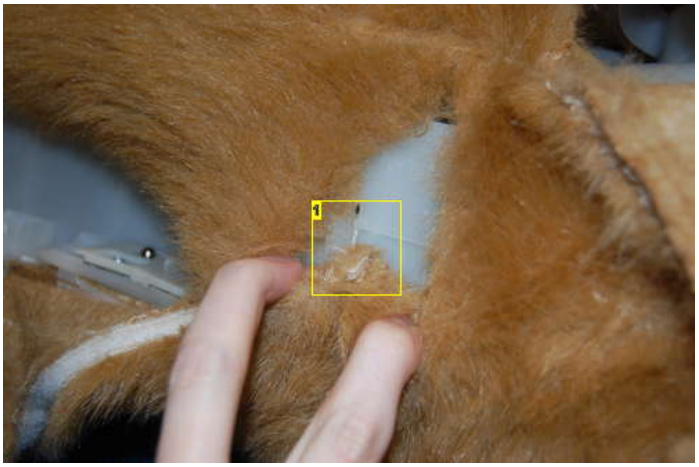


Image Notes
1. Seem that crosses from one leg to the other in the lower front of the pony.
2. Seem continues from the Velcro to cross seem.
3. Seem continues from the Velcro to cross seem.

Image Notes
1. Another place it is sewn to the plastic.

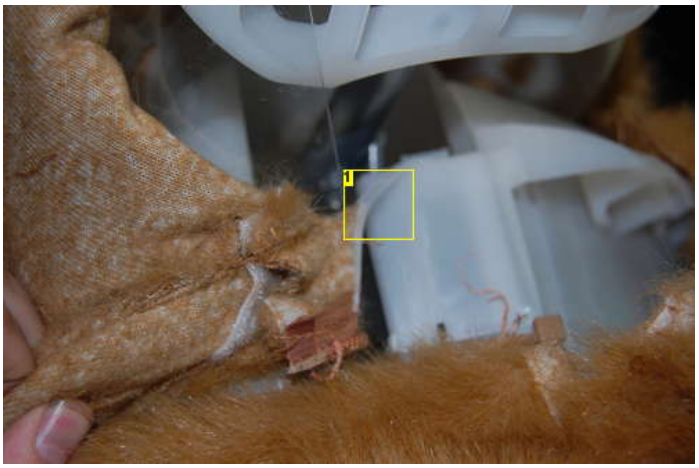


Image Notes
1. and yet again.

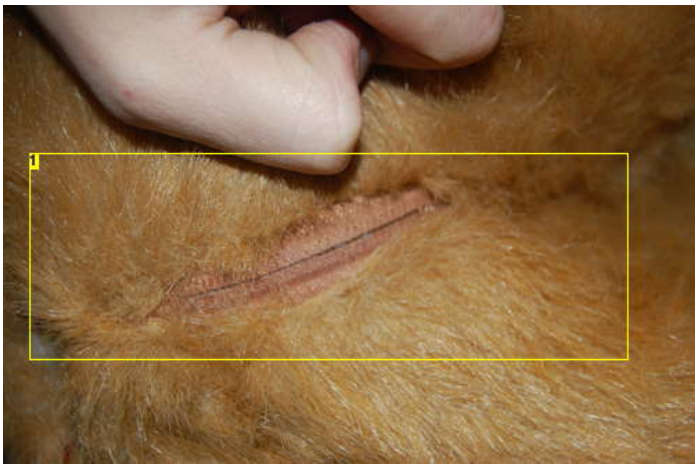


Image Notes
1. Zipper running from the tail to the Velcro under the pony. It is sewn over at each in but after you cut the threads it just unzip without a problem.

Step 7: Removing the skin: The legs

There is an inner seem along each leg. Snip the stitch and tear down each of them till you get to the feet. The feet are held on with a different fabric that is then looped around a rope inside of the hooves. Just cut around the bottom of the leg removing this other fabric and freeing the rest of the leg.

At the top of each leg you will need to remove some stitching leading around the intersection of the inner leg and the lower side of the body. When you get close to the stitches running along the bottom of the pony that you have already cut, snip away the fabric in between. You should not be able to lift the skin up all the way around the head.

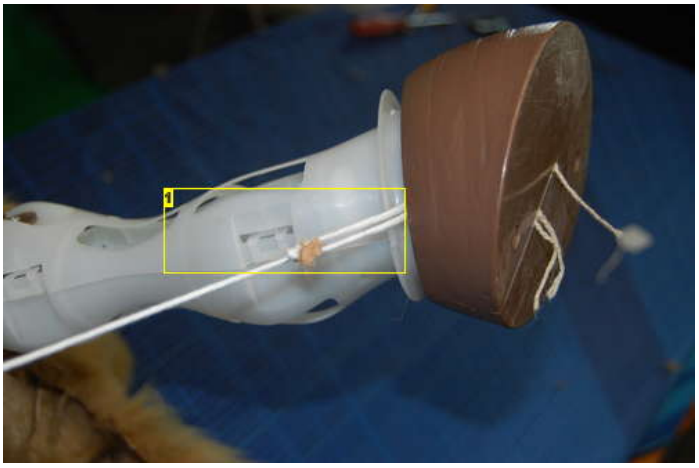


Image Notes
1. The string that was inside of the fabric loop

Image Notes
1. What you should have at the end of this step

Step 8: Removing the skin: the Neck

The very last step is to remove the part that is holding the fabric to the neck. I did this by cutting through the lighter, fur-less fabric at the neck. After I removed it, I found that it was held on by a zip tie inside of that loop of fabric. You can do it my way, or you can insert the wire cutters into the fabric at the nap of the neck and simply cut the large zip tie. This should allow you to remove the skin completely.

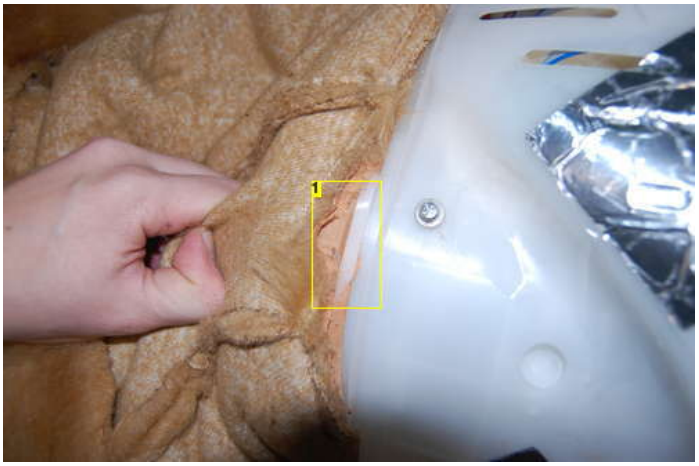


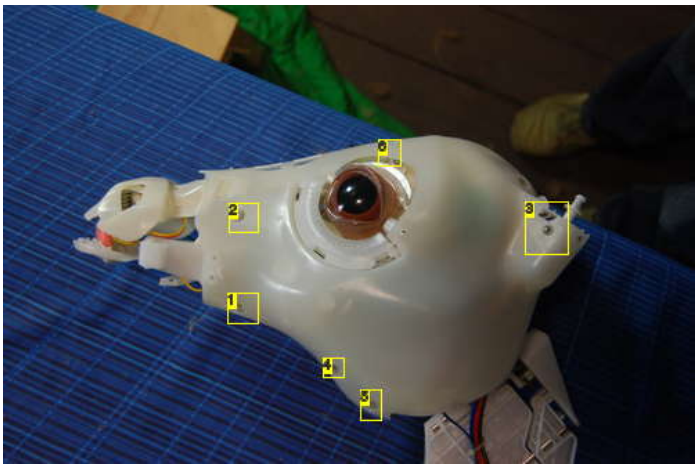
Image Notes
1. the non fur fabric looped around a zip tie.

Image Notes
1. What you should have at the end of this step

Step 9: Removing the face

To get to the main circuit board in the head, you need to removed the left side of the face (the pony's left, not yours) There are quite a few screws that you will need to remove to do this. I tried to mark all of the screws in the images below, but if it will not come loose after removing those, just look around for extras I may have missed. There is also a clip that holds the snout onto the rest of the face. This clip was difficult to remove and I ended up marring the face a bit with a screwdriver and wire snips.

Under the face you will find that the circuit board is held in place by 4 screws. Remove these screws, as we will do most of our work from the lower side.



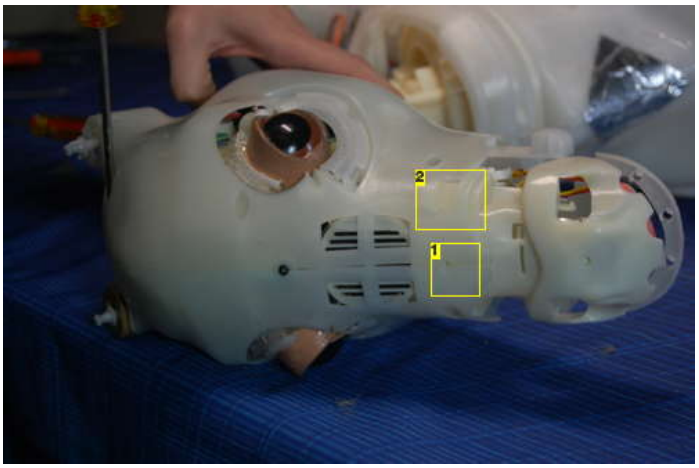
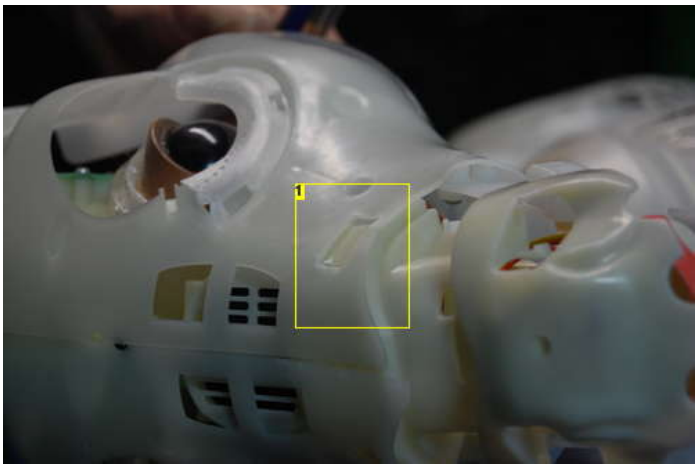


Image Notes
1. Screw
2. Screw
3. Screw on the outside and inside of the ear mount
4. Screw
5. Screw
6. Screw

Image Notes
1. Screw
2. Clip



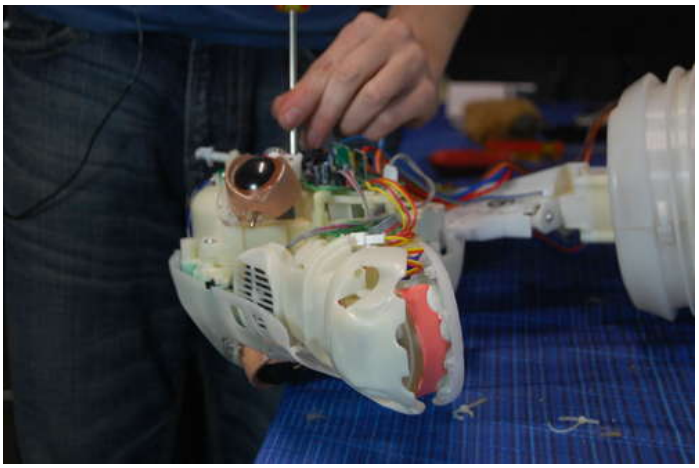
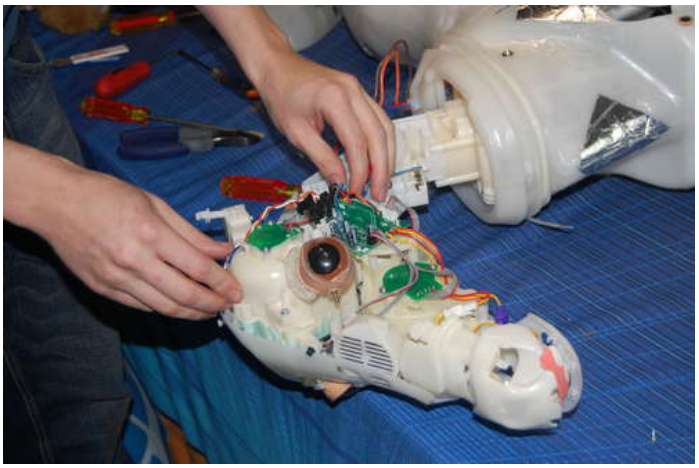
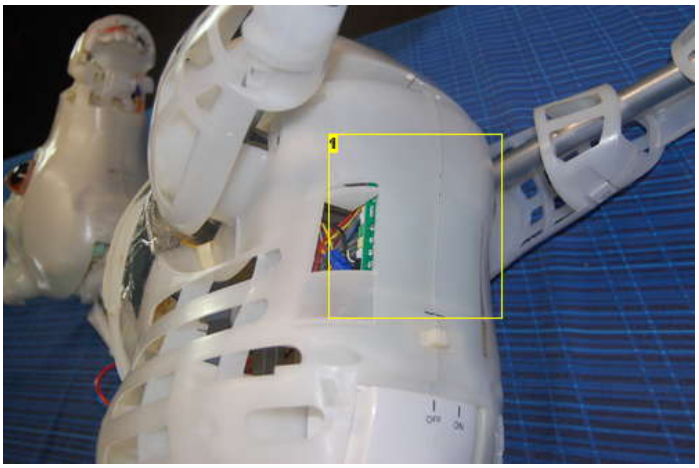


Image Notes
1. Clip



Step 10: Getting access to the Circuit board in the lower body.

I removed every screw I could find from the pony and still could not get the body open. I could not see any clips I could open, or anything else I could remove to release it, so I did the next best thing and dremeled a hole in the stomach. This ended up working out in the end as it provided a good place to put the fuel for the flame thrower. You will want the hole large enough to allow your PVC tube to just slide in.



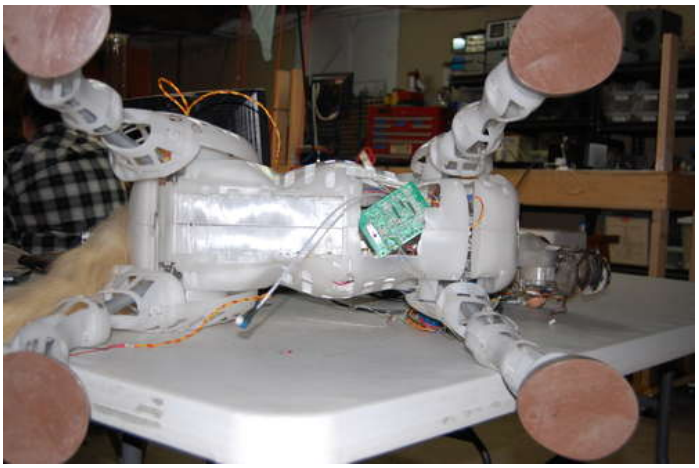


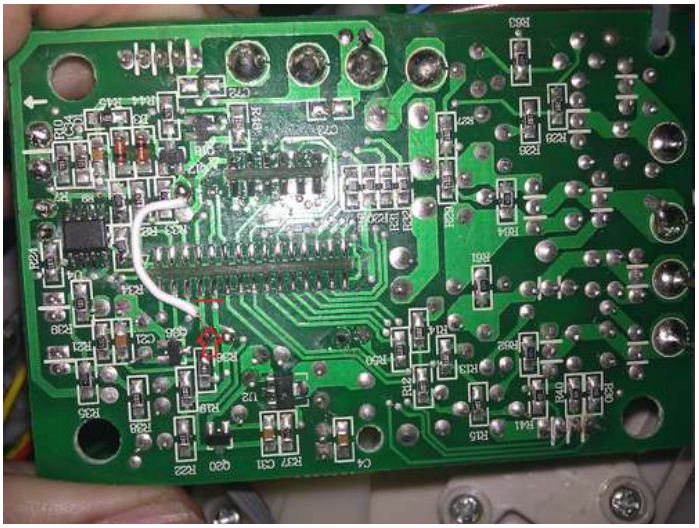
Image Notes

1. Circuit board for the body. location where we will load the fuel.

Step 11: Cutting the power to the Microcontroler

To take control of the pony we will cut the power and the ground to the Micro-controller that is currently controlling the pony. There are two controllers in the pony, one in the head, and one in the body. The one in the head sends commands to the one in the body so we will only be cutting power to the head, and this will take care of both.

To do this, cut the trace going to the 4th and 5th pin on the larger of the two boards sticking out at a right angle. The traces will be on the back side of the board. The 4th pin should have a white wire soldered to it. Using a razor you should be able to cut the trace without a problem.



Step 12: Tapping power for the Arduino

Now we need to power our Arduino, but there is no need to add another battery when we already have the 6 C cells powering the pony itself. Tapping into the power being pulled into the circuit in the pony's head will give us around 9v. I had a few 9v wall warts laying around so I cut the cable off of one with 5.5mm/2.1mm barrel jack on it. I tied this into the connectors going to the head. You can also purchase an adapter from adafruit ,as that will be much easier.

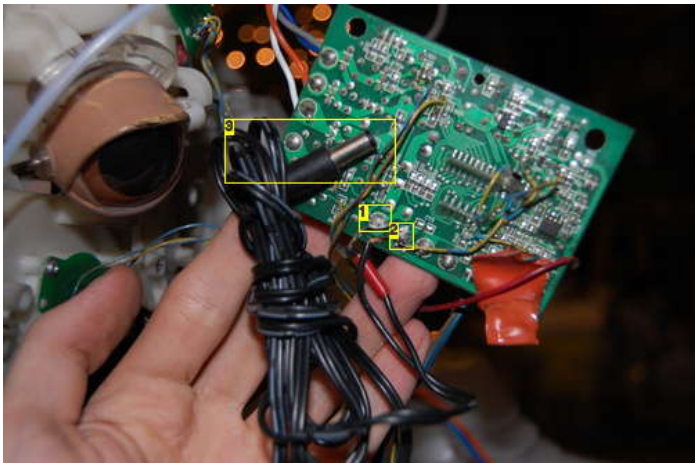


Image Notes
1. V
2. V-
3. 5.5mm/2.1mm barrel connector

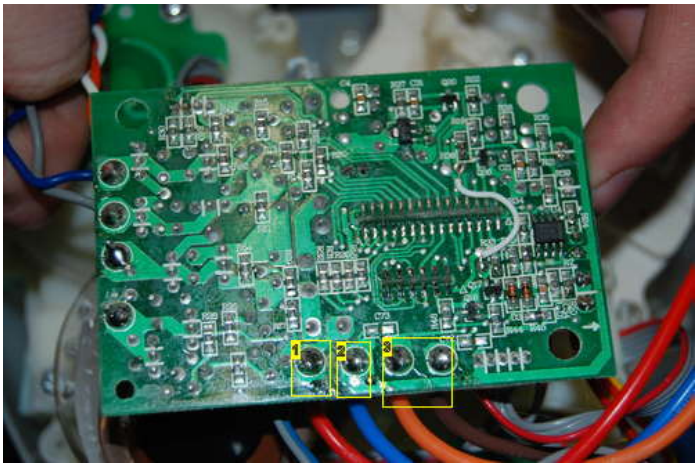


Image Notes
1. Power
2. Ground
3. Data lines

Step 13: Tapping the lines into the motor control circuit.

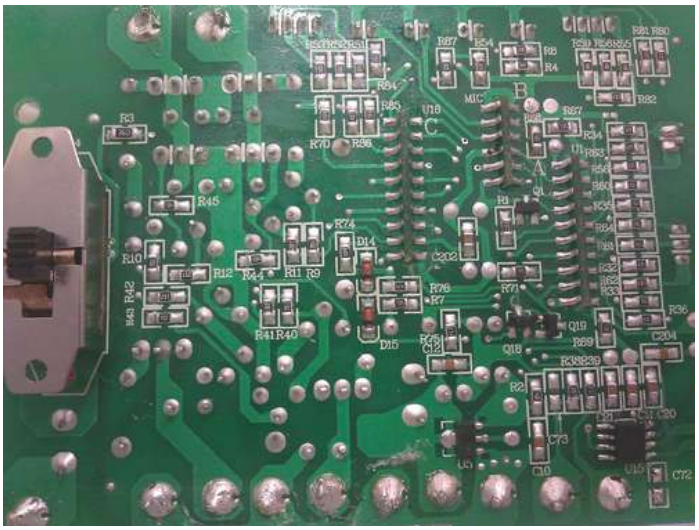
You will need to tap into the the lines coming out of the motor control circuit coming out of the micro-controllers. We will do this at 4 spots on each of the Circuit boards.

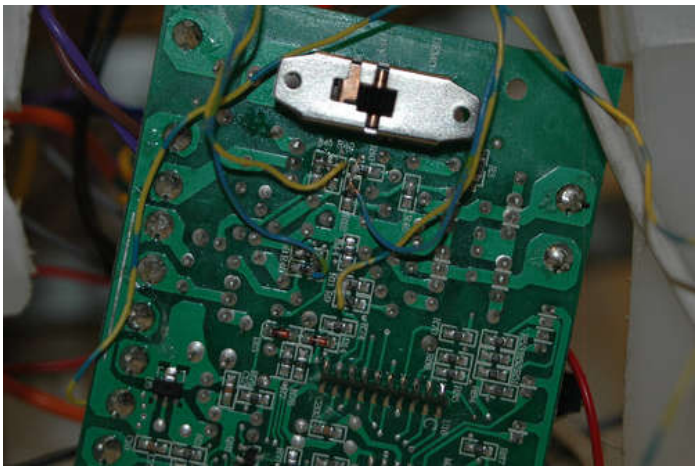
On the board in the pony's head you will need to solder your wires into R14 ,R15, R27, and R28

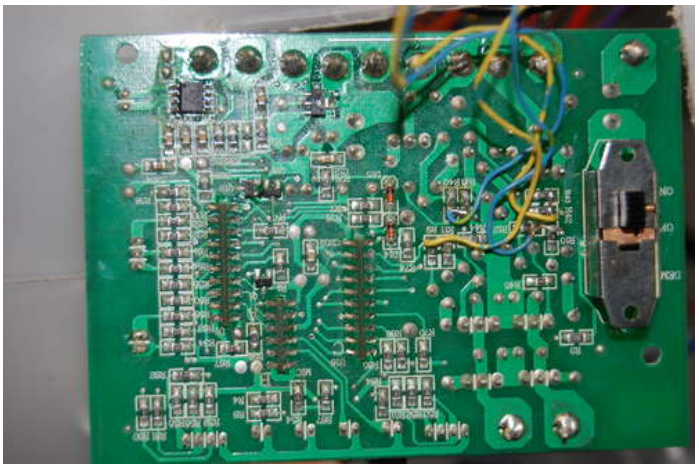
R14 and R15 move the head up and down plus open and close the mouth.
R27 and R28 move the head move left or right as well as move the eyes and ears.

On the board in the body, you will want to solder you wires onto R10,R42 and R11,R41

R10/42 move the head left and right
R11 Moves the tail (only one way)
R41 bobs the head up and down at the neck (moves one way around in a circle like the tail)







Step 14: Taping into the encoders.

There are 4 encoders that will tell you the position of the head. Two of them are located in the head and 2 of them are located in the body. The two in the head are easy since you can see them when you take the pony's face off.

Solder a ~2' long wire to each of these encoders. I used 18g wire.

For the encoders in the body, I was unable to find an easy way to get to them, so I cut the end off of the wire. We will solder the wires from this cable right into the bread board so strip them and you are finished with them for now. Try to leave these wires as long as you can.

Image Notes
1. Reads the state of the head up and down as well as the mouth open and closed. Listed as Wipe1 on the board
2. Reads the state of the head tilt left and right, plus the movement of the eyes and ears. Listed as Wipe2 on the board.

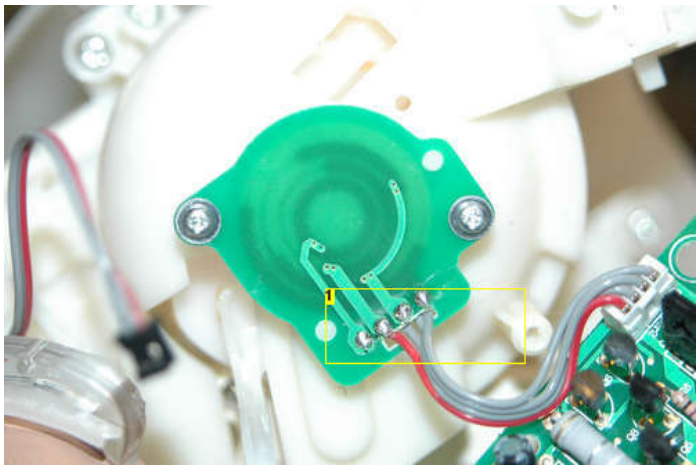
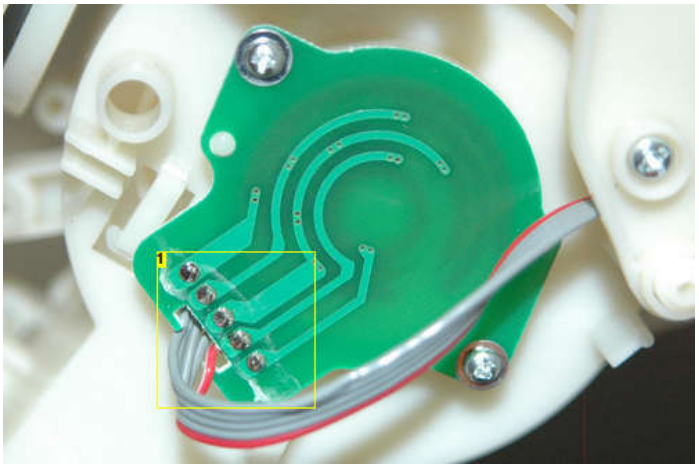


Image Notes
1. Solder onto the encoder rather then cutting the cable.



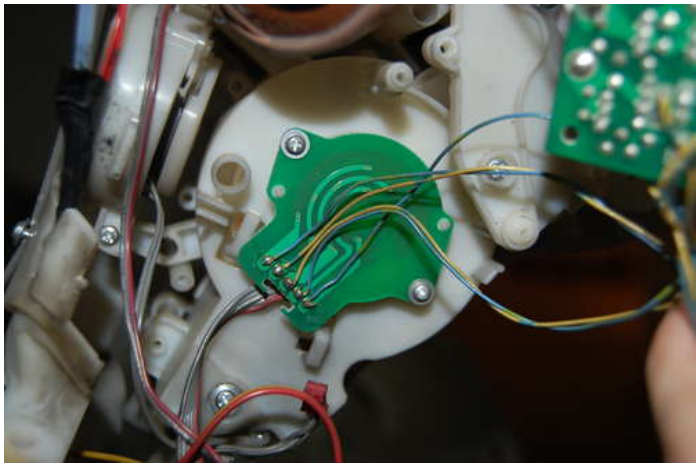
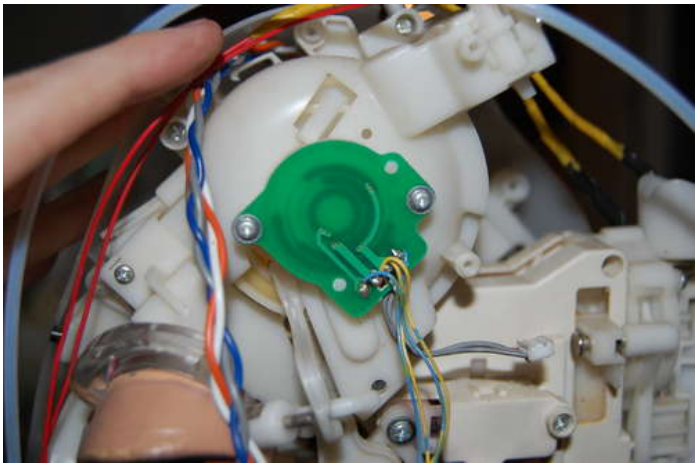


Image Notes
1. Solder to the pads rather then cutting the cable.



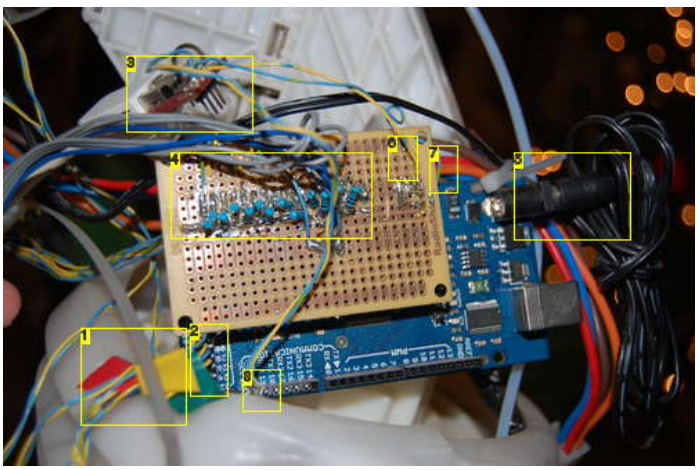
Step 15: Getting the morors and sensors connected to the arduino.

For the current code you will want to have the pins as such

Resistor label --- Pin on the Arduino

R14 Pin 23
R15 Pin 25
R27 Pin 27
R28 Pin 29
R10 Pin 37
R11 Pin 35
R41 Pin 31
R42 Pin 33

To get the pins connected I soldered them to the end of .1 male header.



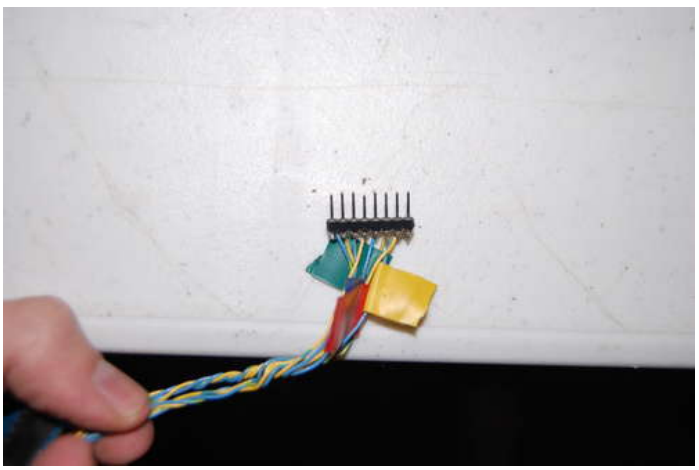
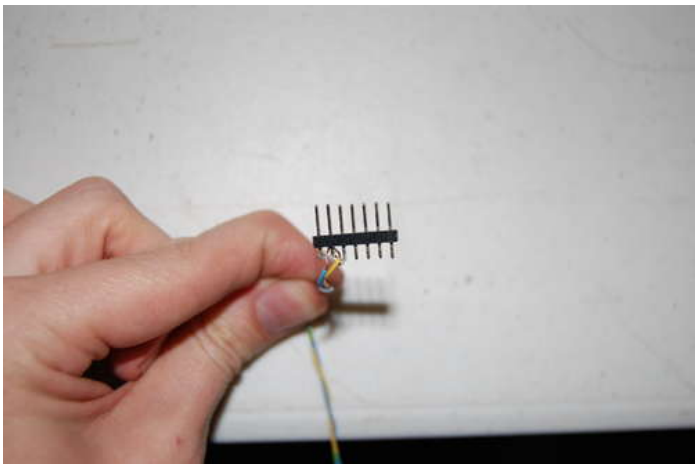


Image Notes
1. Electrical tape so that I know what pin is connected to what motor.
2. Going to the motor control
3. Nunchuck breakout board.
4. Pull downs for the encoder.
5. Plugged into the pony's power supply.
6. Ground pin for nunchuck
7. 3.3v going to nunchuck
8. I2C pins for the nunchick

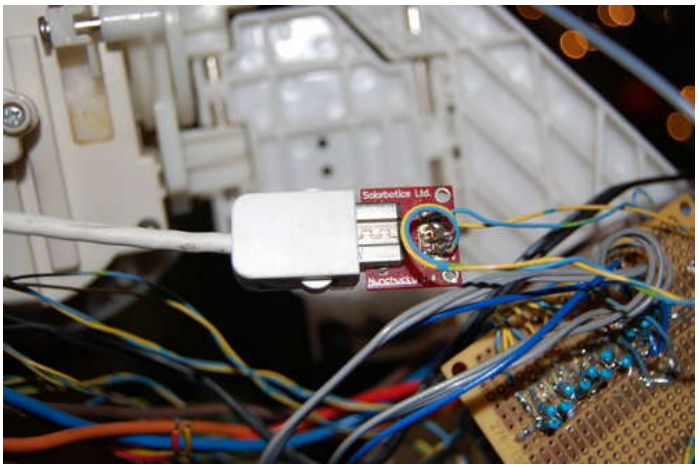


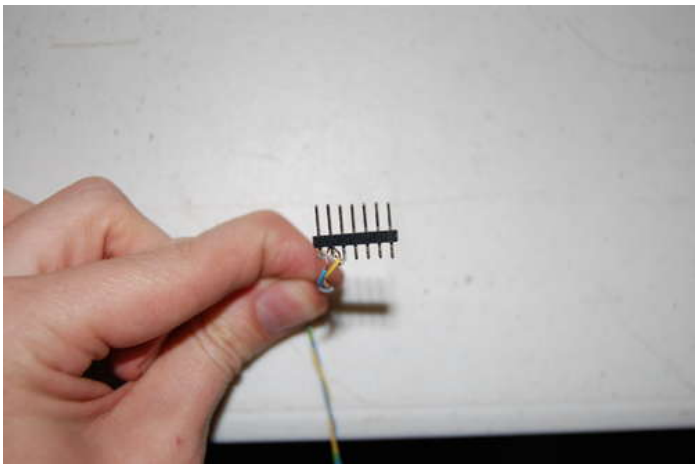
Step 16: Connecting a wii nunchuck into the system.

Now you will need to connect the Wii nunchuck breakout board from adafruit. If you want to run the wii nunchuck at 5v you can just use the .1 pitched pins that are on the breakout board, you can set the the input pin 19 as 5v output and pin 18 as input. I connected mine with wires and chose to play it safe by running it at 3.3v.

On your mega, connect it as such
Gnd: Ground
3.3v: 3.3v
Data: 20
Clk: 21

You will want to check the arduino wiring documentation if you are using something other then a mega





Step 17: The Arduino Code.

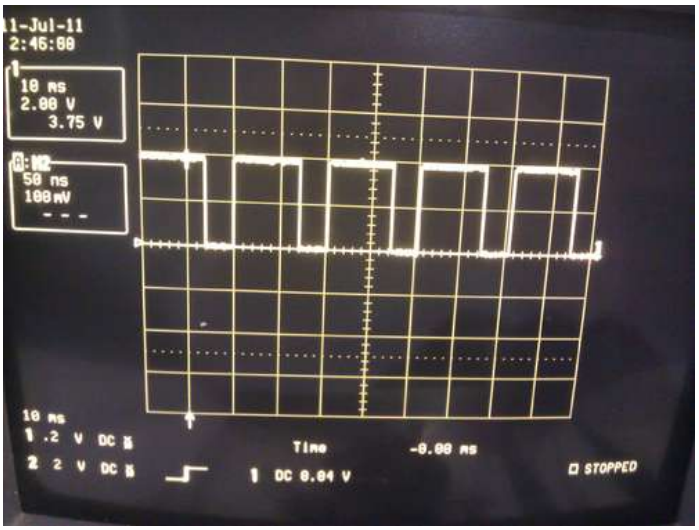
The code should be loaded on the arduino mega using the arduino IDE . Before doing so, you will need to put the modified wii nunchuck file into your arduino libraries folder. It should be in the root of your arduino IDE install. It should look something like "C:\User\joe\arduino-0022\libraries**WiiChuck** **WiiChuck.h** . Make sure that you put it inside of a folder named WiiChuck so that it can be found by the Arduino. After you load this on your Arduino, you should be ready to start moving the pony around.

The sketch has to bitbang the PWM sent to the motor controls as there are to many pins to do it on PWM pins. I think it runs too slow for the Arduino to do it with hardware anyways.

Current controls work like this:
Push the joystick one way and the head will start moving that way from a dead stop.
Move it the opposite direction from the way it is currently moving and it will stop moving.
Move it up or down and it will move that way from a dead stop.
Move the opposite direction then the head is moving and it will stop moving.

C moves the Tail
Z shakes the head

The mouth moves when the head is moving up and down.
The ears and eyes move when the head is shaking.



File Downloads

FirePony.pde (8 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'FirePony.pde']

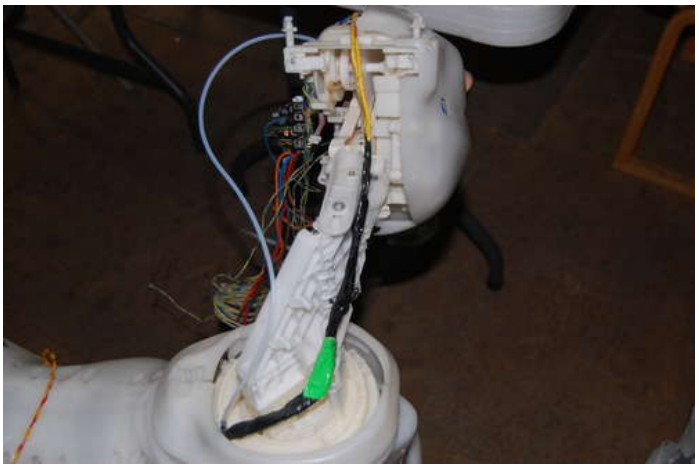
WiiChuck.h (6 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'WiiChuck.h']

Step 18: Getting the fuel to the head

The fuel we are using for the fire is automobile starter fuel. This can be purchased at most any store that sells automotive supplies.

To attach the fuel source, we will connect one spray caps from the starter fluid to a teflon tube that runs from the hole in the belly up along the neck and then out just over the nose where we will place our igniter. The tube will not be the right size to fit into the cap so use a drill bit that is the same diameter as the tube to bore it out until it fits.

Be prepared for some frustration as Teflon is very hard to push into things do to its extremely low Coefficient of friction. Once you get it in, put some hot glue around the joint to hold them together.



Step 19: Building an ignition system.

The ignition system is an electric grill ignitor that has had the leads on it extended. Just over the mouth (or wherever you want the fire to come out) we have attached a metal plate that is connected to one lead of the ignitor. The Teflon fuel tube is epoxied right over this lead. Finally, the other lead off of the ignitor is connected to a wire on the other side if the fuel tube, so that the aprk will arc across the tube.

The fuel has nothing to burn so if the ignitor is not running it will blow out in a small breeze. When modifying the ignitor you will want to connect one 6 foot wire to each lead. Ours had 4 leads, if you have the same type you can put electrical tape over the other two so that they do not spark inside of the body. You will also need to insulate the leads that you extended with tape, shrink tube, or hot glue to keep it from sparking at the ignitor rather than at your extension leads.

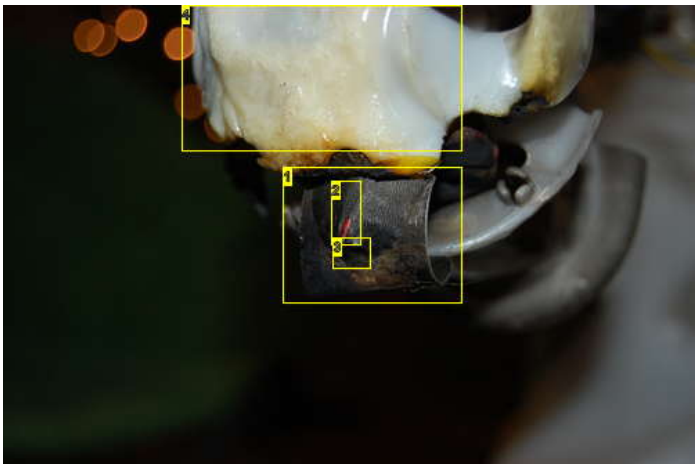


Image Notes
1. Aluminum sheet hooked to one half of the spark generator.
2. Wire hooked to the other side of the spark generator.
3. Exit point for the fuel.
4. What happens when the plastic gets too hot.

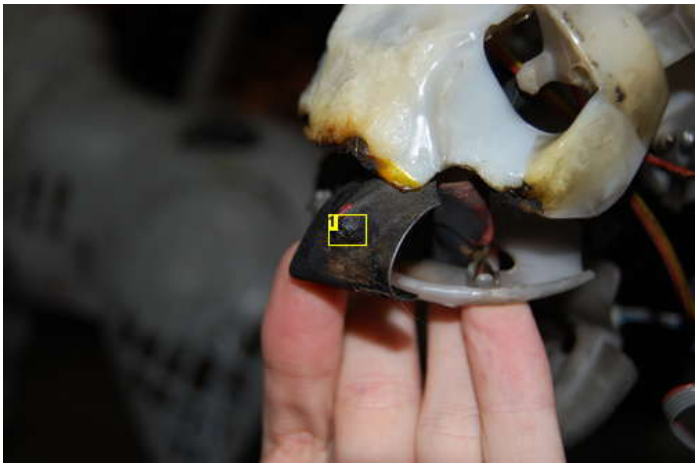


Image Notes
1. Fuel exit tube. Hard to see.

Step 20: Remote fuel trigger

The Remote fuel trigger uses a bowden cable to pull down on a sheet of hinged plexi glass, which pushes the spray nozzle on the fuel can. A bowden cable is the type of cable that is used in bike brakes. This allows it to be flexible in the middle, but still allows it to transfer force in relation to its end points.

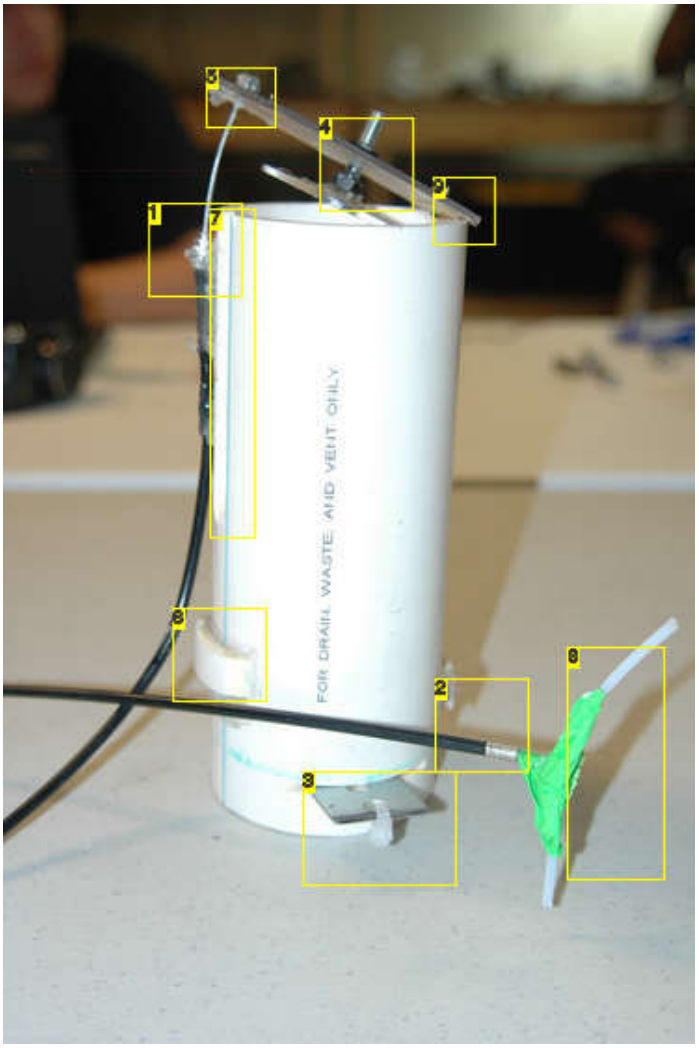


Image Notes

1. bowden cable
2. bowden cable
3. Plate to hold the fuel can inside the tube.
4. two bolts and a few nuts threaded into a piece of plexi glass. This let you change it slightly for variable sized fuel containers.
5. bowden cable Connecting to the chunk of plexy glass to pull/push down on the fuel.
6. this locks on ether side of the tube so that it can lock inside of the pony.
7. Slit in the side of the container so that the fuel tube can come out of the side.
8. This is just a handle to make the cable easier to pull.
9. hinge

Step 21: Follow up

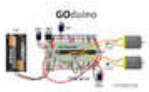
I know that I covered a lot and I am sure I missed something. If something was not clear feel free to ask a question and I will try to clear it up or update the instructable. I also have a short video from not long after maker fair that explains some of the operation of the pony.

Just remember to keep it safe and have fun.

Related Instructables

Ardu-pong! the Arduino based pong console by kyle brinkerhoff

arduino Internet PC robot controlled using iphone (video) by Avadhut.Deshmukh



GOduino - The Arduino Uno + Motor Driver clone (Photos) by techbitar

Arduino Robot for lowest cost by doncrush

OCCU(PI) Bot by randofo

Maker Faire NYC 2011 (Photos) by caitlinsdad

How to Make an Obstacles Avoiding Robot- Arduino Style by robotkid249

Tiny Programmer (Photos) by maxhirez

Tweet-a-watt - How to make a twittering power meter...

by **adafruit** on March 29, 2009

Author:adafruit Adafruit Industries

All-original DIY electronics kits - Adafruit Industries is a New York City based company that sells kits and parts for original, open source hardware electronics projects featured on www.adafruit.com as well as other cool open source tronix' that we think are interesting and well-made.

Intro: Tweet-a-watt - How to make a twittering power meter...

Tweet-a-watt - How to make a twittering power meter...

This project documents my adventures in learning how to wire up my home for wireless power monitoring. I live in a rented apartment so I don't have hacking-access to a meter or breaker panel. Since I'm still very interested in measuring my power usage on a long term basis, I built wireless outlet reporters. Building your own power monitor isn't too tough and can save money but I'm not a fan of sticking my fingers into 120V power. Instead, I'll used the existing Kill-a-watt power monitor, which works great and is available at my local hardware store.

My plan is to have each room connected to a 6-outlet power strip which powers all the devices in that room (each kill-a-watt can measure up to 15A, or about 1800W, which is plenty!). That way I can track room-by-room usage, for example "kitchen", "bedroom", "workbench", and "office".

Each wireless outlet/receiver can be built for ~$55 with a few easily-available electronic parts and light soldering, no microcontroller programming or high voltage engineering is necessary!

You can see my setup including graphs and reports at http://twitter.com/tweetawatt

If you'd like to build one for yourself

1. Buy a kit : get all the parts you need, there's a starter kit at the adafruit webshop
2. Make: turn each Kill-a-Watt into a wireless power level transmitter
3. Software: Download & run it on your computer to get data and save it to a file and/or publish it

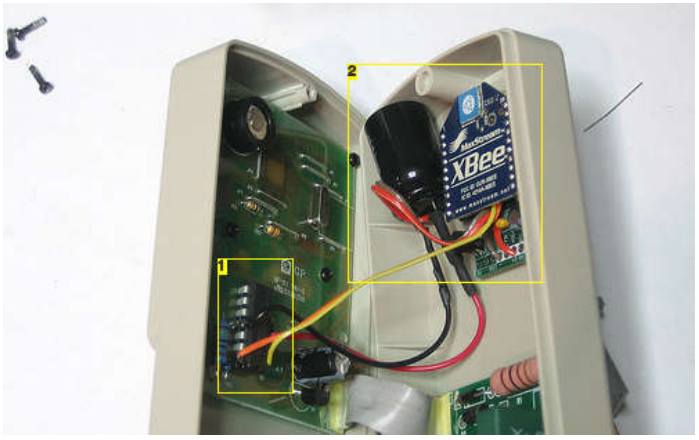
If you want to know how it was made, check out:

1. Listen: write simple software for my computer (or Arduino, etc) to listen for signal and compute the current power usage
2. Store: Create a database backend that will store the power usage for long-term analysis at http://wattcher.appspot.com
3. View: Graph and understand trends in power usage

Check out the latest readings at http://wattcher.appspot.com

Image Notes

1. Wireless transmission indicator
2. Uses a Kill-a-Watt, available at many hardware or electronic stores
3. Measures up to 15A (1800 Watts!)



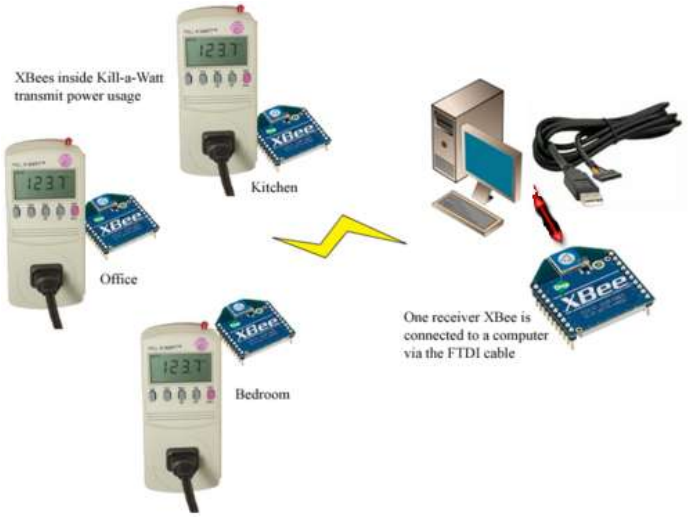


Image Notes
1. XBee listens to data coming from the Kill-a-Watt measurement sensors
2. XBee wireless module in an adapter and a few passive components to keep things running smooth

Step 1: Make it!

Before you start...

You should only attempt this project if you are comfortable and competent working with high voltage electricity, electronics and computers. Once the project is complete it is enclosed and there are no exposed high voltages. However, you must only work on the project when its not plugged in and **never ever attempt to test, measure, open, or probe the circuitboards while they are attached to a wall socket.** If something isn't working: stop, remove it from the wall power, then open it up and examine. Yes it takes a few more minutes but it's a lot safer!

Your safety is your own responsibility, including proper use of equipment and safety gear, and determining whether you have adequate skill and experience. Power tools, electricity, and other resources used for this projects are dangerous, unless used properly and with adequate precautions, including safety gear. Some illustrative photos do not depict safety precautions or equipment, in order to show the project steps more clearly. This projects is not intended for use by children.

Use of the instructions and suggestions is at your own risk. Adafruit Industries LLC, disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with applicable laws.

OK, if you agree we can move on!

Make a tweet-a-watt

To make the tweet-a-watt setup, we will have to go through a few steps

1. Prepare by making sure we have everything we need and know the skills necessary to build the project
2. Build the receiver setup by soldering up one of the adapter kits
3. Configure the XBee wireless modems
4. Build the transmitter setup by modifying a Kill-a-Watt to transmit via the XBee
5. Run the software, which will retrieve data and save it to a file, upload it to a database and/or twitter

Step 2: Prep

Tutorials

Learn how to solder with tons of tutorials!
Don't forget to learn how to use your multimeter too!

Tools

There are a few tools that are required for assembly. None of these tools are included. If you don't have them, now would be a good time to borrow or purchase them. They are very very handy whenever assembling/fixing/modifying electronic devices! I provide links to buy them, but of course, you should get them wherever is most convenient/inexpensive. Many of these parts are available in a place like Radio Shack or other (higher quality) DIY electronics stores.

I recommend a "basic" electronics tool set for this kit, which I describe here.

Soldering iron . One with temperature control and a stand is best. A conical or small 'screwdriver' tip is good, almost all irons come with one of these.

A low quality (ahem, $10 model from radioshack) iron may cause more problems than its worth!

Do not use a "ColdHeat" soldering iron, they are not suitable for delicate electronics work and can damage the kit (see here)

Solder . Rosin core, 60/40. Good solder is a good thing. Bad solder leads to bridging and cold solder joints which can be tough to find. Dont buy a tiny amount, you'll run out when you least expect it. A half pound spool is a minimum.

Multimeter/Oscilloscope . A meter is helpful to check voltages and continuity.

Flush/diagonal cutters . Essential for cutting leads close to the PCB.

Desoldering tool . If you are prone to incorrectly soldering parts.

'**Handy Hands' with Magnifying Glass** . Not absolutely necessary but will make things go much much faster.

Check out my recommended tools and where to buy.

Good light. More important than you think.

Step 3: Make the Receiver

Overview

We'll start with the receiver hardware, that's the thing that plugs into the computer and receives data from the wireless power plug. The receiver hardware does 'double duty', it also is used to update the XBee's modems' firmware (which, unfortunately, is necessary because they come from the factory with really old firmware) and configure the modems.

What you'll need

The receiver is essentially, an XBee, with a USB connection to allow a computer to talk to it the XBee.

Name **FTDI cable**
Description A USB-to-serial converter. Plugs in neatly into the Adafruit XBee adapter to allow a computer to talk to the XBee.
Datasheet TTL-232R 3.3V or 5.0V
Distributor Mouser
Qty 1

Name **Adafruit XBee Adapter kit**
Description I'll be using my own design for the XBee breakout/carrier board but you can use nearly any kind as long as you replicate any missing parts such as the3.3V supply and LEDs
You will have 2 adapter kits but you should only assemble one for this part! The other one needs different instructions so just hold off!
Datasheet Webpage
Distributor Adafruit
Qty 1

Name **XBee module**
Description We'll be using the XBee "series 1" point-to-multipoint 802.15.4 modules with a chip antenna part # XB24-ACI-001. They're inexpensive and work great. This project most likely won't work with any other version of the XBee, and certainly not any of the 'high power' Pro types!
Datasheet
Distributor Adafruit
Qty 1

Solder the adapter together!

This step is pretty easy, just go over to the XBee adapter webpage and solder it together according to the instructions!

Remember: You will have 2 adapter kits but you should only solder one of them at this point! The other one needs different instructions so just hold off!

Connect to the XBee

Now its time to connect to the XBees

Find your FTDI cable - use either 3.3V or 5V. These cables have a USB to serial converter chip molded into them and are supported by every OS. Thus configuring or upgrading or connecting is really trivial. Simply plug the cable into the end of the module so that the black wire lines up with GND. There is a white outline showing where the cable connects.

You'll need to figure out which serial port (COM) you are using. Plug in the FTDI cable, USB adapter, Arduino, etc. Under Windows, check the device manager, look for "USB Serial Port"

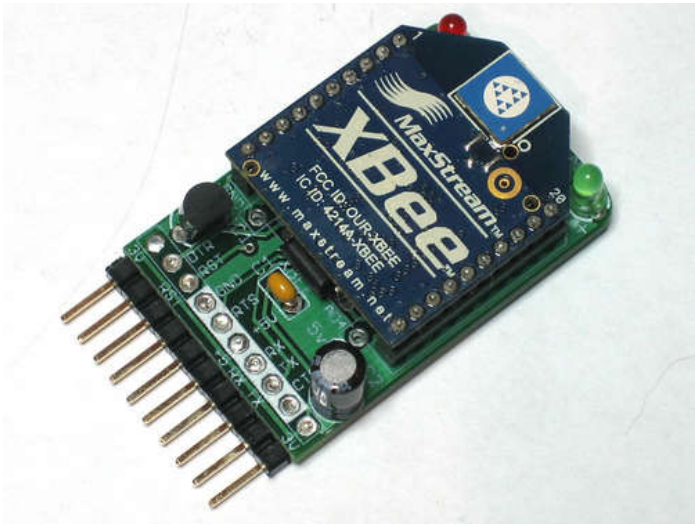
Digi/Maxstream wrote a little program to help configure XBees, its also the only way I know of to upgrade them to the latest firmware. Unfortunately it only runs on Windows. Download X-CTU from Digi and install it on your computer

After installing and starting the program, select the COM port (COM4 here) and baud rate (9600 is default). No flow control, 8N1. Make sure the connection box looks just like the image (other than the com port which may be different)

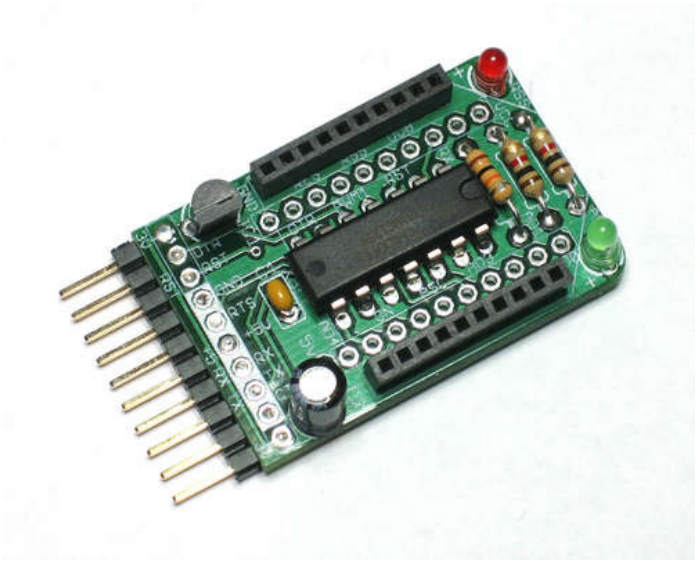
To verify, click **Test** / **Query**

Hopefully the test will succeed. If you are having problems: check that the XBee is powered, the green LED on the adapter board should be blinking, the right COM port & baud rate is selected, etc.

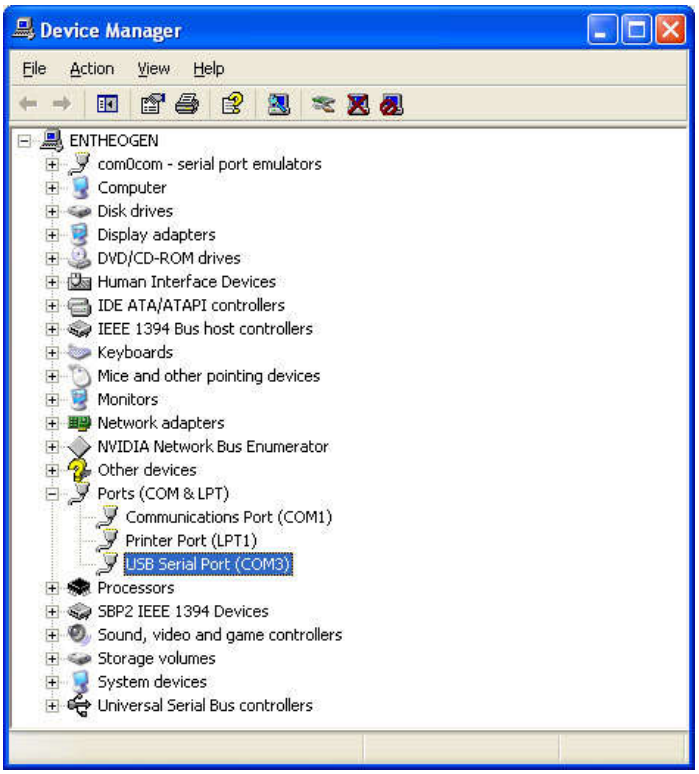
Now unplug the adapter from the FTDI cable, carefully replace the first XBee with the other one and make sure that one is talking fine too. Once you know both XBees are working with the adapter, its time to upgrade and configure them, the next step!

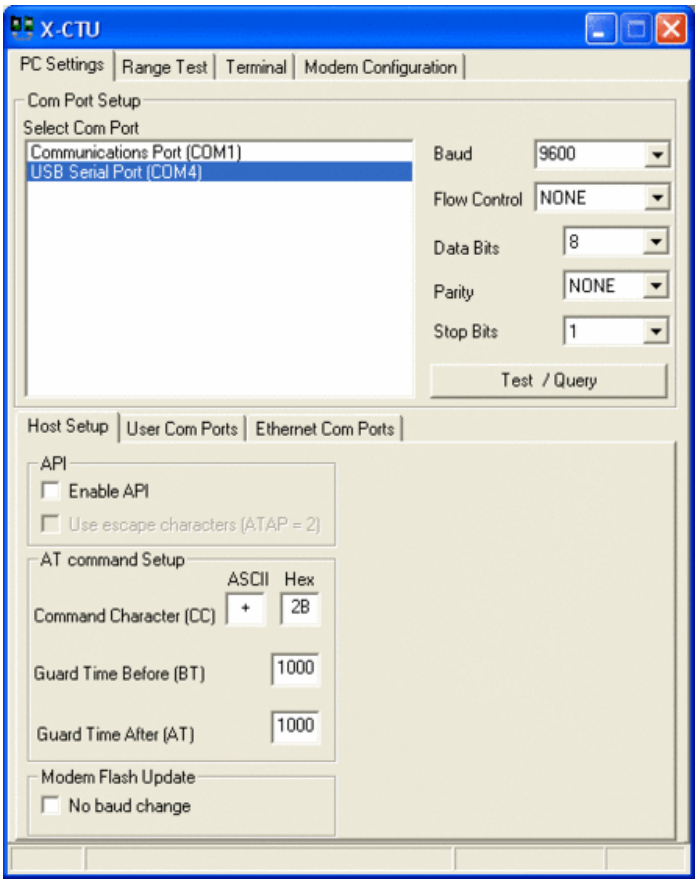

MaxStream
XBee
FCC ID: OUR-XBEE
IC ID: 4214A-XBEE
www.maxstream.net

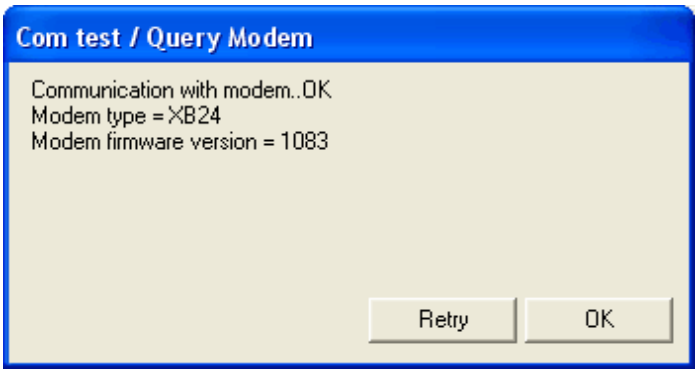

Digi
XBee
FCC ID: OUR-XBEE
IC ID: 4214A-XBEE




XBee
MaxStream
FCC ID: OUR-XBEE
IC ID: 4214A-XBEE
www.maxstream.net







Step 4: Configure

Overview

OK so far you have assembled **one** of the XBee adapter boards and connected it to your computer using the FTDI cable. (The other adapter is for later so don't do anything with it yet!) The XBees respond to the X-CTU software and are blinking just fine. Next we will update the firmware

Upgrading the firmware

There's a good chance your XBees are not running the latest firmware & there's a lot of features added, some of which we need to get this project running. So next up is upgrading!

Go to the Modem Configuration tab. This is where the modem is configured and updated

Click **Download new versions** ... and select to download the latest firmwares from the Web

Once you have downloaded the newest firmware, its time to upgrade!

Click on **Modem Parameters** -> "**Read** " to read in the current version and settings

Now you will know for sure what function set, version and settings are stored in the modem

Select from the **Version** dropdown the latest version available

Check the **Always update firmware** checkbox

And click **Write** to initialize and program the new firmware in!

That's it, now you have the most recent firmware for your modem. You should now uncheck the **Always update firmware** checkbox. If you have problems, like for example timing out or not being able to communicate, make sure the RTS pin is wired up correctly as this pin is necessary for upgrading. FTDI cables are already set up for this so you shouldn't have a problem

Rinse & Repeat

Upgrade the firmware on both of the XBees so they are both up to date

At this point it might be wise to label the two XBees in a way that lets you tell them apart. You can use a sharpie, a sticker or similar to indicate which one is the receiver and which is the transmitter

Configure the transmitter XBee

Both XBee's need to be upgraded with the latest firmware but only the transmitter (which is going to be put inside a Kill-a-Watt) needs to be configured. The configure process tells the XBee what pins we want to read the sensor data off of. It also tells the XBee how often to send us data, and how much.

Plug the **transmitter** XBee into the USB connection (put the receiver XBee away) and start up X-CTU or a Terminal program. Connect at 9600 baud, 8N1 parity.Then configure each one as follows:

1. Set the **MY** address (the identifier for the XBee) to **1** (increment this for each transmitter so you can tell them apart, we'll assume you only have one for now)
2. Set the Sleep Mode **SM** to 4 (Cyclic sleep)
3. Set the Sleep Time **ST** to 3 (3 milliseconds after wakeup to go back to sleep)
4. Set the Sleep Period **SP** to **C8** (0xC8 hexadecimal = 200 x 10 milliseconds = 2 seconds between transmits)
5. Set ADC 4 **D4** to **2** (analog/digital sensor enable pin **AD4**)
6. Set ADC 0 **D0** to **2** (analog/digital sensor enable pin **AD0**)
7. Set Samples to TX **IT** to 13 (0x13 = 19 A/D samples per packet)
8. Set Sample Rate **IR** to 1 (1 ms between A/D samples)

if you think there will be more XBee's in the area that could conflict with your setup you may also want to

1. Set the PAN **ID** to a 4-digit hex number (its **3332** by default)

You can do this with X-CTU or with a terminal program such as hyperterm, minicom, zterm, etc. with the command string
ATMY=1,SM=4,ST=3,SP=C8,D4=2,D0=2,IT=13,IR=1
You'll need to start by getting the modem's attention by waiting 10 seconds, then typing in +++ quickly, then pausing for another 5 seconds. Then use **AT** to make sure its paying **AT** tention to your commands

Basically what this means is that we'll have all the XBees on a single PAN network, each XBee will have a unique identifier, they'll stay in sleep mode most of the time, then wake up every 2 seconds to take 19 samples from ADC 0 and 4, 1ms apart. If you're having difficulty, make sure you upgraded the firmware!

Make sure to WRITE the configuration to the XBee's permanent storage once you've done it. If you're using X-CTU click the "Write" button in the top left. If you're using a terminal, use the command **ATWR** !

Note that once the XBee is told to go into sleep mode, you'll have to reset it to talk to it because otherwise it will not respond and X-CTU will complain. You can simply unplug the adapter from the FTDI cable to reset or touch a wire between the RST and GND pins on the bottom edge of the adapter.

Now that the transmitters are all setup with unique **MY** number ID's, make sure that while they are powered from USB the green LED blinks once every 2 seconds (indicating wakeup and data transmit)

Configure the receiver XBee

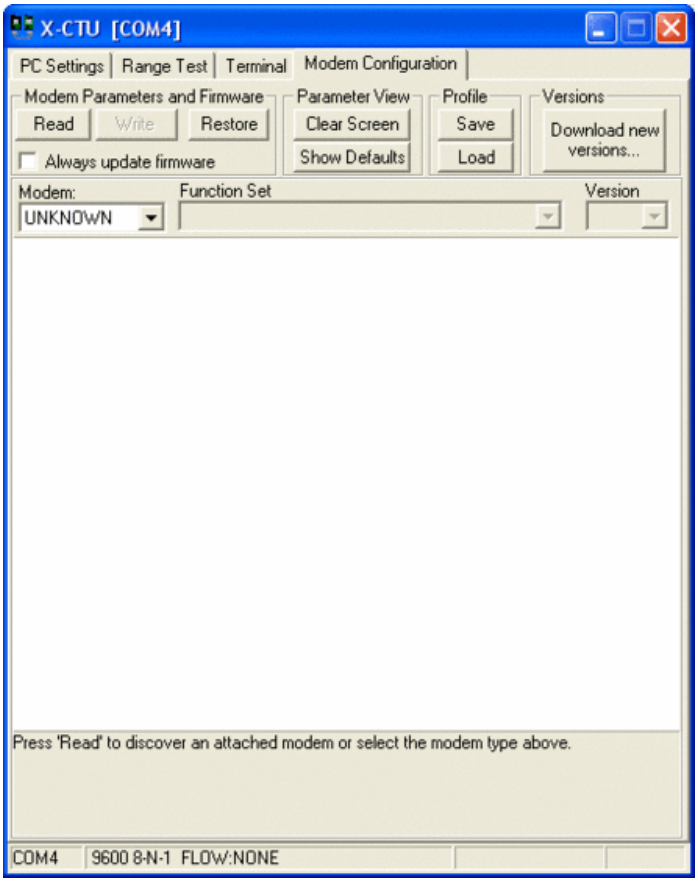
Plug the receiver XBee into the USB connection (put the receiver XBee away) and start up X-CTU . If you set the PAN ID in the previous step, you will have to do the same here

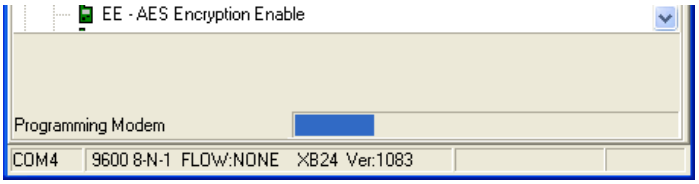
- Set the PAN **ID** to the same hex number as above

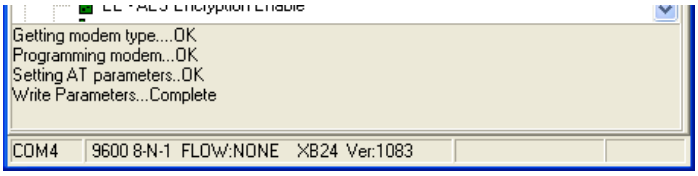
If you didn't change the PAN above, then there's nothing for you to do here, just skip this step

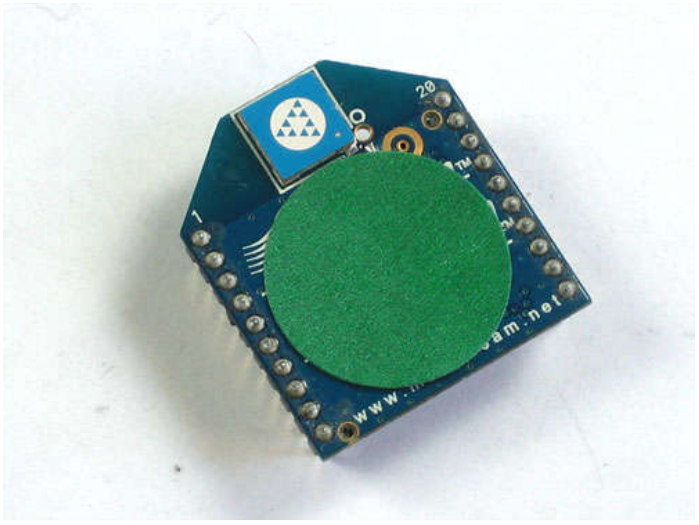
Next!

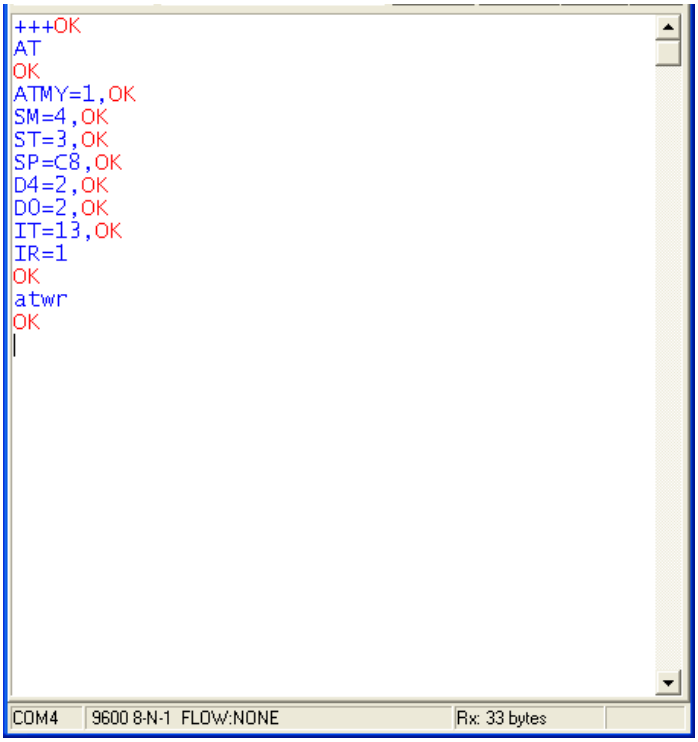
Now that the XBees are configured and ready, its time to go to the next step where we make the Kill-a-Watt hardware

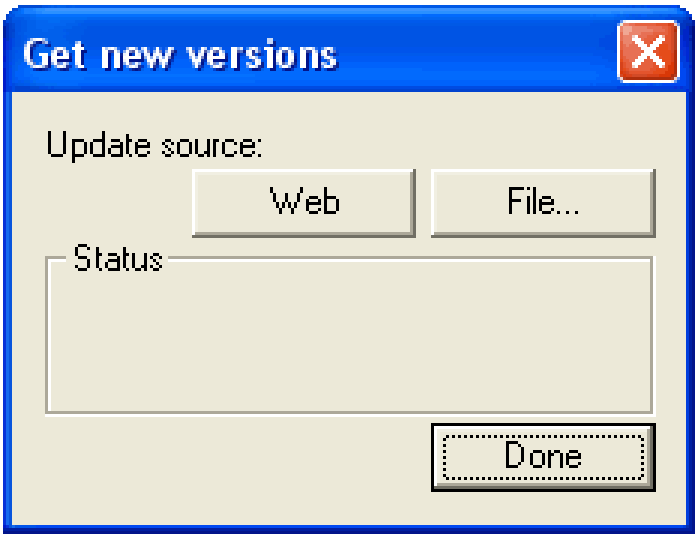

X-CTU [COM4]
PC Settings
Range Test
Terminal
Modem Configuration
Modem Parameters and Firmware
Read
Write
Restore
Always update firmware
Parameter View
Clear Screen
Show Defaults
Profile
Save
Load
Versions
Download new versions...
Modem:
UNKNOWN
Function Set
Version
Press 'Read' to discover an attached modem or select the modem type above.
COM4
9600 8-N-1 FLOW:NONE

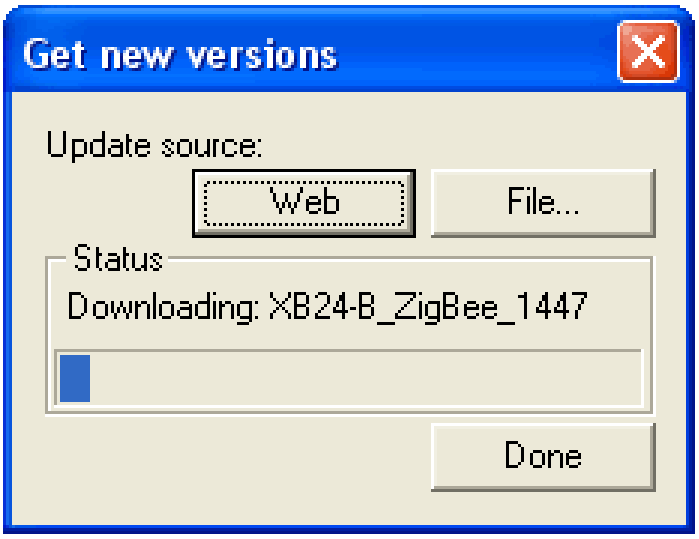

EE - AES Encryption Enable
Programming Modem
COM4
9600 8-N-1 FLOW:NONE
XB24 Ver:1083

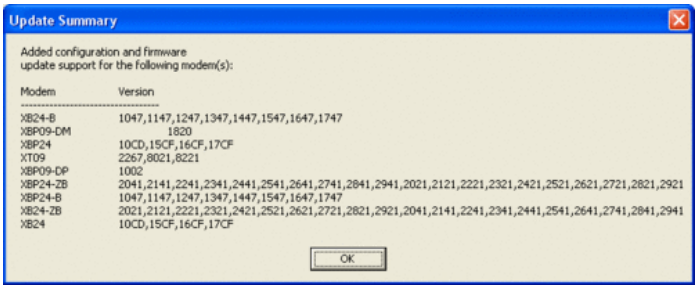

Getting modem type....OK
Programming modem...OK
Setting AT parameters..OK
Write Parameters...Complete
COM4
9600 8-N-1 FLOW:NONE
XB24 Ver:1083

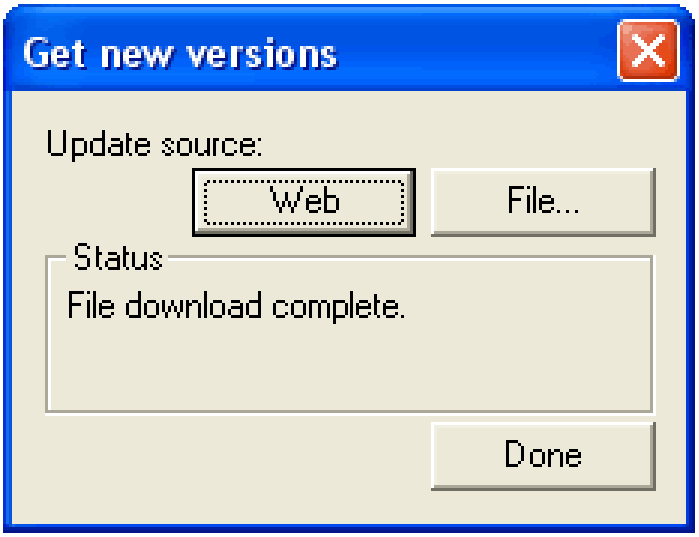


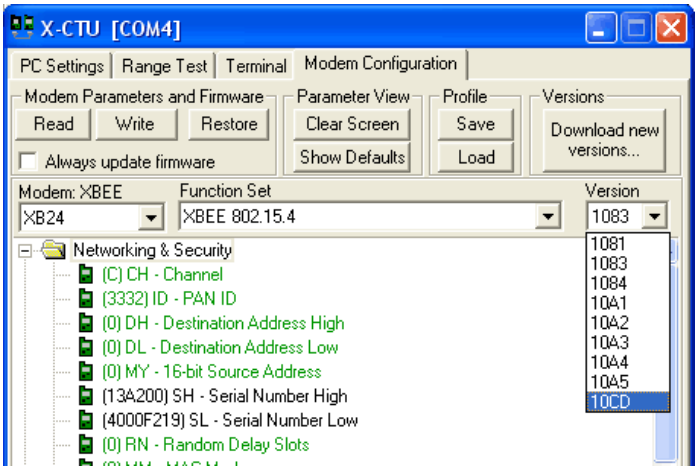

+++OK
AT
OK
ATMY=1,OK
SM=4,OK
ST=3,OK
SP=C8,OK
D4=2,OK
DO=2,OK
IT=13,OK
IR=1
OK
atwr
OK
COM4
9600 8-N-1 FLOW:NONE
Rx: 33 bytes

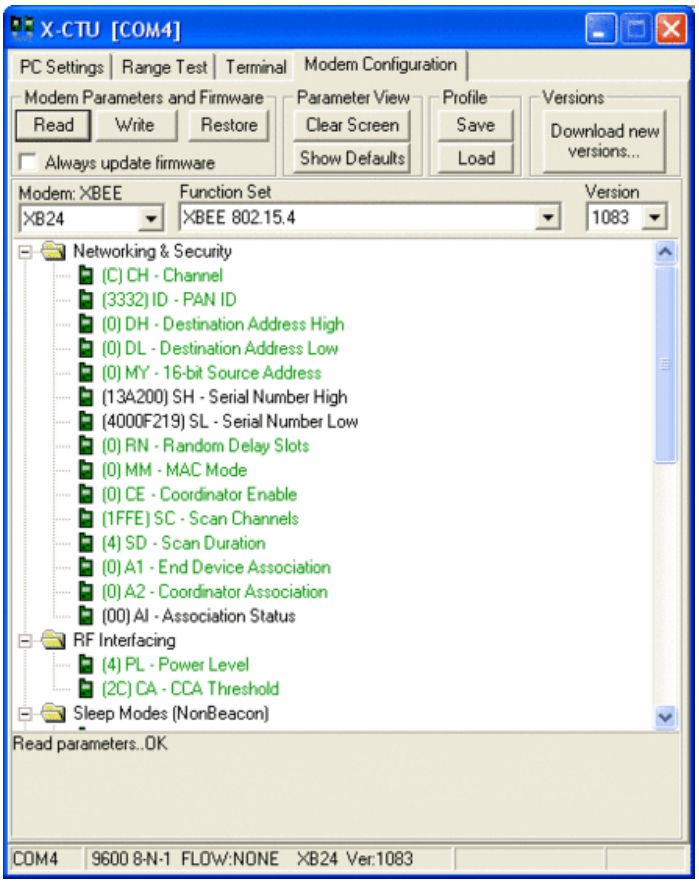

Get new versions
Update source:
Web
File...
Status
Done

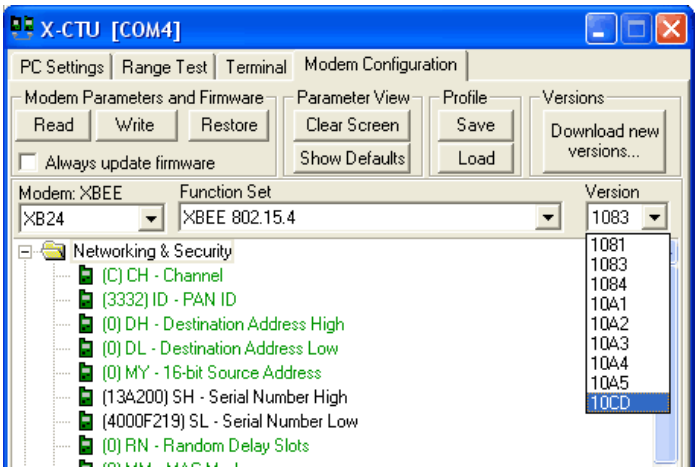

Get new versions
Update source:
Web
File...
Status
Downloading: XB24-B_ZigBee_1447
Done

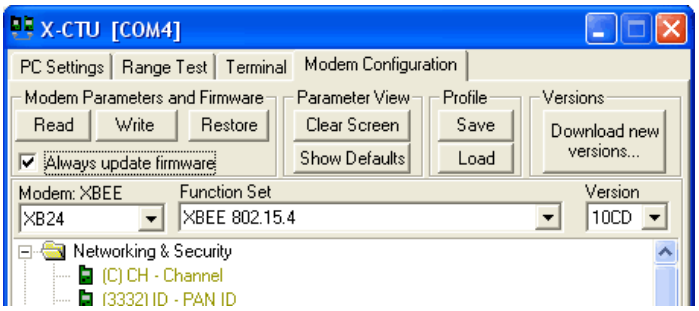

Update Summary
Added configuration and firmware
update support for the following modem(s):
Modem Version
XB24-B 1047,1147,1247,1347,1447,1547,1647,1747
XBP09-DM 1820
XBP24 10CD,15CF,16CF,17CF
XT09 2267,8021,8221
XBP09-DP 1002
XBP24-ZB 2041,2141,2241,2341,2441,2541,2641,2741,2841,2941,2021,2121,2221,2321,2421,2521,2621,2721,2821,2921
XBP24-B 1047,1147,1247,1347,1447,1547,1647,1747
XB24-ZB 2021,2121,2221,2321,2421,2521,2621,2721,2821,2921,2041,2141,2241,2341,2441,2541,2641,2741,2841,2941
XB24 10CD,15CF,16CF,17CF
OK

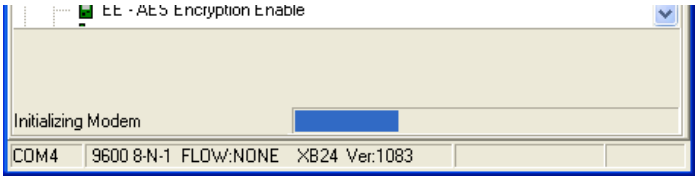

Get new versions
Update source:
Web
File...
Status
File download complete.
Done


X-CTU [COM4]
PC Settings
Range Test
Terminal
Modem Configuration
Modem Parameters and Firmware
Read
Write
Restore
Always update firmware
Parameter View
Clear Screen
Show Defaults
Profile
Save
Load
Versions
Download new versions...
Modem: XBEE
XB24
Function Set
XBEE 802.15.4
Version
1083
Networking & Security
(C) CH - Channel
(3332) ID - PAN ID
(0) DH - Destination Address High
(0) DL - Destination Address Low
(0) MY - 16-bit Source Address
(13A200) SH - Serial Number High
(4000F219) SL - Serial Number Low
(0) RN - Random Delay Slots
1081
1083
1084
10A1
10A2
10A3
10A4
10A5
10CD









Step 5: Solder the Transmitter - parts list

Before you start...

You should only attempt this project if you are comfortable and competent working with high voltage electricity, electronics and computers. Once the project is complete it is enclosed and there are no exposed high voltages. However, you must only work on the project when its not plugged in and **never ever attempt to test, measure, open, or probe the circuitboards while they are attached to a wall socket.** If something isn't working: stop, remove it from the wall power, then open it up and examine. Yes it takes a few more minutes but it's a lot safer!

Your safety is your own responsibility, including proper use of equipment and safety gear, and determining whether you have adequate skill and experience. Power tools, electricity, and other resources used for this projects are dangerous, unless used properly and with adequate precautions, including safety gear. Some illustrative photos do not depict safety precautions or equipment, in order to show the project steps more clearly. This projects is not intended for use by children.

Use of the instructions and suggestions is at your own risk. Adafruit Industries LLC, disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with applicable laws.

OK, if you agree we can move on!

Transmitter partslist

For each outlet you want to monitor, you'll need:

Name: **Kill-a-Watt**
Description: "Off the shelf" model P4400 power monitor
Datasheet: P3 Kill-a-watt
Distributor: Lots! Also check hardware/electronics stores
Qty: 1

Name: **Adafruit XBee Adapter**
Description: I'll be using my own design for the XBee breakout/carrier board but you can use nearly any kind as long as you replicate any missing parts such as the3.3V

supply and LEDs
Datasheet: Webpage
Distributor: Adafruit
Qty: 1

Name: **XBee module**
Description: We'll be using the XBee "series 1" point-to-multipoint 802.15.4 modules with a chip antenna part # XB24-ACI-001. They're inexpensive and work great. This project most likely won't work with any other version of the XBee, and certainly not any of the 'high power' Pro types!
Distributor: Adafruit
Qty: 1

Name: **D3**
Description: 1N4001 diode. Any power diode should work fine. Heck, even a 1n4148 or 1n914 should be OK. But 1N4001 is suggested and is in the kit.
Datasheet: Generic 1N4001
Distributor: Digikey Mouser
Qty: 1

Name: **D2**
Description: Large diffused LED, for easy viewing. The kit comes with green.
Qty: 1

Name: **C3**
Description: 220uF, 4V or higher (photo shows 100uF)
Datasheet: Generic
Distributor: Digikey Mouser
Qty: 1

Name: **C4**
Description: 10,000uF capacitor (wow!) / 6.3V (photo shows a mere 2200uF) Try to get 16mm diameter, 25mm long
Datasheet: Generic
Distributor: Digikey [Mouser]
Qty: 1

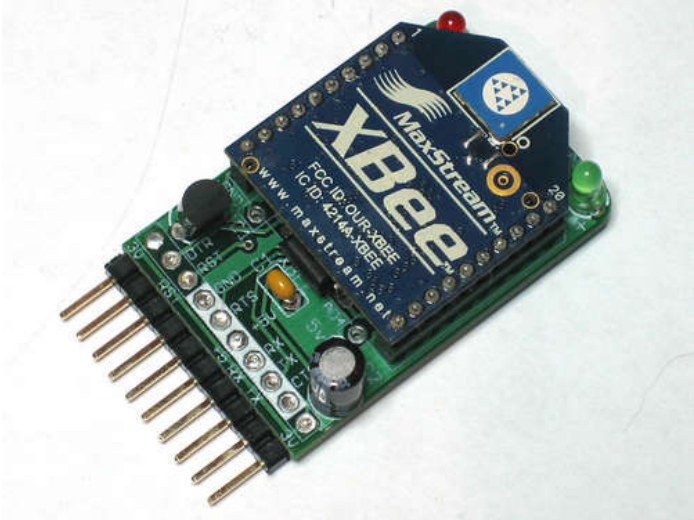
Name: **R4 R6**
Description: 10K 1/4W 1% resistor (brown black black red gold) or 10K 1/4W 5% resistor (brown black orange gold). 1% is preferred but 5% is OK
Datasheet: Generic
Distributor: Mouser Digikey
Qty: 2

Name: **R3 R5**
Description: 4.7K 1/4W 1% resistor (yellow violet black brown gold) or 4.7K 1/4W 5% resistor (yellow violet red gold). 1% is preferred but 5% is OK.
Datasheet: Generic
Distributor: Mouser Digikey
Qty: 2

Name: **Ribbon cable**
Description: Ribbon cable, or other flexible wire, at least 6 conductors, about 6" long
Datasheet: Generic Ribbon
Distributor: Digikey
Qty: 6"

Name: **Heat shrink**
Description: Heat shrink! A couple inches of 1/8" and 1/16" each
Datasheet: Generic

It will run you about $50-$60 for each outlet




Digi
XBee
FCC ID: OUR-X8E
IC ID: 4214A-XBEE


10v 100uF 10v


16v 2200uF 16v

Step 6: Transmitter Schematic

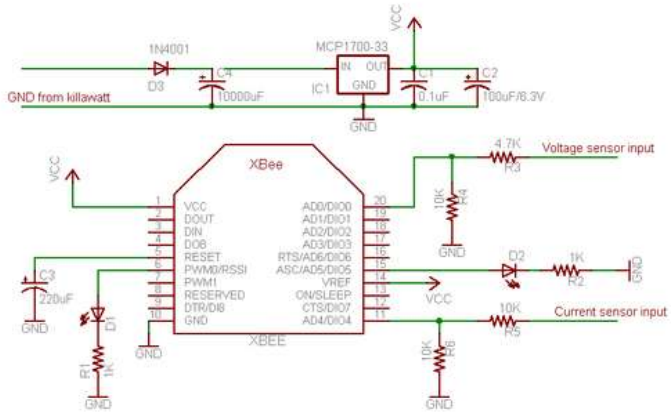
The XBee radio does all of the hard work, it listens on two analog input ports (AD0 and AD4) for voltage and current data. Then it transmits that information wirelessly to the host computer receiver XBee. There are a few we have to engineer around to make it Work:

1. We want to run the XBee off the Kill-a-Watt's internal power supply. However its current limited and wont provide 50mA in a burst when the XBee transmits. We solve this by adding a simple 'rechargeable battery' in the form of a really large capacitor **C4** .

2. The Kill-a-Watt runs at 5V but XBees can only run at 3.3V so we have a voltage regulator **IC1** and two capacitors two stabilize the 3.3V supply, **C1** and **C2** .

3. The XBee will transmit every few seconds, even while the capacitor is charging. This means that it will keep draining the capacitor, resetting, and trying again, basically freaking out while the power supply is still building. We prevent this by adding another fairly big capacitor **C3** on the reset line. This slows down the XBee, delaying the startup by a few seconds & keeps the XBee from starting up till we have solid power.

4. The XBee analog sensors run at 3.3V but the Kill-a-Watt sensors run at 5V. We use simple voltage dividers **R3** /**R4** and **R5** /**R6** to reduce the analog signal down to a reasonable level



Step 7: Assemble and create the transmitter - 1

Open up your kit and get out the parts for the transmitter. Remember that we'll be using most of but not all of an XBee adapter kit. The two small LEDs, the 74HC125N chip, a 10K and 1K resistor are not used and you should put them aside for a future project so you don't accidentally use them here.

Check to make sure you've got everything you need. The only thing not shown here is the XBee radio and Kill-a-Watt.

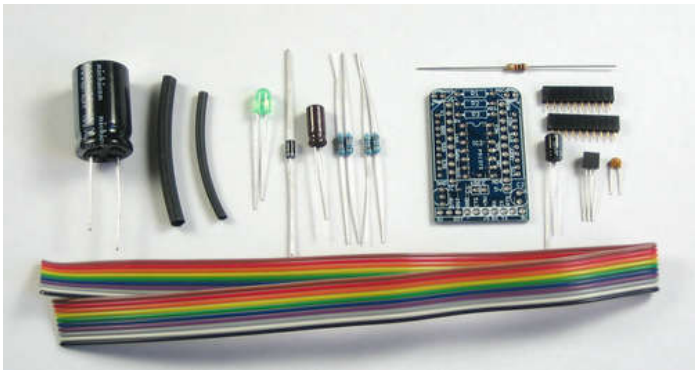
Place the PCB of adapter kit and get ready to solder by heating up your soldering iron, and preparing your hand tools

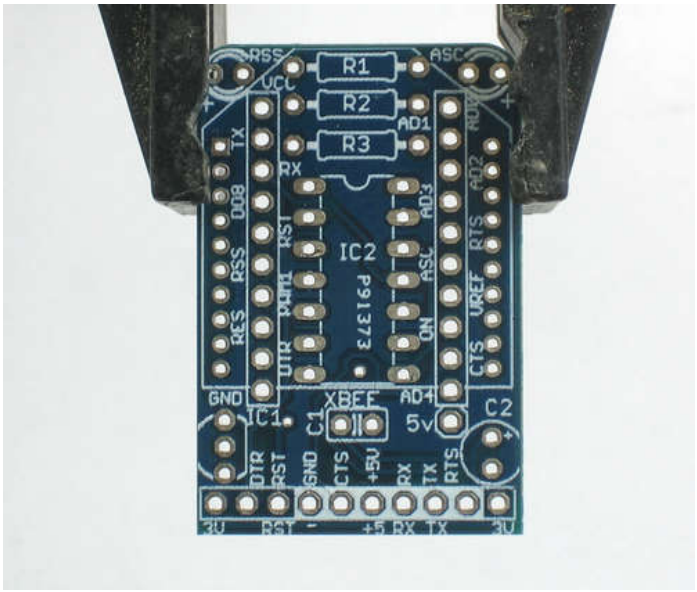
We'll start by soldering in the 3.3V regulator, which is identical to the standard XBee Adapter kit you made in the receiver instructions. Don't forget to check the polarity of **C2** and that **IC1** is in the right way. Then solder and clip the three components.

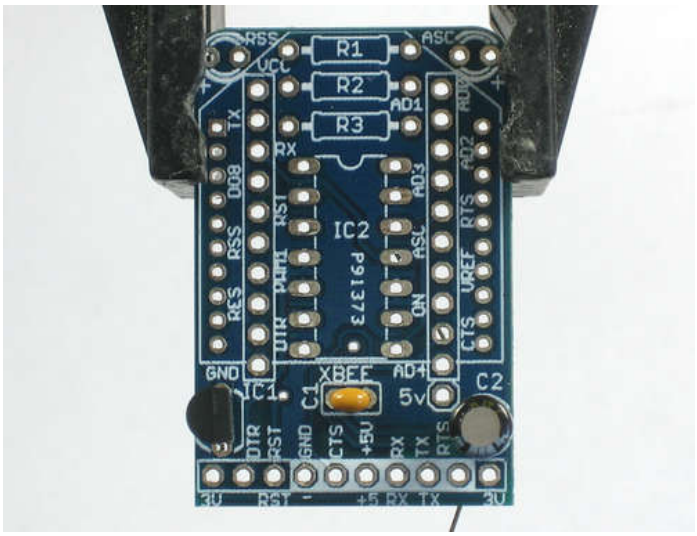
Now we will veer from the standard XBee adapter instructions and add a much larger LED on the ASC line so that we can easily see it blinking when its in the Kill-a-Watt. Make sure to watch for the LED polarity, because a backwards LED will make debugging very difficult. The longer lead goes in the + marked solder hole.

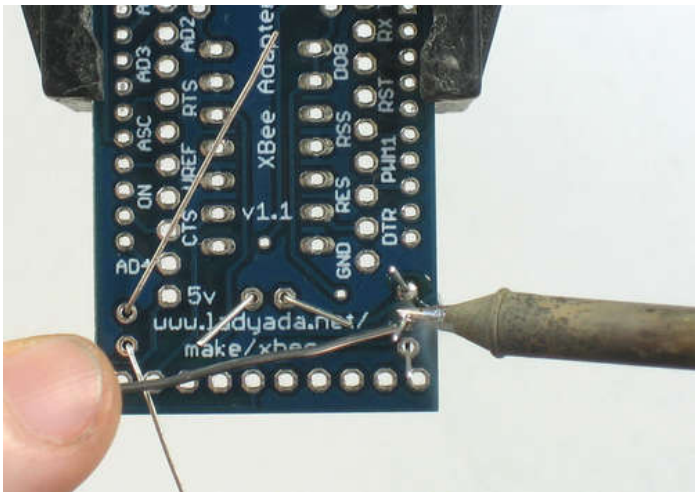
Give the LED about half an inch of space beyond the end of the PCB as shown. Also solder in the matching 1K resistor **R2**

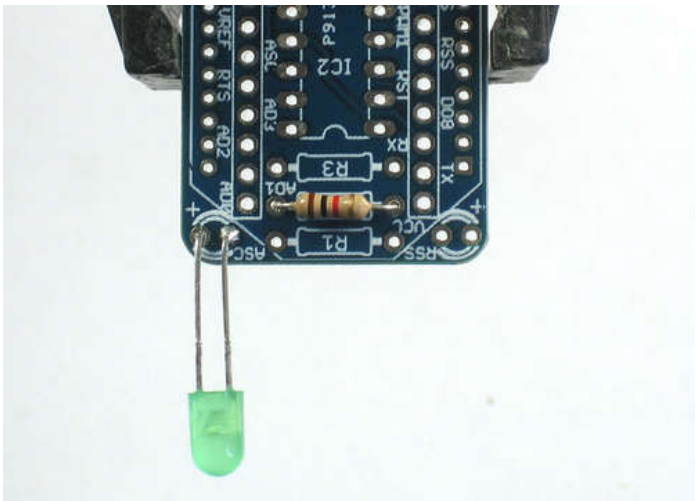
Solder in the two 2mm 10pin female headers in the adapter kit. Be careful with the solder so that you don't accidentally fill the female header. Use a sparing amount to make sure there's a connection but its not overflowing

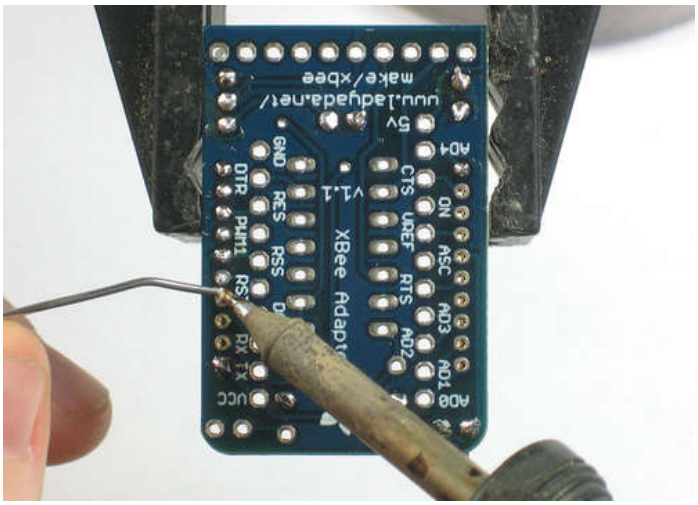


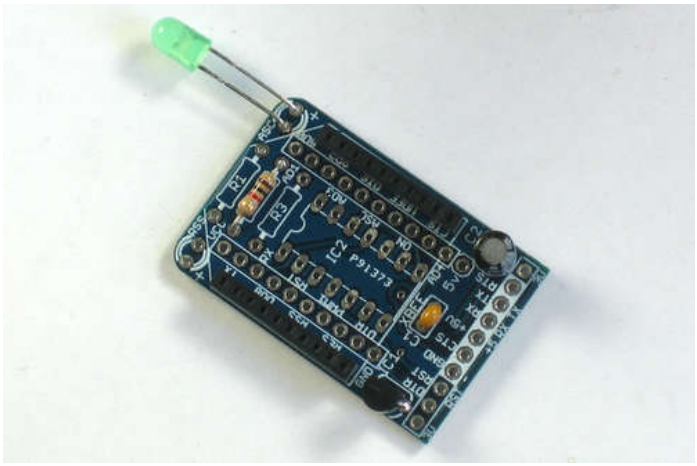












Step 8: Assemble and create the transmitter - 2

Now its time to prepare the wires we need for the next few stops. Use your diagonal cutters to notch off the brown, red, orange and yellow wires from the end of the rainbow ribbon cable in the kit.

Then tear off the four wires from the rest of the cable.

Do the same for the black and white wires and the single green wire. Then cut the green wire so its only about 1.5" long. You should now have 3 strips of wire, one 6" with 4 conductors, one 6" with 2 conductors and one 1.5" with 1 conductor

Use wirestrippers to strip the ends of the green wire, 1/4" from the ends

Then tin the green wire by heating the ends of the wire and applying a little solder to bind together the stranded wire.

Use the green wire to create a jumper between the **VREF** pin, 7th from the top on the right and the **VCC** pin on the top left.

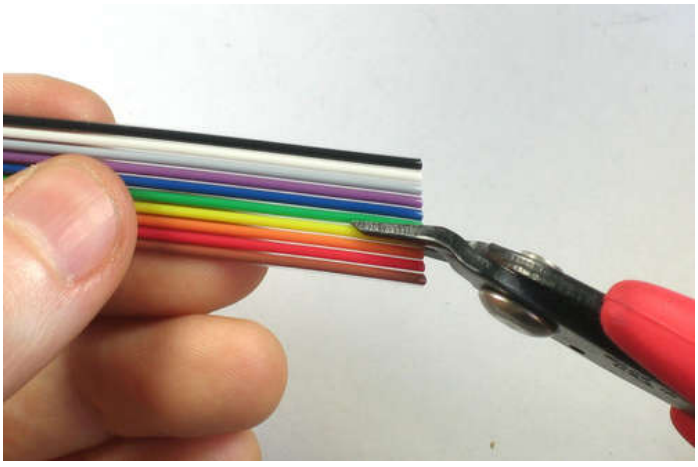
Double check to make sure you get this right! Then solder it in place. This will set the reference point of the analog converter to 3.3V

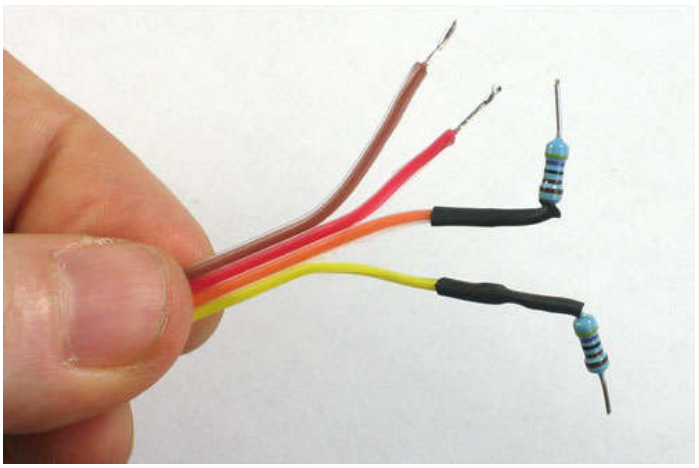
Go back to the 4-piece ribbon cable. Split the ends with the diagonal cutter, then strip and tin all 8 ends.

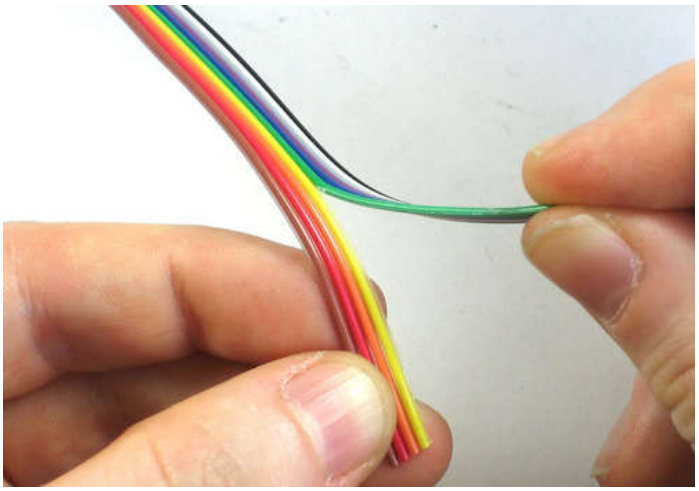
Put a 4.7K resistor in a vise or holder, then clip one end off and tin it just like the wires.

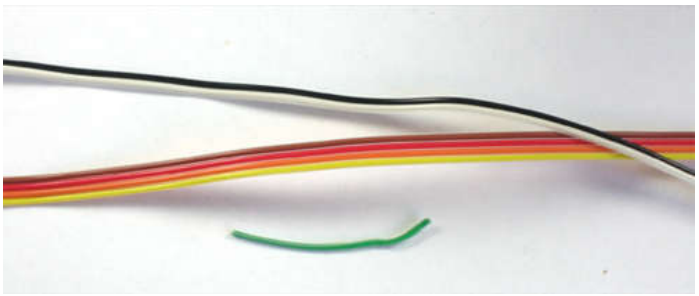
Cut a 1/2" piece of 1/16" heat shrink and slip it onto the yellow wire, making sure there's clearance between the heatshrink and the end of the wire. Then solder the yellow wire to the 4.7k resistor.

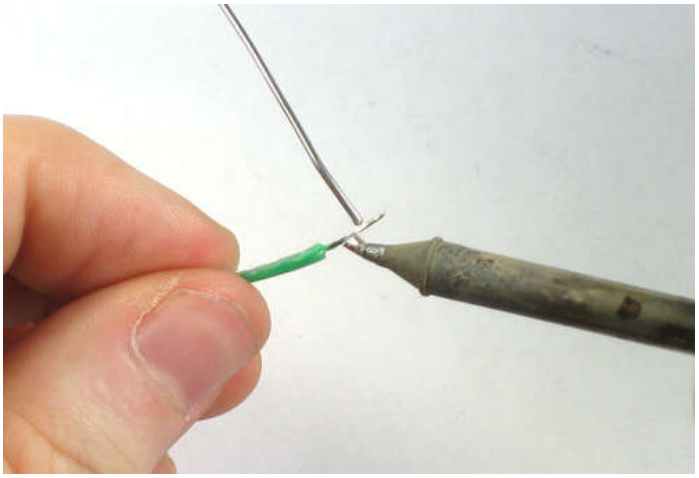
Do the same for the orange wire and the other 4.7K resistor. Use a heat source (a heat gun or hair drier is perfect) to shrink the heatshrink over the soldered wire/resistor joint.Then bend the resistor 90degrees and clip the other end of the 4.7k resistors

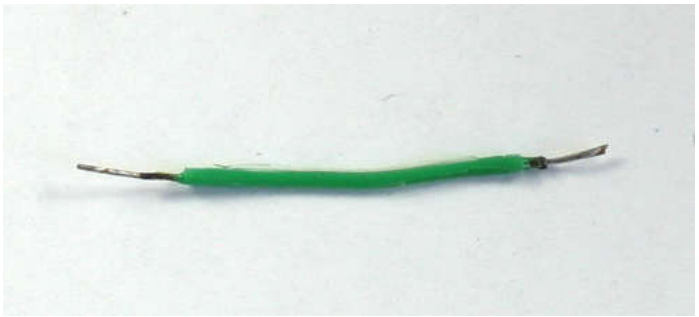


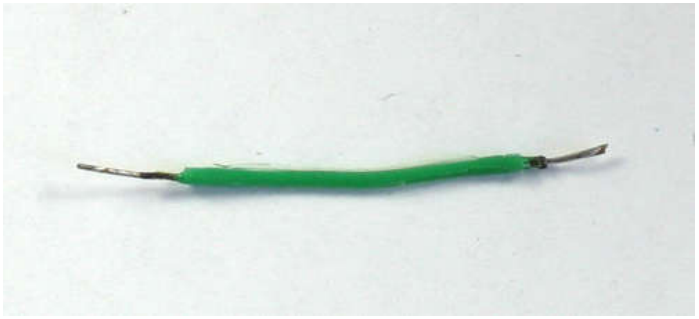


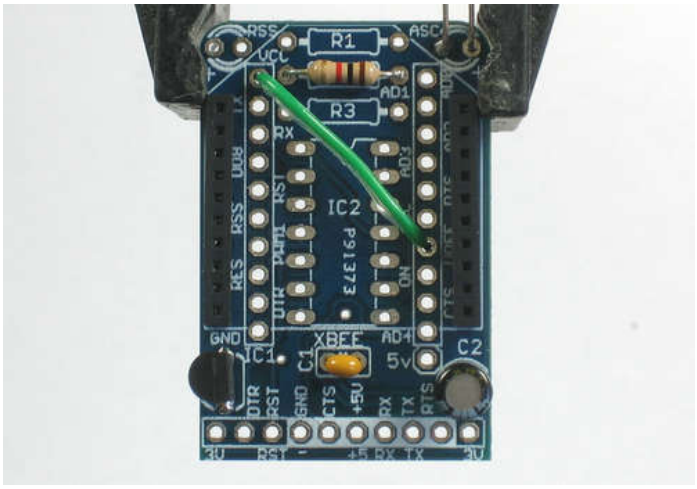












Step 9: Assemble and create the transmitter - 3

Now we will build the voltage divider. Take the two 10K resistors and connect them as shown. One goes from **AD0** and one from **AD4** . Both then connect to ground. Conveniently, the chip we are not using had grounded pins so we can 'reuse' those pins.

Now comes the tricky part. We want to connect the other end of the 4.7K resistor to the AD0 pin but the 10K resistor is already there. Use your soldering iron to melt a blob of solder onto the top of the 10K resistor and then piggyback the 4.7K resistor by soldering to the top of the 10K resistor.

Solder the orange wire to the **AD0** pin, the yellow to the **AD4**

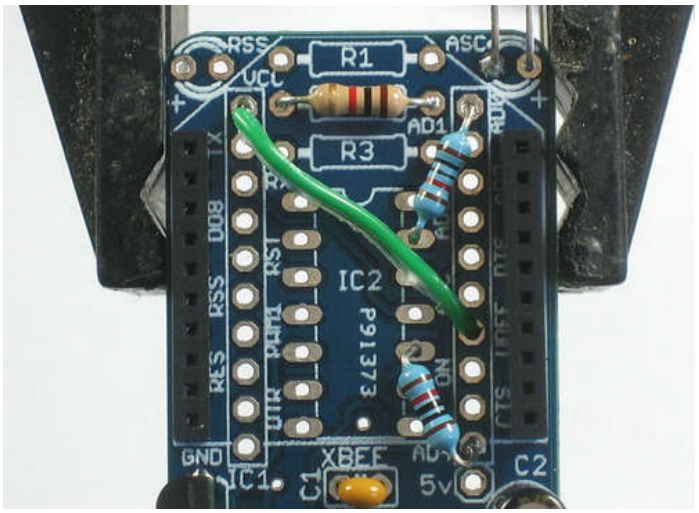
The other two wires are for carrying power. The red wire should be soldered to the +5V pin on the bottom of the adapter PCB. The brown wire to the **GND** pin.

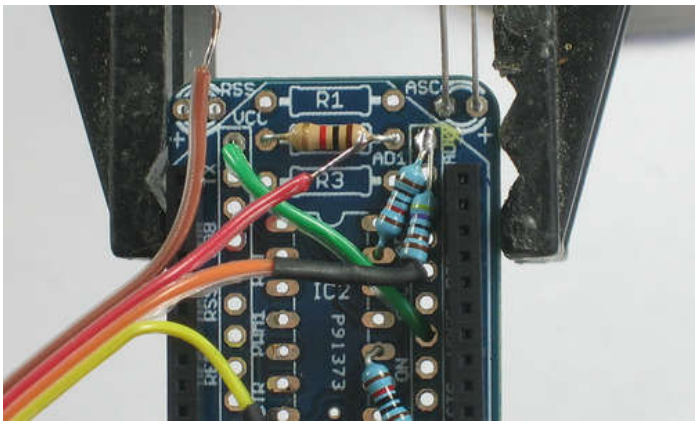
We're nearly done with the adapter soldering. Lastly is the 220uF reset capacitor. We'll connect this to the RST pin, 5th from the top on the left. Make sure the long lead is connected to the RST pin and the shorter lead goes to the 4th pin of where the chip would go. Check the photo on the left to make sure you've got it in right.

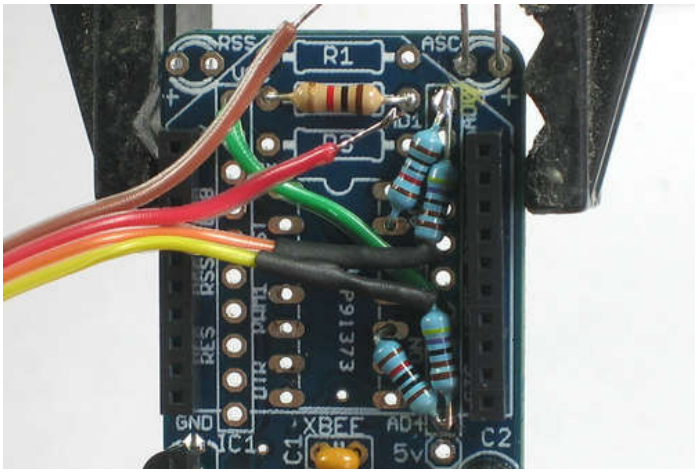
The capacitor wont fit underneath the XBee module so give it some lead length so that the cylindrical bulk is next to the 3.3V regulator.

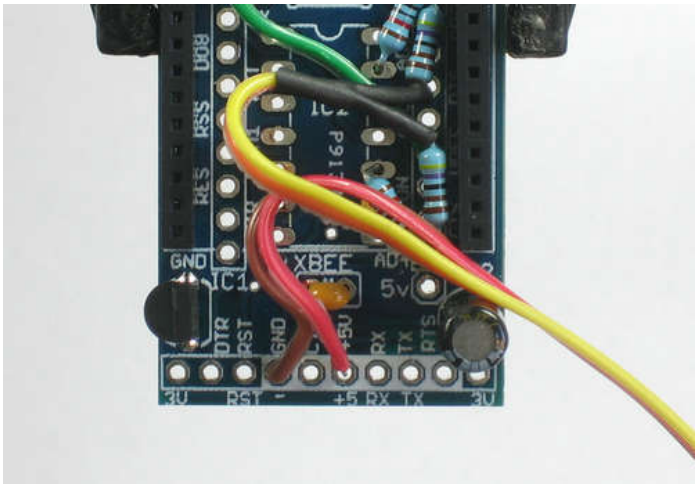
For reference, the images below show what the back should look like.

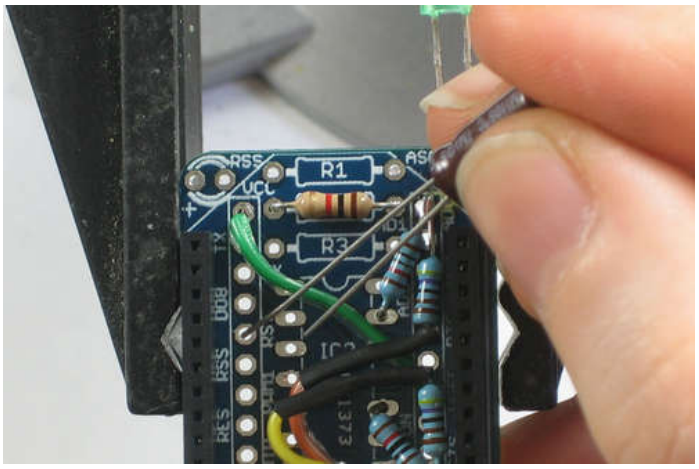
... and what it should look like with the XBee modem installed. Make sure the pins on the XBee line up with the header.

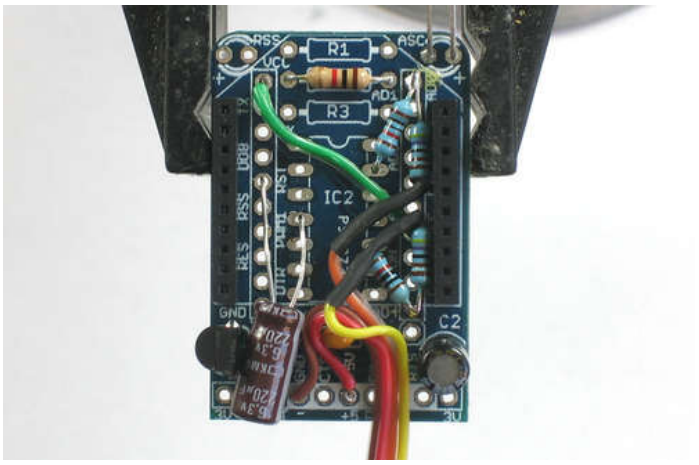

RSS
R1
ASC
VCC
AD1
R3
IC2
GND
XBEE
IC1
C1
5v
C2

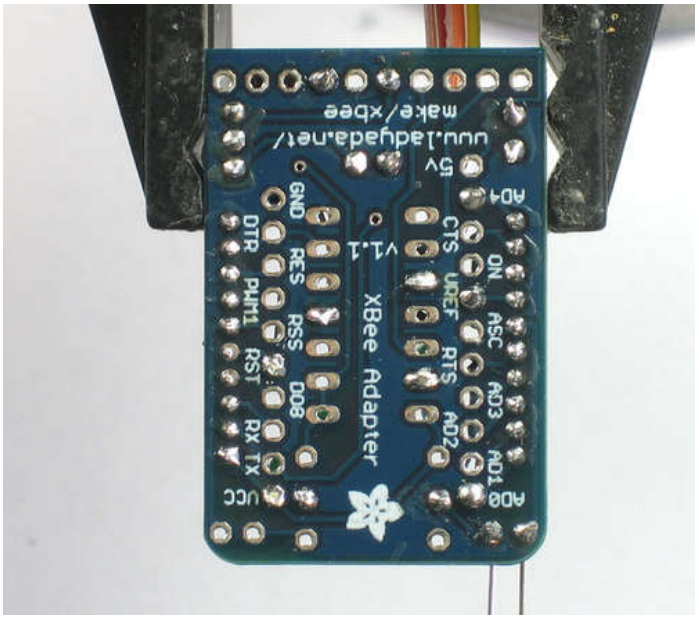

RSS
R1
ASC
VCC
AD1
R3
IC2


RSS
R1
ASC
R3
GND
XBEE
IC1
C1
5v
C2


GND
XBEE
IC1
5v
DTR
RST
GND
RX
TX
RTS
3V
RST
+5
RX
TX
3V


RSS
R1
VCC
R3


RSS
R1
ASC
VCC
AD1
R3
IC2
GND
C2



Step 10: Assemble and create the transmitter - 4

Now replace the PCB with the huge capacitor.

Clip the long leads down. You'll need to use the "-" stripe to keep track of which pin is negative and which is positive.

Tin both leads with solder.

Solder the other end of the red ribbon wire (that goes to +5V on the XBee adapter) to the positive pin of the capacitor.

Then solder the brown wire (that goes to GND on the XBee adapter) to the negative pin.

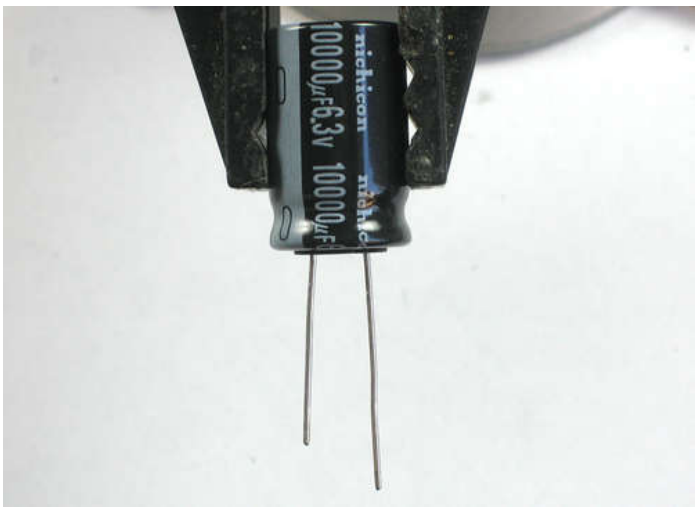
Clip the cathode lead down of the 1N4001 diode, that's the end with the white stripe. on it. Solder the diode so that the white-stripe side is connected to the positive pin of the big capacitor.

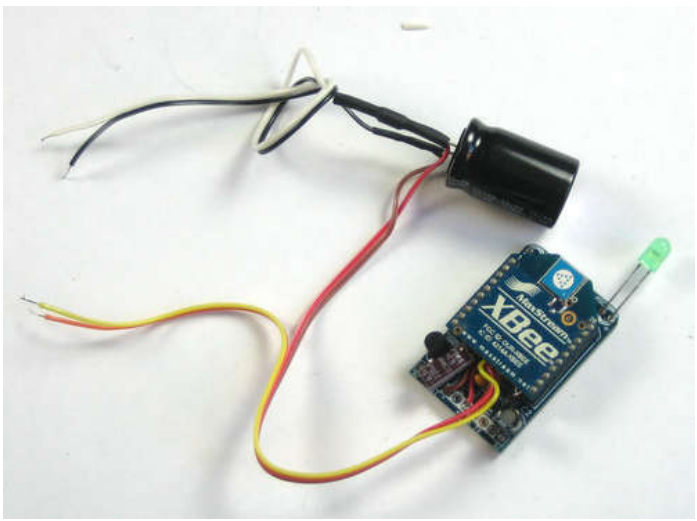
Take the black and white ribbon from earlier. Split, strip and tin the four ends. Cut a 1" piece of 1/8" heatshrink and slip it onto the white wire. Slip a 1/2" piece of 1/16" heat shrink onto the black wire.

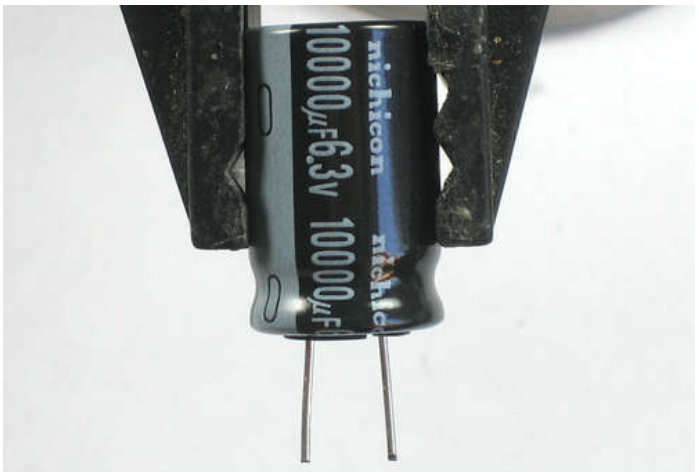
Clip the other end of the diode (the side without a white stripe) and solder the white wire to it. Solder the black wire to the negative pin of the big capacitor.

Now shrink the heatshrink so that the capacitor leads and diode are covered.

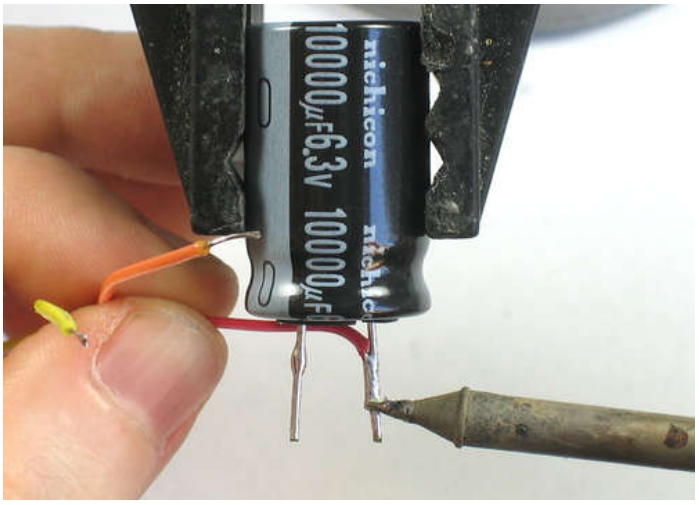
All right, here is what you should have, an adapter with two sensor lines (orange and yellow) hanging off and two power lines (red and brown) that are connected to the big capacitor. Then there are two black&white wires connected to the capacitor, the white one through a diode.

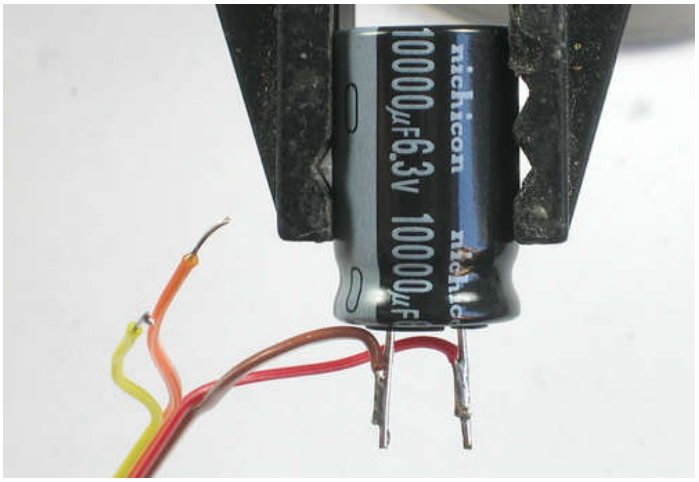


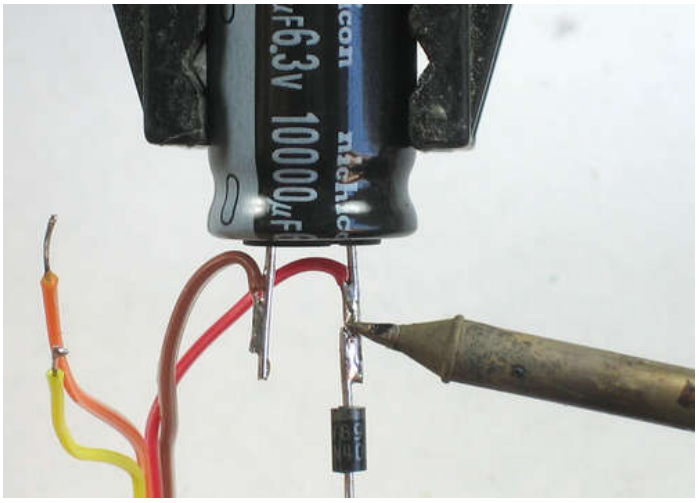


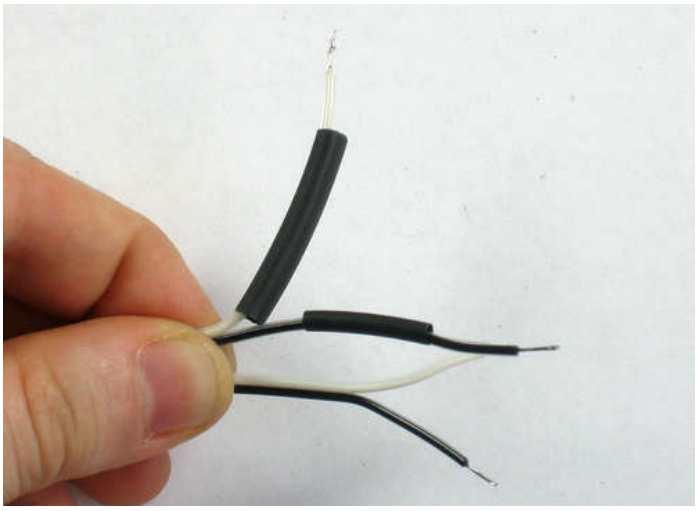

nichicon
10000µF6.3v 10000µF

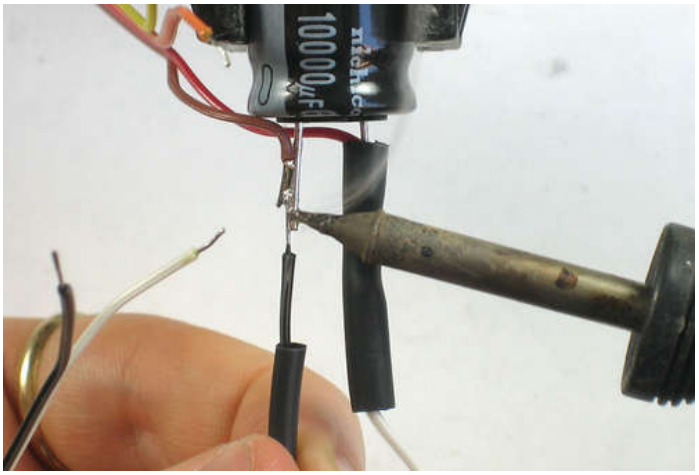

nichicon
10000µF6.3v 10000µF


nichicon
10000µF6.3v 10000µF


nichicon
10000µF6.3v 10000µF


6.3v 10000µF





Step 11: Assemble and create the transmitter - 5

Now its time to open the Kill-a-Watt! There are only 3 screws that hold it together, and they are found on the back.

Now its time to open the Kill-a-Watt! There are only 3 screws that hold it together, and they are found on the back.

Use a 3/8 drill bit to make a hole near the right corner of the case back. This is what the LED will stick out of. (Ignore the white tape and #4, this is a recycled kill-a-watt :)

Now find the LM2902N chip. This is a quad op-amp that senses the power line usage. We're going to piggy-back right on top of it, and borrow the ground, 5V power and 2 sensor outputs!

With your soldering iron, melt a bit of solder on pin 1, 4, 11 and 14 of the chip. Make sure you have the chip oriented correctly, the notch indicates where pins 1 and 14 are

Solder the white wire (5V to the XBee) to pin 4. Solder the black wire (ground) to pin 11 directly across.

Now solder the yellow wire to pin 1 and the orange wire to pin 14.

Use two small pieces of sticky foam and stick them onto the back of the case.

Then place the XBee adapter and capacitor on the tape so that the LED sticks out of the hole drilled earlier

Tuck the excess ribbon cable out of the way so that they are not near the 120V connections which could make them go poof.

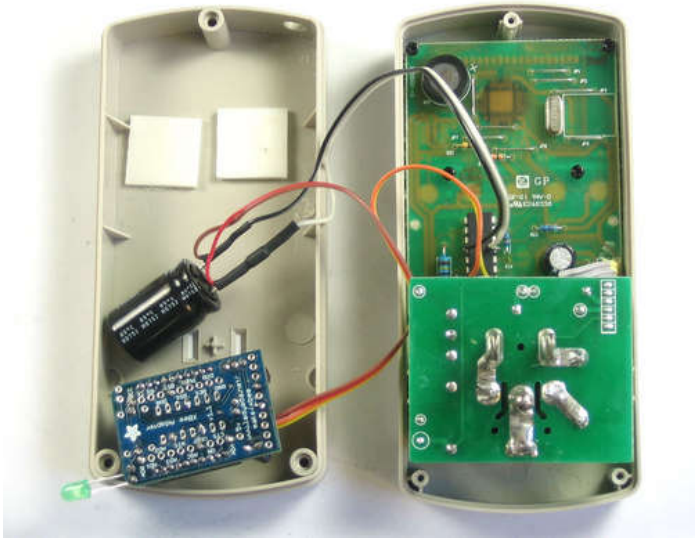
Close it up and plug it in.

You'll notice its a bit finicky for a few seconds as the big capacitor charges up. The display may not come up for 15-30 seconds, and it may fade in and out at first. The numbers may also be wrong for a bit as it powers up. Within about 30 seconds, you should see the display stabilize and the indicator LED blinking every 2 seconds!

Go back to your computer, plug the receiver XBee into the USB adapter and make sure it has the latest firmware uploaded and set it to the same PAN ID as the transmitters. You will see the RSSI LED (red LED) light up. That means you have a good link!

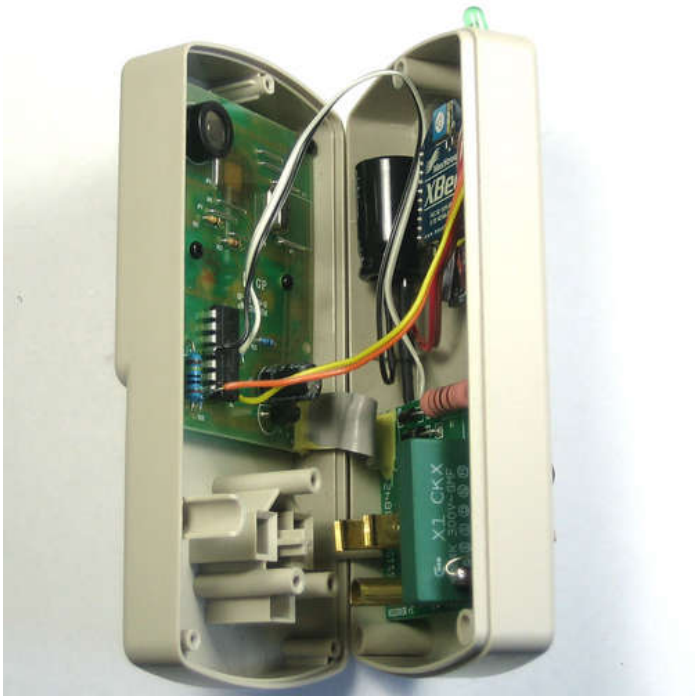
Open up the Terminal in X-CTU (or another terminal program) and connect at 9600 baud 8N1 parity and you'll see a lot of nonsense. Whats important is that a new chunk of nonsense gets printed out once every 2 seconds, indicating a packet of data has been received.

The hardware is done. Good work!

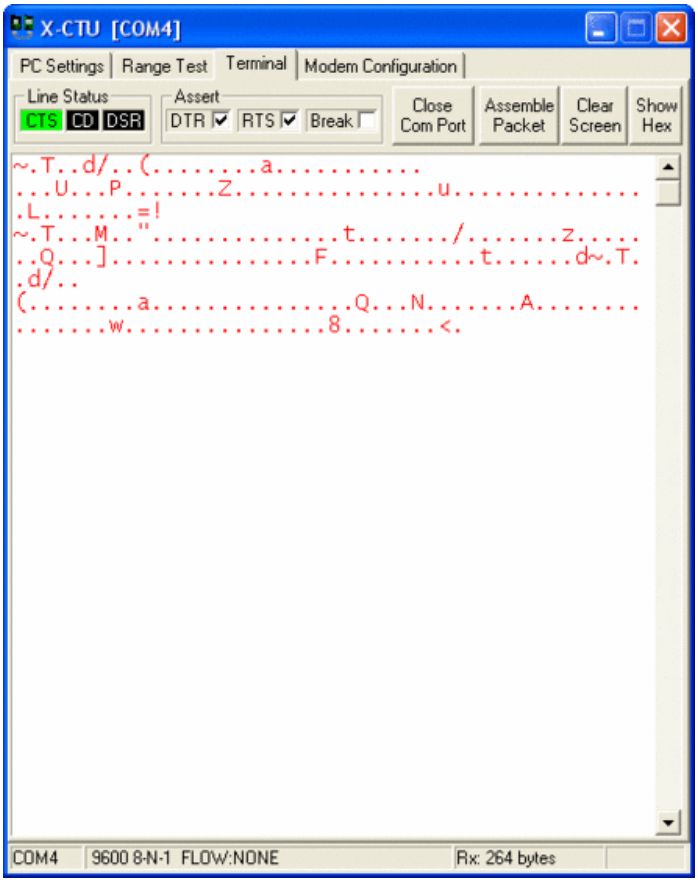



MaxStream
XBee
GP


MaxStream
XBee
GP

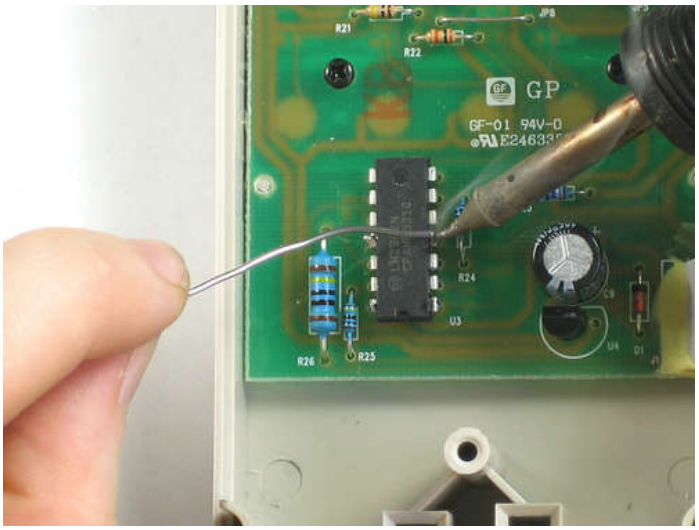

X1 CKX

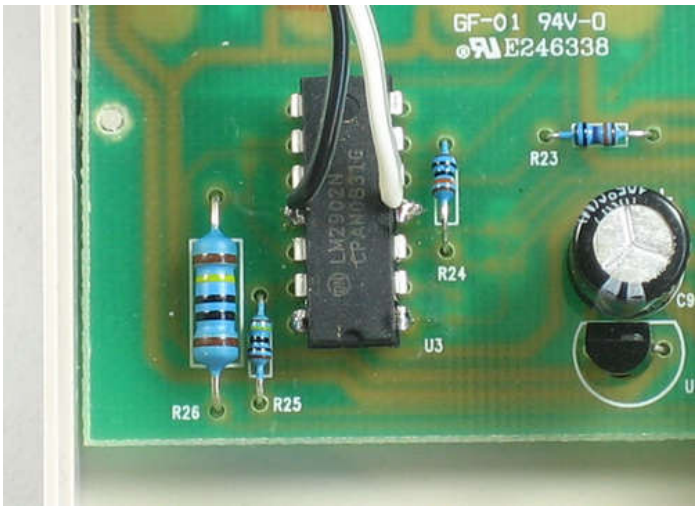

KILL A WATT™
123.7
Volt
Amp
Watt
Hz
KWH
VA
PF
Hour

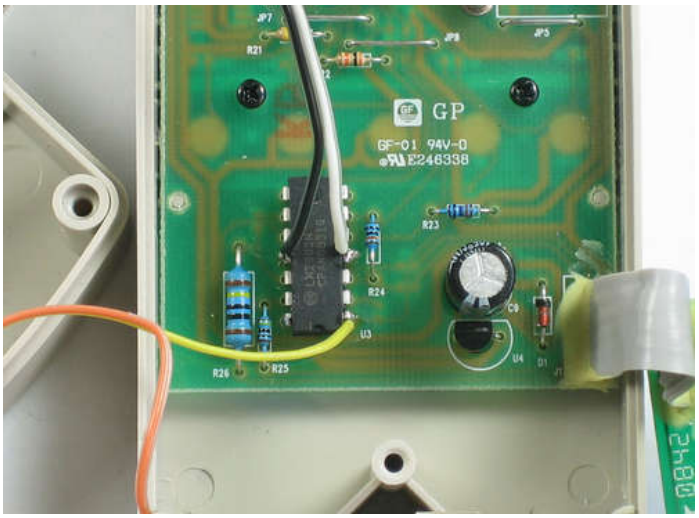

X-CTU [COM4]
PC Settings
Range Test
Terminal
Modem Configuration
Line Status
CTS
CD
DSR
Assert
DTR
RTS
Break
Close Com Port
Assemble Packet
Clear Screen
Show Hex
~.T..d/..(........a.....
...U...P.......Z.............u.....
.L.......=!
~.T...M..".............t......./.....z.....
..Q...]..............F..........t.....d~.T.
.d/..
(........a..............Q...N.......A.....
.......w..............8.......<.
COM4
9600 8-N-1 FLOW:NONE
Rx: 264 bytes

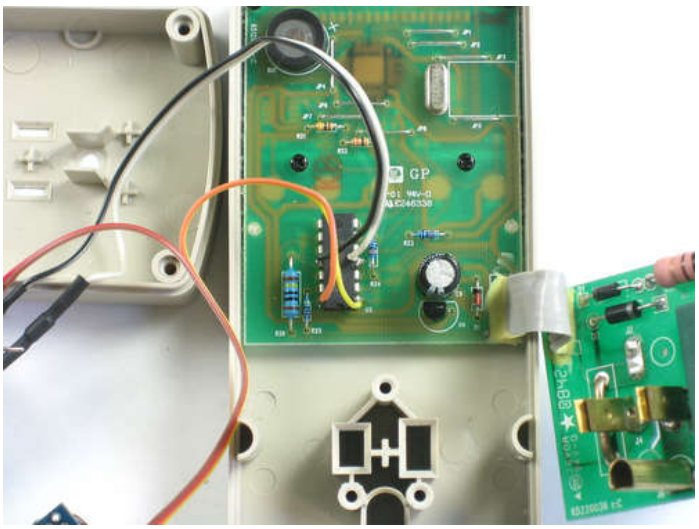

MODEL P4400 KILL A WATT™

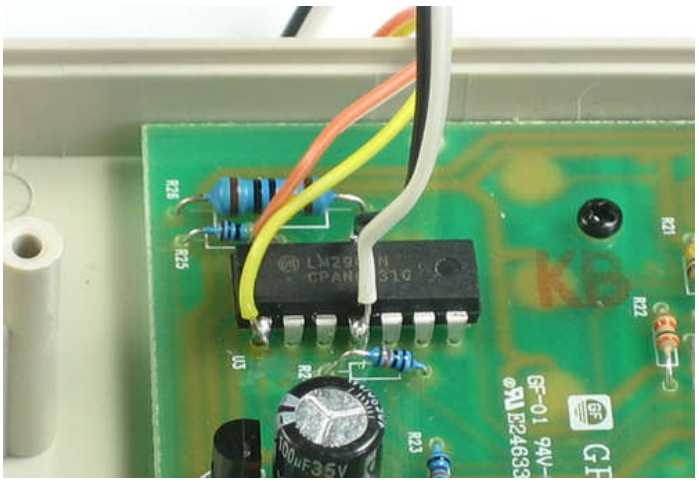

MODEL P4400 KILL A WATT™


GP
GF-01 94V-0


GF-01 94V-0
E246338
R23
R24
C9
U3
R26
R25


GP
GF-01 94V-0
E246338





Step 12: Software

Introduction

Now that the hardware is complete, we come to the exciting part: running the software that retrieves the data from our receiver XBee and saves it to our computer or uploads it to a database or updates our twitter feed or....whatever you'd like!

Here is how it works, the XBee inside the Kill-a-Watt is hooked up to two analog signals. One is the voltage signal which indicates the AC voltage read. In general this is a sine wave that is 120VAC. One tricky thing to remember is that 120V is the 'RMS' voltage, and the 'true voltage' is +-170VDC. (You can read more about RMS voltage at wikipedia basically its a way to indicate how much 'average' voltage there is.) The second reading is the AC current read. This is how much current is being drawn through the Kill-a-Watt. If you multiply the current by the voltage, you'll get the power (in Watts) used!

The XBee's Analog/Digital converter is set up to take a 'snapshot' of one sine-cycle at a time. Each double-sample (voltage and current) is taken 1ms apart and it takes 17 of them. That translates to a 17ms long train of samples. One cycle of power-usage is 1/60Hz long which is 16.6ms. So it works pretty well!

Lets look at some examples of voltage and current waveforms as the XBee sees them.

For example this first graph is of a laptop plugged in. You'll see that its a switching supply, and only pulls power during the peak of the voltage curve.

Now lets try plugging in a 40W incandescent light bulb. You'll notice that unlike the switching supply, the current follows the voltage almost perfectly. That's because a lightbulb is just a resistor!

Finally, lets try sticking the meter on a dimmable switch. You'll see that the voltage is 'chopped' up, no longer sinusoidal. And although the current follows the voltage, its still matching pretty well.

The XBee sends the raw data to the computer which, in a python script, figures out what the (calibrated) voltage and amperage is at each sample and multiplies each point together to get the Watts used in that cycle. Since there's almost no device that changes the power-usages from cycle-to-cycle, the snapshot is a good indicator of the overall power usage that second. Then once every 2 seconds, a single snapshot is sent to the receiver XBee

Install python & friends

The software that talks to the XBee is written in **python** . I used python because its quick to develop in, has multi-OS support and is pretty popular with software and hardware hackers. The XBees talk over the serial port so literally any programming language can/could be used here. If you're a software geek and want to use perl, C, C#, tcl/tk, processing, java, etc. go for it! You'll have to read the serial data and parse out the packet but its not particularly hard.

However, most people just want to get on with it and so for you we'll go through the process of installing **python** and the libraries we need.

1. Download and install python 2.5 from http://www.python.org/download/ I suggest 2.5 because that seems to be stable and well supported at this time. If you use another version there may be issues
2. Download and install pyserial from the package repository (this will let us talk to the XBee thru the serial port)
3. If you're running windows download and install win32file for python 2.5 (this will add file support)
4. Download and install the simplejson python library (this is how the twitter api likes to be spoken to)

Now you can finally download the Wattcher script we will demonstrate here! We're going to download it into the **C:\wattcher** directory, for other OS's you can of course change this directory

Basic configure

We'll have to do a little bit of setup to start, open up the **wattcher.py** script with a text editor and find the line

SERIALPORT = "COM4" # the com/serial port the XBee is connected to

change COM4 into whatever the serial port you will be connecting to the XBee with is called. Under windows its some **COMx** port, under linux and mac its something like **/dev/cu.usbserial-xxxx** check the /dev/ directory and/or **dmesg**

Save the script with the new serial port name

Test it out

Once you have installed python and extracted the scripts to your working directory, start up a terminal (under linux this is just **rxvt** or **xterm** , under mac its **Terminal** , under windows, its a **cmd** window)

I'm going to assume you're running windows from now on, it shouldn't be tough to adapt the instructions to linux/mac once the terminal window is open.

Run the command **cd C:\wattcher** to get to the place where you uncompressed the files. By running the **dir** command you can see that you have the files in the directory

Make sure your transmitter (Kill-a-Watt + Xbee) is plugged in, and blinking once every 2 seconds. Remember it takes a while for the transmitter to charge up power and start transmitting. The LCD display should be clear, not fuzzy. Make sure that there's nothing plugged into the Kill-a-Watt, too. The RSSI (red) LED on the receiver connected to the computer should be lit indicating data is being received. Don't continue until that is all good to go.

Now run **python** by running the command **C:\python25\python.exe wattcher.py**

You should get a steady print out of data. The first number is the XBee address from which it received data, following is the estimated current draw, wattage used and the Watt-hours consumed since the last data came in. Hooray! We have wireless data!

Calibrating

Now that we have good data being received, its time to tweak it. For example, its very likely that even without an appliance or light plugged into the Kill-a-Watt, the script thinks that there is power being used. We need to calibrate the sensor so that we know where 'zero' is. In the Kill-a-Watt there is an autocalibration system but unfortunately the XBee is not smart enough to do it on its own. So, we do it in the python script. Quit the script by typing in Control-C and run it again this time as **C:\python25\python.exe wattcher.py -d** note the **-d** which tells the script to print out debugging information

Now you can see the script printing out a whole mess of data. The first chunk with lots of -1's in it is the raw packet. While its interesting we want to look at the line that starts with **ampdata** :

ampdata: [498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 498, 497, 498, 498, 498]

Now you'll notice that the numbers are pretty much all the same. That's because there's nothing plugged into the tweetawatt and so each 1/60 Hz cycle has a flat line at

'zero'. The A/D in the XBee is 10 bits, and will return values between 0 and 1023. So, in theory, if the system is perfect the value at 'zero' should be 512. However, there are a bunch of little things that make the system imperfect and so zero is only close to 512. In this case the 'zero' calibration point is really 498. When its off there is a 'DC offset' to the Amp readings, as this graph shows:

See how the Amp line (green) is steady but its not at zero, its at 0.4 amps? There is a 'DC offset' of 0.4 amps

OK, open up the wattcher.py script in a text editor.

vrefcalibration = [492, # Calibration for sensor #0]
492, # Calibration for sensor #1
489, # Calibration for sensor #2
492, # Calibration for sensor #3
501, # Calibration for sensor #4
493] # etc... approx ((2.4v * (10Ko/14.7Ko)) / 3

See the line that says **# Calibration for sensor #1** ? Change that to **498**

vrefcalibration = [492, # Calibration for sensor #0]
498, # Calibration for sensor #1
489, # Calibration for sensor #2
492, # Calibration for sensor #3
501, # Calibration for sensor #4
493] # etc... approx ((2.4v * (10Ko/14.7Ko)) / 3

Save the file and start up the script again, this time without the **-d**

Now you'll see that the Watt draw is 2W or less, instead of 40W (which was way off!) The reason its not 0W is that, first off, there's a little noise that we're reading in the A/D lines, secondly there's power draw by the Kill-a-Watt itself and finally, the XBee doesn't have a lot of samples to work with. However <2W is pretty good considering that the full sensing range is 0-1500W

Note the graph with the calibrated sensor:

See how the Amps line is now at 0 steady, there is no DC offset

Logging data

Its nice to have this data but it would be even nicer if we could store it for use. Well, thats automatically done for you! You can set the name of the log file in the wattcher.py script. By default it's **powerdatalog.csv** . The script collects data and every 5 minutes writes a single line in the format **Year Month Day, Time, Sensor#, Watts** for each sensor.As you can see, this is an example of a 40W incandescent lightbulb plugged in for a few hours. Because of the low sample rate, you'll see some minor variations in the Watts recorded. This data can be easily imported directly into any spreadsheet program

Tweeting

Finally we get to the tweeting part of the tweet-a-watt. First open up the wattcher.py script and set

# Twitter username & password
twitterusername = "username"
twitterpassword = "password"

to your username and password on twitter. You can make an account on twitter.com if you don't have one.

Then run the script as usual. Every 8 hours (midnight, 8am and 4pm) the script will sent a twitter using the Twitter API

Then check it out at your account:

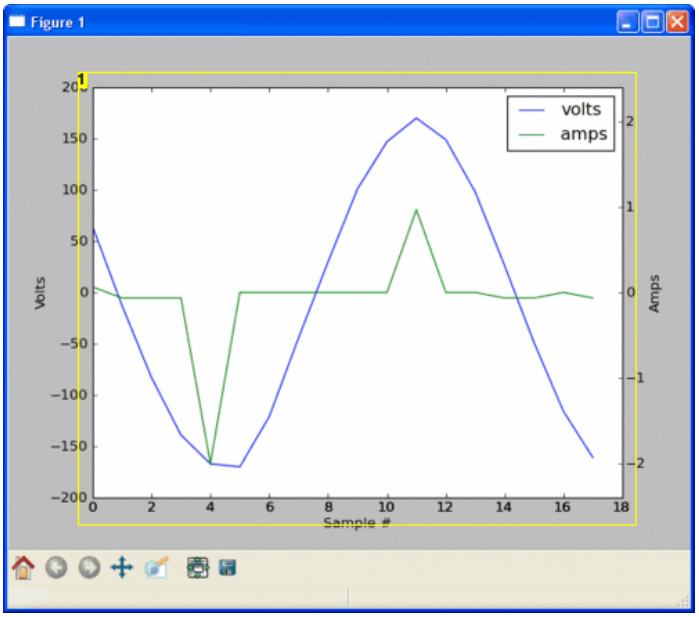


Image Notes
1. A laptop plugged in, switching power supply

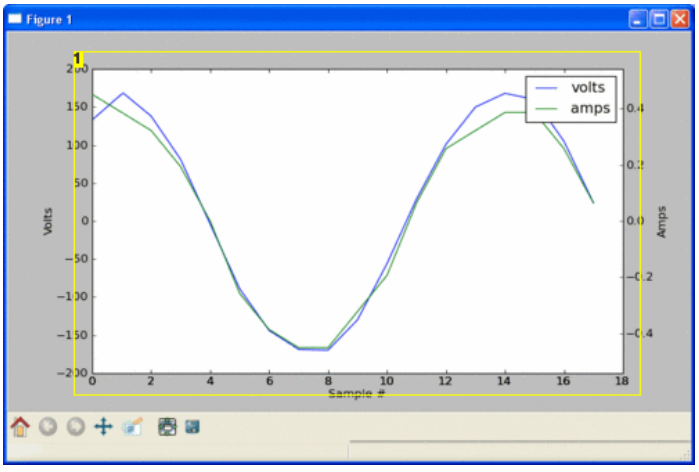
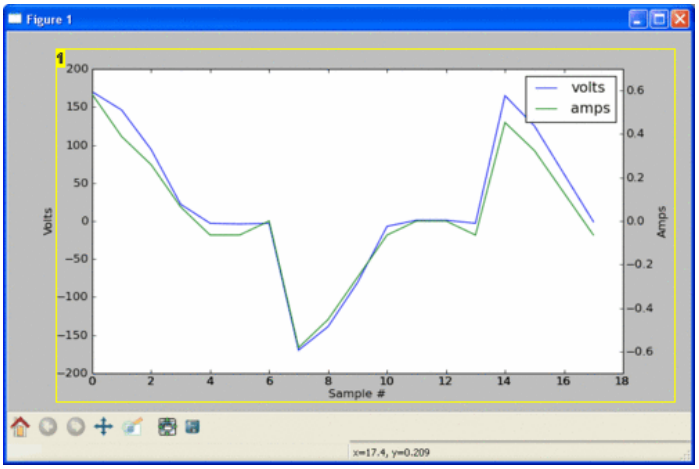


Image Notes
1. 40W lightbulb



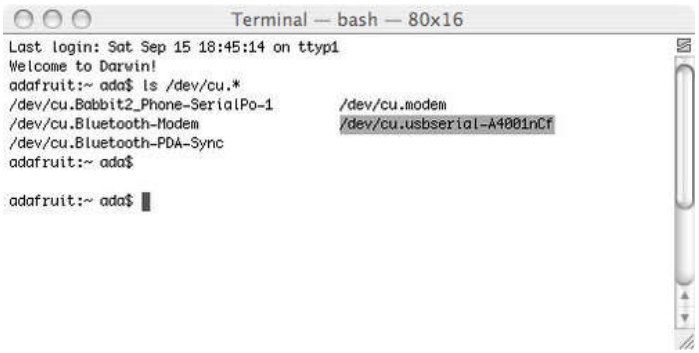
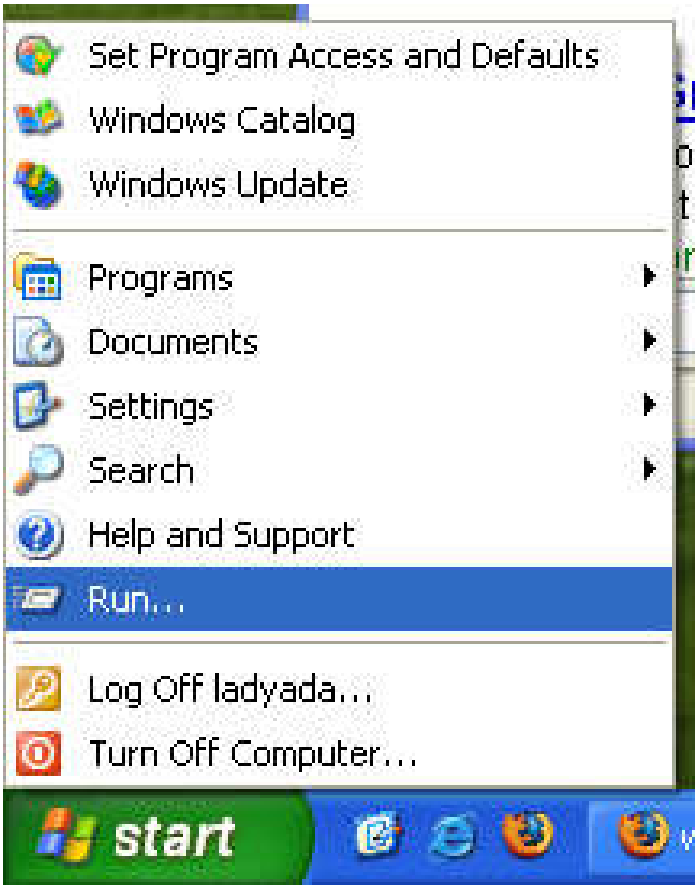
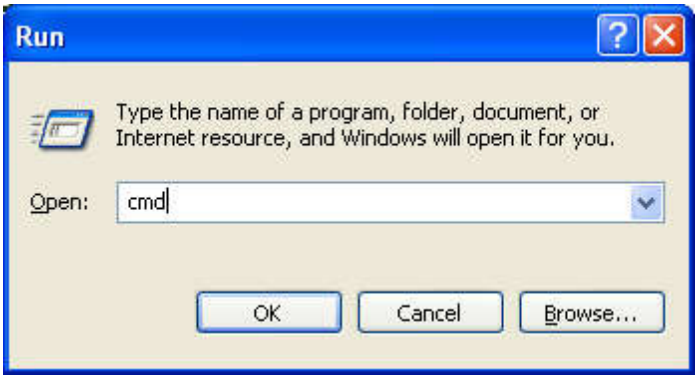
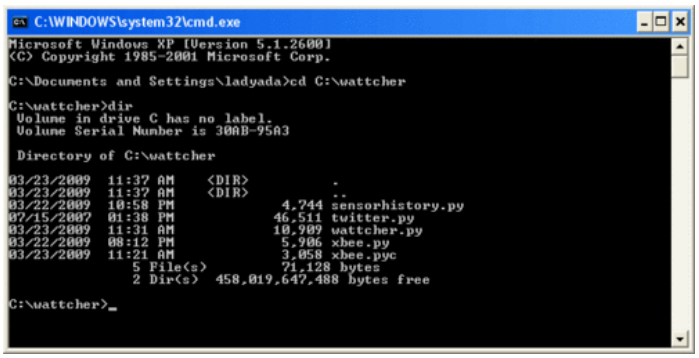
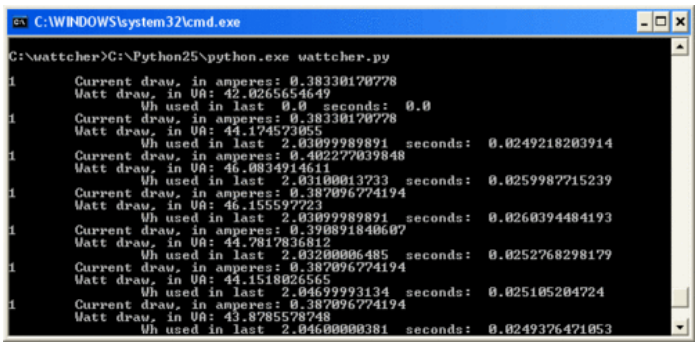


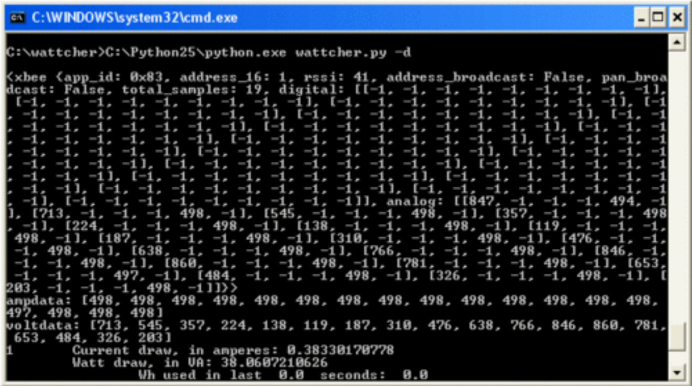
Image Notes
1. Light bulb on dimmer switch

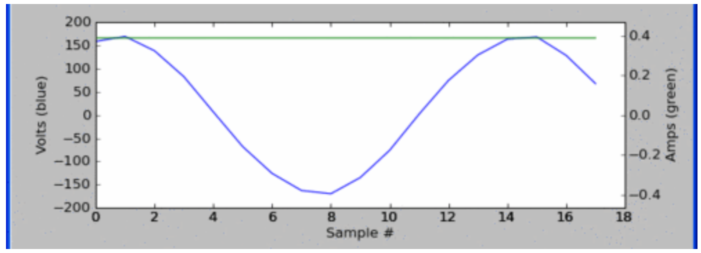


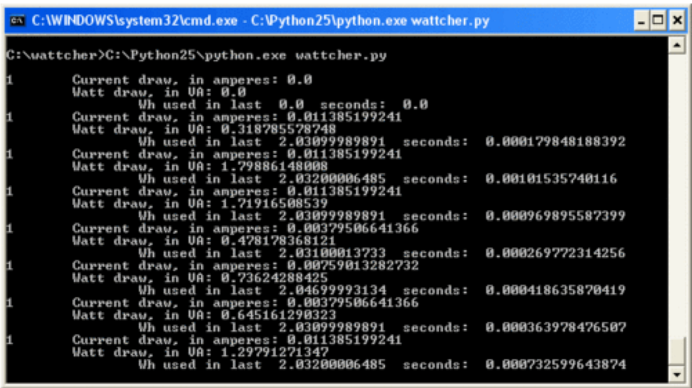


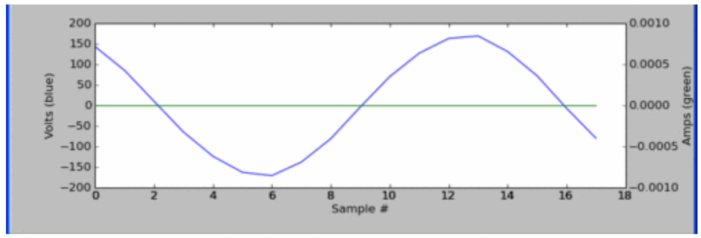


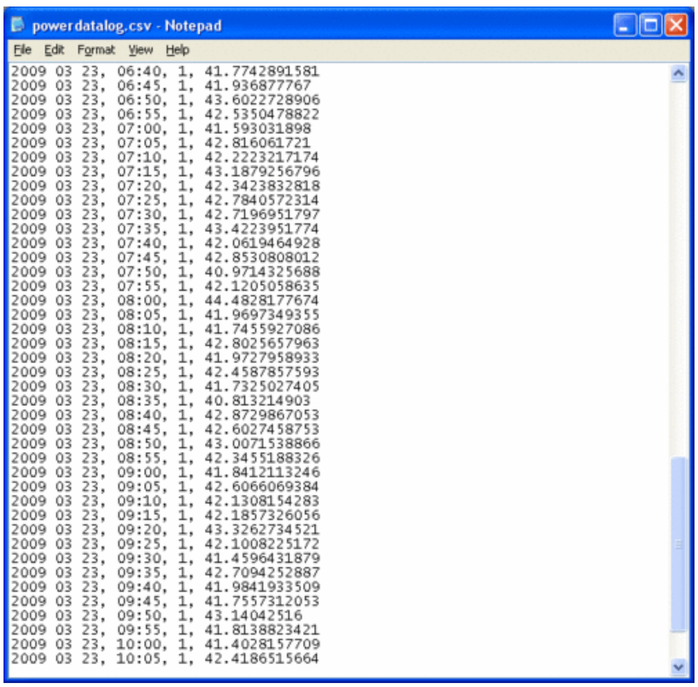






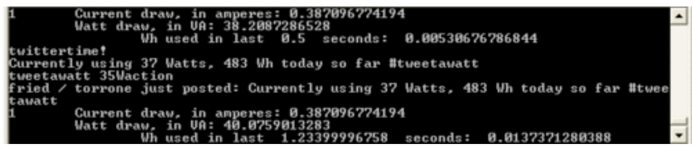





powerdatalog.csv - Notepad

File Edit Format View Help

```
2009 03 23, 06:40, 1, 41.7742891581
2009 03 23, 06:45, 1, 41.936877767
2009 03 23, 06:50, 1, 43.6022728906
2009 03 23, 06:55, 1, 42.5350478822
2009 03 23, 07:00, 1, 41.593031898
2009 03 23, 07:05, 1, 42.816061721
2009 03 23, 07:10, 1, 42.2223217174
2009 03 23, 07:15, 1, 43.1879256796
2009 03 23, 07:20, 1, 42.3423832818
2009 03 23, 07:25, 1, 42.7840572314
2009 03 23, 07:30, 1, 42.7196951797
2009 03 23, 07:35, 1, 43.4223951774
2009 03 23, 07:40, 1, 42.0619464928
2009 03 23, 07:45, 1, 42.8530808012
2009 03 23, 07:50, 1, 40.9714325688
2009 03 23, 07:55, 1, 42.1205058635
2009 03 23, 08:00, 1, 44.4828177674
2009 03 23, 08:05, 1, 41.9697349355
2009 03 23, 08:10, 1, 41.7455927086
2009 03 23, 08:15, 1, 42.8025657963
2009 03 23, 08:20, 1, 41.9727958933
2009 03 23, 08:25, 1, 42.4587857593
2009 03 23, 08:30, 1, 41.7325027405
2009 03 23, 08:35, 1, 40.813214903
2009 03 23, 08:40, 1, 42.8729867053
2009 03 23, 08:45, 1, 42.6027458753
2009 03 23, 08:50, 1, 43.0071538866
2009 03 23, 08:55, 1, 42.3455188326
2009 03 23, 09:00, 1, 41.8412113246
2009 03 23, 09:05, 1, 42.6066069384
2009 03 23, 09:10, 1, 42.1308154283
2009 03 23, 09:15, 1, 42.1857326056
2009 03 23, 09:20, 1, 43.3262734521
2009 03 23, 09:25, 1, 42.1008225172
2009 03 23, 09:30, 1, 41.4596431879
2009 03 23, 09:35, 1, 42.7094252887
2009 03 23, 09:40, 1, 41.9841933509
2009 03 23, 09:45, 1, 41.7557312053
2009 03 23, 09:50, 1, 43.14042516
2009 03 23, 09:55, 1, 41.8138823421
2009 03 23, 10:00, 1, 41.4028157709
2009 03 23, 10:05, 1, 42.4186515664
```



Step 13: Expand

Overview

Once you've got your base system up and running here are some ideas for how to extend, improve or expand it!

Add more outlets

So you can track more rooms, of course

Graphing

If you'd like to play some more with the script, there's some extras built in. For example, you can graph the data as it comes in from the XBee, both Watts used and the actual 'power line' waveform. Simply set **GRAPHIT = True** you'll need to install a mess of python libraries though, including **wxpython** , **numpy** and **pylab**

Remove the computer

It took a few hours, but I hacked my Asus wifi router to also log data for me. There'll be more documentation soon but here's some hints:

Do basically everything in [Do basically everything in MightyOhm's tutorial. You can use the FTDI cable to reprogram the router, just move the pins around. Then add a 16mb USB key (I was given one as schwag so look in your drawers) and install python and the openssl library as well as the other libraries needed like pyserial. The code should pretty much just run! (I'll put up more detailed notes later)]

The router still works as my wifi gateway to the cablemodem, and only uses 5W MightyOhm's tutorial]. You can use the FTDI cable to reprogram the router, just move the pins around. Then add a 16mb USB key (I was given one as schwag so look in your drawers) and install python and the openssl library as well as the other libraries needed like pyserial. The code should pretty much just run! (I'll put up more detailed notes later)

The router still works as my wifi gateway to the cablemodem, and only uses 5W

Step 14: Design - overview

Design overview

For those interested in how to build a sensor node system with a Google Appengine backend, here is the process by which I created it. Of course, you should have the hardware part done first!

1. Listen - designing the parser for the computer that grabs XBee packets, and extracts the useful data
2. Store - how to use GAE to store the data in 'the cloud'
3. Graph - using Google Visualizations to make pretty graphs

Step 15: Design - listen

Data listening & parsing

In this section we will work on the receiver software, that will talk to a receiver XBee and figure out what the sensor data means. I'll be writing the code in **python** which is a fairly-easy to use scripting language. It runs on all OS's and has tons of tutorials online. Also, Google AppEngine uses it so its a good time to learn!

This whole section assumes that you only have 1 transmitter and 1 receiver, mostly to make graphing easier to cope with. In the next section we'll tie in more sensors when we get to the datalogging part!

Raw analog input

We'll start by just getting raw data from the XBee and checking it out. The packet format for XBees is published but instead of rooting around in it, I'll just use the handy XBee library written for python. With it, I can focus on the data instead of counting bytes and calculating checksums.

To use the library, first use the **pyserial** module to open up a serial port (ie COM4 under windows, /dev/ttyUSB0 under mac/linux/etc) You can look at the XBee project page for information on how to figure out which COM port you're looking for. We connect at the standard default baudrate for XBees, which is 9600 and look for packets

```
from xbee import xbee
import serial

SERIALPORT = "COM4" # the com/serial port the XBee is connected to
BAUDRATE = 9600 # the baud rate we talk to the xbee

# open up the FTDI serial port to get data transmitted to xbee
ser = serial.Serial(SERIALPORT, BAUDRATE)
ser.open()

while True:
# grab one packet from the xbee, or timeout
packet = xbee.find_packet(ser)
if packet:
xb = xbee(packet)

print xb
```

Running this code, you'll get the following output:

<xbee {app_id: 0x83, address_16: 1, rssi: 85, address_broadcast: False, pan_broadcast: False, total_samples: 19, digital: [[-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1 , -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1 , -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1 , -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1 , -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1 , -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1 , -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1 , -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1 , -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1]], analog: [[190, -1, -1, -1, 489, -1], [109, -1, -1, -1, 484, -1], [150, -1, -1, -1, 492, -1], [262, -1, -1, -1, 492 , -1], [423, -1, -1, -1, 492, -1], [589, -1, -1, -1, 492, -1], [740, -1, -1, -1, 492, -1], [843, -1, -1, -1, 492, -1], [870, -1, -1, -1, 496, -1], [805, -1, -1, -1, 491, -1], [680, -1, -1, -1, 492, -1], [518, -1, -1, -1, 492, -1], [349, -1, -1, -1, 491, -1], [199, -1, -1, -1, 491, -1], [116, -1, -1, -1, 468, -1], [108, -1, -1, -1, 492, -1], [198, -1, -1, -1, 492, -1], [335, -1, -1, -1, 492, -1], [523, -1, -1, -1, 492, -1]]}>

which we will reformat to make a little more legible

```
<xbee {
app_id: 0x83,
address_16: 1,
rssi: 85,
address_broadcast: False,
pan_broadcast: False,
total_samples: 19,
digital: [[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1],
[-1, -1, -1, -1, -1, -1, -1, -1, -1]],
analog: [[190, -1, -1, -1, 489, -1],
[109, -1, -1, -1, 484, -1],
[150, -1, -1, -1, 492, -1],
[262, -1, -1, -1, 492, -1],
[423, -1, -1, -1, 492, -1],
[589, -1, -1, -1, 492, -1],
[740, -1, -1, -1, 492, -1],
[843, -1, -1, -1, 492, -1],
[870, -1, -1, -1, 496, -1],
[805, -1, -1, -1, 491, -1],
[680, -1, -1, -1, 492, -1],
```

[518, -1, -1, -1, 492, -1],
[349, -1, -1, -1, 491, -1],
[199, -1, -1, -1, 491, -1],
[116, -1, -1, -1, 468, -1],
[108, -1, -1, -1, 492, -1],
[198, -1, -1, -1, 492, -1],
[335, -1, -1, -1, 492, -1],
[523, -1, -1, -1, 492, -1]]

}>

OK now its clear whats going on here. First off, we get some data like the transmitter ID (address_16) and signal strength (RSSI). The packet also tells us how many sample are available (19). Now, the digital samples are all -1 because we didn't request any to be sent. The library still fills them in tho so thats why the non-data is there. The second chunk is 19 sets of analog data, ranging from 0 to 1023. As you can see, the first sample (#0) and fifth sample (#4) contain real data, the rest are -1. That corresponds to the hardware section where we setup AD0 and AD4 to be our voltage and current sensors.

We'll tweak our code so that we can extract this data only and ignore the rest of the packet.

This code creates two arrays, voltagedata and ampdata where we will stick the data. We throw out the first sample because usually ADCs are a bit wonky on the first sample and then are good to go after that. It may not be necessary tho

```
#!/usr/bin/env python
import serial
from xbee import xbee

SERIALPORT = "COM4" # the com/serial port the XBee is connected to
BAUDRATE = 9600 # the baud rate we talk to the xbee
CURRENTSENSE = 4 # which XBee ADC has current draw data
VOLTSENSE = 0 # which XBee ADC has mains voltage data

# open up the FTDI serial port to get data transmitted to xbee
ser = serial.Serial(SERIALPORT, BAUDRATE)
ser.open()

while True:
# grab one packet from the xbee, or timeout
packet = xbee.find_packet(ser)
if packet:
xb = xbee(packet)

#print xb
# we'll only store n-1 samples since the first one is usually messed up
voltagedata = [-1] * (len(xb.analog_samples) - 1)
ampdata = [-1] * (len(xb.analog_samples ) -1)
# grab 1 thru n of the ADC readings, referencing the ADC constants
# and store them in nice little arrays
for i in range(len(voltagedata)):
voltagedata[i] = xb.analog_samples[i+1][VOLTSENSE]
ampdata[i] = xb.analog_samples[i+1][CURRENTSENSE]
print voltagedata
print ampdata
```

Now our data is easier to see:

Voltage: [672, 801, 864, 860, 755, 607, 419, 242, 143, 108, 143, 253, 433, 623, 760, 848, 871, 811]
Current: [492, 492, 510, 491, 492, 491, 491, 491, 492, 480, 492, 492, 492, 492, 492, 492, 497, 492]

Note that the voltage swings from about 100 to 900, sinusoidally.

Normalizing the data

Next up we will 'normalize' the data. The voltage should go from -170 to +170 which is the actual voltage on the line, instead of 100 to 900 which is just what the ADC reads. To do that we will get the average value of the largest and smallest reading and subtract it from all the samples. After that, we'll normalize the Current measurements as well, to get the numbers to equal the current draw in Amperes.

```
#!/usr/bin/env python
import serial
from xbee import xbee

SERIALPORT = "COM4" # the com/serial port the XBee is connected to
BAUDRATE = 9600 # the baud rate we talk to the xbee
CURRENTSENSE = 4 # which XBee ADC has current draw data
VOLTSENSE = 0 # which XBee ADC has mains voltage data

# open up the FTDI serial port to get data transmitted to xbee
ser = serial.Serial(SERIALPORT, BAUDRATE)
ser.open()

while True:
# grab one packet from the xbee, or timeout
packet = xbee.find_packet(ser)
if packet:
xb = xbee(packet)
```

```
#print xb
# we'll only store n-1 samples since the first one is usually messed up
voltagedata = [-1] * (len(xb.analog_samples) - 1)
ampdata = [-1] * (len(xb.analog_samples ) -1)
# grab 1 thru n of the ADC readings, referencing the ADC constants
# and store them in nice little arrays
for i in range(len(voltagedata)):
voltagedata[i] = xb.analog_samples[i+1][VOLTSENSE]
ampdata[i] = xb.analog_samples[i+1][CURRENTSENSE]

# get max and min voltage and normalize the curve to '0'
# to make the graph 'AC coupled' / signed
min_v = 1024 # XBee ADC is 10 bits, so max value is 1023
max_v = 0
for i in range(len(voltagedata)):
if (min_v > voltagedata[i]):
min_v = voltagedata[i]
if (max_v < voltagedata[i]):
max_v = voltagedata[i]

# figure out the 'average' of the max and min readings
avgv = (max_v + min_v) / 2
# also calculate the peak to peak measurements
vpp = max_v-min_v

for i in range(len(voltagedata)):
#remove 'dc bias', which we call the average read
voltagedata[i] -= avgv
# We know that the mains voltage is 120Vrms = +-170Vpp
voltagedata[i] = (voltagedata[i] * MAINSVPP) / vpp

# normalize current readings to amperes
for i in range(len(ampdata)):
# VREF is the hardcoded 'DC bias' value, its
# about 492 but would be nice if we could somehow
# get this data once in a while maybe using xbeeAPI
ampdata[i] -= VREF
# the CURRENTNORM is our normalizing constant
# that converts the ADC reading to Amperes
ampdata[i] /= CURRENTNORM

print "Voltage, in volts: ", voltagedata
print "Current, in amps: ", ampdata
```

We'll run this now to get this data, which looks pretty good, there's the sinusoidal voltage we are expecting and the current is mostly at 0 and then peaks up and down once in a while. Note that the current is sometimes negative but that's OK because we multiply it by the voltage and if both are negative it still comes out as a positive power draw

Voltage, in volts: [-125, -164, -170, -128, -64, 11, 93, 148, 170, 161, 114, 46, -39, -115, -157, -170, -150, -99]

Current, in amps: [0.064516129032258063, -1.096774193548387, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.096774193548387,]
0.0, 0.0, 0.0, -0.064516129032258063, 0.0, 0.0, -0.70967741935483875, 0.0, 0.0]

Basic data graphing

Finally, I'm going to add a whole bunch more code that will use the **numpy** graphing modules to make a nice graph of our data. Note that you'll need to install **wxpython** as well as **numpy** , and **matplotlib** !

At this point, the code is getting waaay to big to paste here so grab "wattcher.py Mains graph" from the download page!

Run it and you should see a graph window pop up with a nice sinusoidal voltage graph and various amperage data. For example this first graph is of a laptop plugged in. You'll see that its a switching supply, and only pulls power during the peak of the voltage curve.

Now lets try plugging in a 40W incandescent light bulb. You'll notice that unlike the switching supply, the current follows the voltage almost perfectly. Thats because a lightbulb is just a resistor!

Finally, lets try sticking the meter on a dimmable switch. You'll see that the voltage is 'chopped' up, no longer sinusoidal. And although the current follows the voltage, its still matching pretty well.

Graphing wattage!

OK neat, its all fun to watch waveforms but what we -really want- is the Watts used. Remember, P = VI otherwise known as Watts = Voltage * Current. We can calculate total Watts used by multiplying the voltages and currents at each sample point, then summing them up over a cycle & averaging to get the power used per cycle. Once we have Watts, its easy to just multiply that by 'time' to get Watt-hours!

Download and run the wattcher.py - watt grapher script from the download page

Now you can watch the last hour's worth of watt history (3600 seconds divided by 2 seconds per sample = 1800 samples) In the image above you can see as I dim a 40-watt lightbulb. The data is very 'scattered' looking because we have not done any low-pass filtering. If we had a better analog sampling rate, this may not be as big a deal but with only 17 samples to work with, precision is a little difficult

Done!

OK great! We have managed to read data, parse out the analog sensor payload and process it in a way that gives us meaningful graphs. Of course, this is great for

instantaneous knowledge but it -would- be nice if we could have longer term storage, and also keep track of multiple sensors. In the next step we will do that by taking advantage of some free 'cloud computing' services!

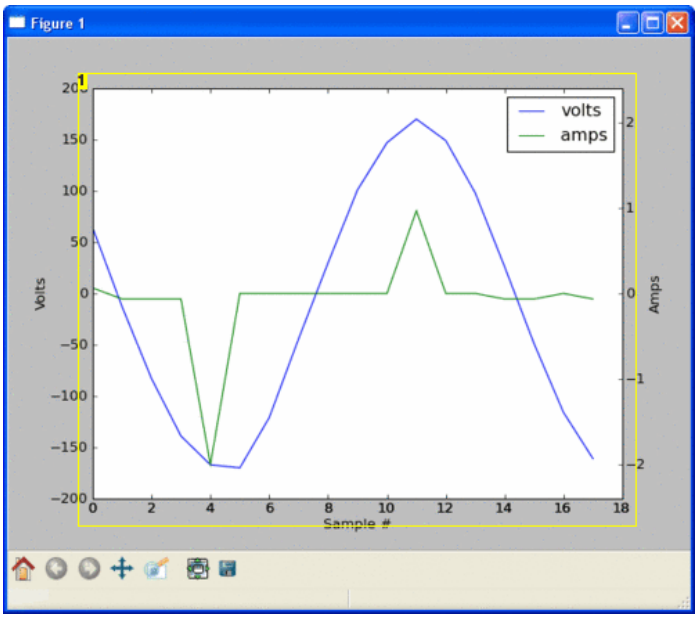


Image Notes
1. A laptop plugged in, switching power supply

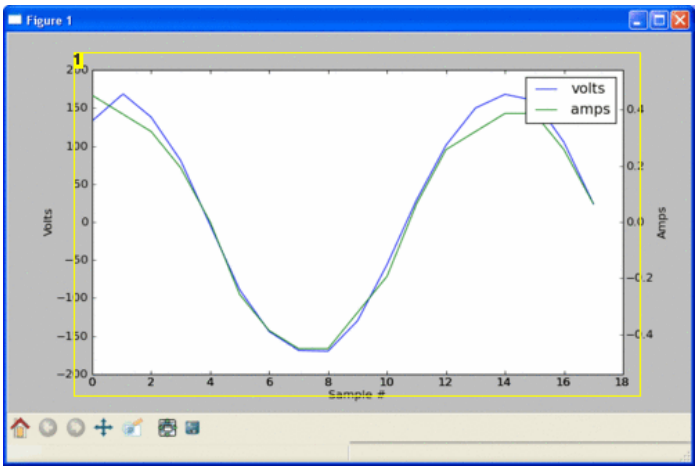


Image Notes
1. 40W lightbulb

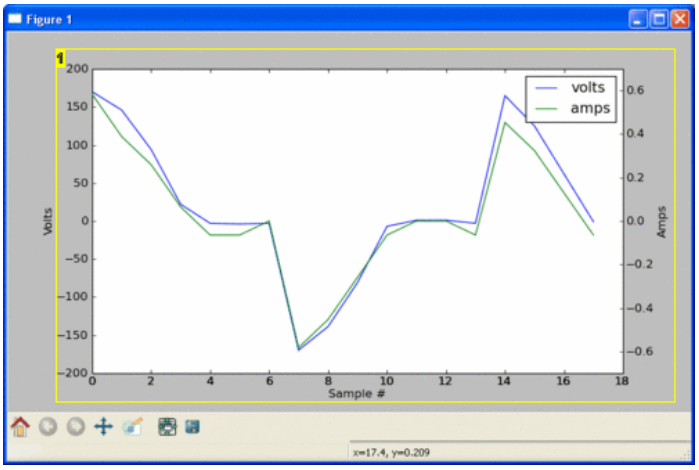
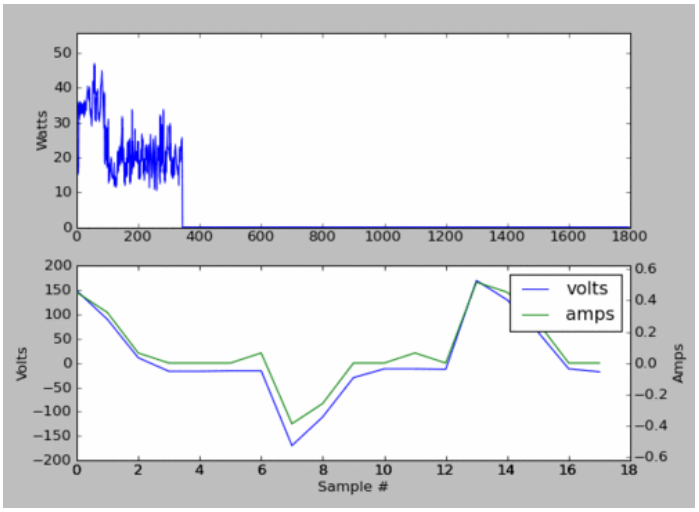


Image Notes
1. Light bulb on dimmer switch



Step 16: Design - store

Introduction

OK we are getting good data from our sensors, lets corral it into more useful chunks and store it in a database. We could make a database on the computer, but since we'd like to share this data, it makes more sense to put it online. There are custom services that are specifically designed to do this sort of thing like Pachube but I'm going to reinvent the wheel and design my own web-app that stores and displays energy data. (Mostly I want to play around with Google App Engine!)

You have 5 minutes!

We get data every few seconds from the XBee modem inside the kill-a-watt. We could, in theory, put data into our database every 2 seconds but that would quickly balloon the amount of storage necessary. It would also make sorting through the data difficult. So instead lets add up all the sensor data for 5 minutes and then take the average.

We'll do this by keeping two timers and one tally. One timer will track how long its been since the last sensor signal was sent, and the other will track if its been 5 minutes. The tally will store up all the Watt-hours (Watt measurements * time since last sensor data). Then at the end we can average by the 5 minutes

This chunk of code goes near the beginning, it creates the timers and tally and initializes them

```
...
fiveminutetimer = lasttime = time.time() # get the current time
cumulativewatthr = 0
...
```

Then later on, after we get our data we can put in this chunk of code:

```
# add up the delta-watthr used since last reading
# Figure out how many watt hours were used since last reading
elapsedseconds = time.time() - lasttime
dwatthr = (avgwatt * elapsedseconds) / (60.0 * 60.0) # 60 seconds in 60 minutes = 1 hr
lasttime = time.time()
print "\t\tWh used in last ",elapsedseconds," seconds: ",dwatthr
cumulativewatthr += dwatthr

# Determine the minute of the hour (ie 6:42 -> '42')
currminute = (int(time.time())/60) % 10
# Figure out if its been five minutes since our last save
if (((time.time() - fiveminutetimer) >= 60.0) and (currminute % 5 == 0)):
# Print out debug data, Wh used in last 5 minutes
avgwattsused = cumulativewatthr * (60.0*60.0 / (time.time() - fiveminutetimer))
print time.strftime("%Y %m %d, %H:%M"),", ",cumulativewatthr,"Wh = ",avgwattsused," W average")

# Reset our 5 minute timer
fiveminutetimer = time.time()
cumulativewatthr = 0
```

Note that we calculate delta-watthours, the small amount of power used every few seconds. Then we can get the average watts used by dividing the watthours by the number of hours that have passed (about 1/12th). Instead of going by exact 5 minutes, I decided to only report on the 5's of the hour (:05, :10, etc) so that its easier to send all the data at once if theres multiple sensors that started up at different times.

Download **wattcher-5minreporter.py** from the Download page. If you run this, you'll get a steady stream

Near the end you can see the timestamp, the Watthrs used in the last few minutes and the average Wattage

Multisensor!

We have good data but so far it only works with one sensor. Multiple sensors will mess it up! Time to add support for more than one XBee so that I can track a few rooms. I'll do that by creating an object class in python, and using the XBee address (remember that from part 1?) to track. I'll replace the code we just wrote with the following:

At the top, instead of the timer variables, I'll have a full class declaration, and create an array to store them:

```
####### store sensor data and array of histories per sensor
class Fiveminutehistory:
def init (self, sensornum):
self.sensornum = sensornum
self.fiveminutetimer = time.time() # track data over 5 minutes
self.lasttime = time.time()
self.cumulativewatthr = 0

def addwatthr(self, deltawatthr):
self.cumulativewatthr += float(deltawatthr)

def reset5mintimer(self):
self.cumulativewatthr = 0
self.fiveminutetimer = time.time()

def avgwattover5min(self):
return self.cumulativewatthr * (60.0*60.0 / (time.time() - self.fiveminutetimer))

def str (self):
return "[id#: %d, 5mintimer: %f, lasttime; %f, cumulativewatthr: %f]" % (self.sensornum, self.fiveminutetimer, self.lasttime, self.cumulativewatthr)

######### an array of histories
sensorhistories = []
```

When the object is initialized with the sensor ID number, it also sets up the two timers and cumulative Watthrs tracked. I also created a few helper functions that will make the code cleaner

Right below that I'll create a little function to help me create and retrieve these objects. Given an XBee ID number it either makes a new one or gets the reference to it

```
####### retriever
def findsensorhistory(sensornum):
for history in sensorhistories:
if history.sensornum == sensornum:
return history
# none found, create it!
history = Fiveminutehistory(sensornum)
sensorhistories.append(history)
return history
```

Finally, instead of the average Watt calculation code written up above, we'll replace it with the following chunk, which retreives the object and tracks power usage with the object timers

```
# retreive the history for this sensor
sensorhistory = findsensorhistory(xb.address_16)
#print sensorhistory

# add up the delta-watthr used since last reading
# Figure out how many watt hours were used since last reading
elapsedseconds = time.time() - sensorhistory.lasttime
dwatthr = (avgwatt * elapsedseconds) / (60.0 * 60.0) # 60 seconds in 60 minutes = 1 hr
sensorhistory.lasttime = time.time()
```

```
print "\t\tWh used in last ",elapsedseconds," seconds: ",dwatthr
sensorhistory.addwatthr(dwatthr)

# Determine the minute of the hour (ie 6:42 -> '42')
currminute = (int(time.time())/60) % 10
# Figure out if its been five minutes since our last save
if (((time.time() - sensorhistory.fiveminutetimer) >= 60.0) and (currminute % 5 == 0)):
# Print out debug data, Wh used in last 5 minutes
avgwattsused = sensorhistory.avgwattover5min()
print time.strftime("%Y %m %d, %H:%M"),", ",sensorhistory.cumulativewatthr,"Wh = ",avgwattsused," W average"

# Reset our 5 minute timer
sensorhistory.reset5mintimer()
```

The code basically acts the same except now it wont choke on multiple sensor data! Below, my two Kill-a-Watts, one with a computer attached (100W) and another with a lamp (40W)

Onto the database!

The App Engine

So we want to have an networked computer to store this data so we can share the data, but we really don't want to have to run a server from home! What to do? Well as mentioned before, you can use Pachube or similar, but I will show how to roll-your-own with Google App Engine (GAE) . GAE is basically a free mini-webserver hosted by Google, that will run basic webapps without the hassle of administrating a database server. Each webapp has storage, some frameworks and can use Google accounts for authentication. To get started I suggest checking out the GAE website, documentation, etc. I'll assume you've gone through the tutorials and jump right into designing my power data storage app called Wattcher (a little confusing I know)

First, the **app.yaml** file which defines my app looks like this:

```
application: wattcher
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
script: wattcherapp.py
```

Pretty simple, just says that the app uses **wattcherapp.py** as the source file

Next, we'll dive into the python code for our webapp. First, the includes and database index. To create a database, we actually define it -in the python file-, GAE then figures out what kind of database to create for you by following those directions (very different than MySQL where you'd create the DB separately)

```
import cgi, datetime

from google.appengine.api import users
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext import db

class Powerusage(db.Model):
author = db.UserProperty() # the user
sensornum = db.IntegerProperty() # can have multiple sensors
watt = db.FloatProperty() # each sending us latest Watt measurement
date = db.DateTimeProperty(auto_now_add=True) # timestamp
```

We use the default includes. We have a single database table called **Powerusage** , and it has 4 entries: one for the user, one for the sensor number, one for the last reported Watts used and one for a datestamp

Each 'page' or function of our webapp needs its own class. Lets start with the function that allows us to store data in the DB. I'll call it PowerUpdate.

```
class PowerUpdate(webapp.RequestHandler):
def get(self):

# make the user log in
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))

powerusage = Powerusage()

if users.get_current_user():
powerusage.author = users.get_current_user()
#print self.request
if self.request.get('watt'):
powerusage.watt = float(self.request.get('watt'))
else:
self.response.out.write('Couldnt find \'watt\' GET property!')
return
if self.request.get('sensornum'):
powerusage.sensornum = int(self.request.get('sensornum'))
else:
powerusage.sensornum = 0 # assume theres just one or something

powerusage.put()
self.response.out.write('OK!')
```

When we send a request to do that with a GET call (ie requesting the webpage), we'll first make sure the user is authenticated and logged in so we know their name. Then we'll create a new database entry by initializing a new instantiation of Powerusage. Then we'll look the GET request for the watt data, which would be in the format watt=39.2 or similar. That's parsed for us, thankfully and we can also get the sensor number which is passed in the format sensornum=3. Finally we can store the data into the permanent database

Next is a useful debugging function, it will simply print out all the data it has received for your account!

```
class DumpData(webapp.RequestHandler):
def get(self):

# make the user log in
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))

self.response.out.write('<html><body>Here is all the data you have sent us:<p>')

powerusages = db.GqlQuery("SELECT * FROM Powerusage WHERE author = :1 ORDER BY date", users.get_current_user())

for powerused in powerusages:
if powerused.sensornum:
self.response.out.write('<b>%s</b>\'s sensor #%d' %
(powerused.author.nickname(), powerused.sensornum))
else:
self.response.out.write(<b>%s</b>' % powerused.author.nickname())

self.response.out.write(' used: %f Watts at %s<p>' % (powerused.watt, powerused.date))
self.response.out.write("</body></html>")
```

This function simply SELECT's (retrieves) all the entries, sorts them by date and prints out each one at a time

Finally we'll make a basic 'front page' that will show the last couple of datapoints sent

```
class MainPage(webapp.RequestHandler):
def get(self):

self.response.out.write('<html><body>Welcome to Wattcher!<p>Here are the last 10 datapoints:<p>')

powerusages = db.GqlQuery("SELECT * FROM Powerusage ORDER BY date DESC LIMIT 10")

for powerused in powerusages:
if powerused.sensornum:
self.response.out.write('<b>%s</b>\'s sensor #%d' %
(powerused.author.nickname(), powerused.sensornum))
else:
self.response.out.write('<b>%s</b>' % powerused.author.nickname())

self.response.out.write(' used: %f Watts at %s<p>' % (powerused.watt, powerused.date))
self.response.out.write("</body></html>")
```

Its very similar to the DataDump function but its only 10 points of data and from all users, nice to use when you just want to 'check it out' but don't want to log in

Finally, we have a little initializer structure that tells GAE what pages link to what functions

```
application = webapp.WSGIApplication(
[('/', MainPage),]
('/report', PowerUpdate),
('/dump', DumpData)],
debug=True)

def main():
run_wsgi_app(application)

if name == "main ":
main()
```

Test!

OK lets try it out, first lets visit **http://wattcher.appspot.com/report**

Remember we made it a requirement to supply -some- data. Lets try again **http://wattcher.appspot.com/report?watt=19.22&sensornum=1**

Yay we got an OK! Lets check out the data stored by visiting **http://wattcher.appspot.com/dump**

There's two entries because I did a little testing beforehand but you can see that there are 2 entries. Nice!

We can also visit the GAE control panel and browse the data 'by hand'

Anyways, now that that's working, lets go back and add the reporting technology to our sensor-reader script

Getting the report out

Only a little more hacking on the computer script and we're done. We want to add support for sending data to GAE. Unfortunately right now our authentication is done through Google accounts so its not easy to run on an Arduino. To adapt it you'd have to send the username in the Report GET and hope nobody else uses the same one (unless you also add a basic password system)

Anyhow, I totally ripped off how to do this from some nice people on the Internet

Download **appengineauth.py** from the download page , and change the first few lines if necessary. We hardcode the URL we're going to and the account/password as

well as the GAE app name

```
users_email_address = "myaccount@gmail.com"
users_password = "mypassword"
my_app_name = "wattcher"
target_authenticated_google_app_engine_uri = 'http://wattcher.appspot.com/report'
```

The real work happens at this function **sendreport** where it connects and sends the Watt data to the GAE site

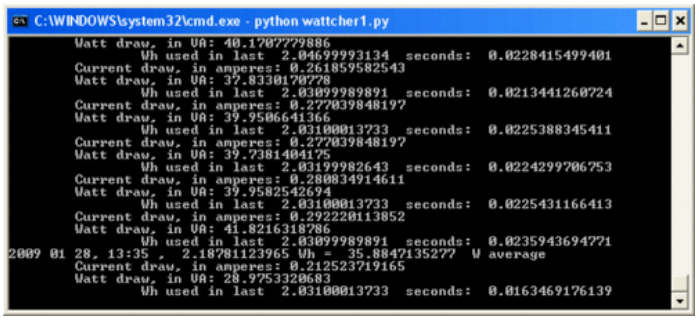
```
def sendreport(sensornum, watt):
# this is where I actually want to go to
serv_uri = target_authenticated_google_app_engine_uri + "?watt="+str(watt)+"&sensornum="+str(sensornum)

serv_args = {}
serv_args['continue'] = serv_uri
serv_args['auth'] = authtoken

full_serv_uri = "http://wattcher.appspot.com/_ah/login?%s" % (urllib.urlencode(serv_args))

serv_req = urllib2.Request(full_serv_uri)
serv_resp = urllib2.urlopen(serv_req)
serv_resp_body = serv_resp.read()

# serv_resp_body should contain the contents of the
# target_authenticated_google_app_engine_uri page - as we will have been
# redirected to that page automatically
#
# to prove this, I'm just gonna print it out
print serv_resp_body
```

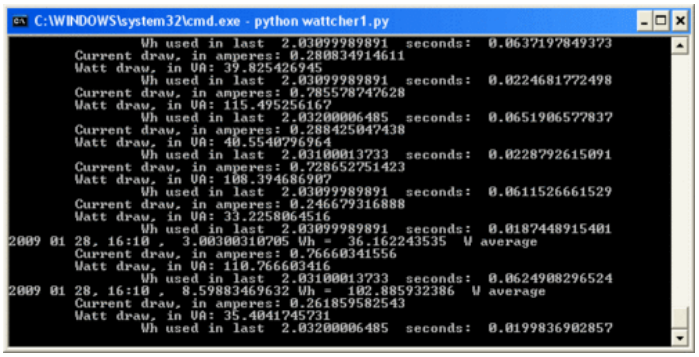
Finally, we wrap up by adding the following lines to our computer script, which will send the data nicely over to GAE!

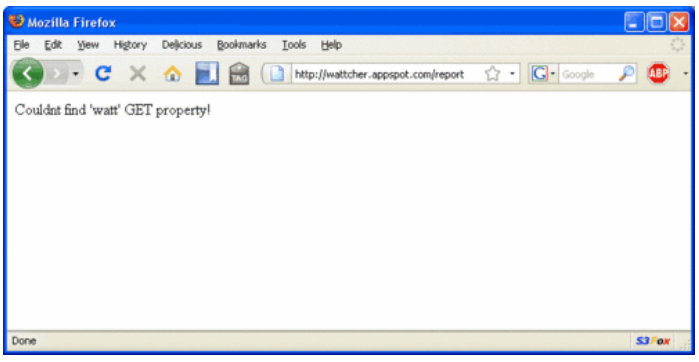
```
# Also, send it to the app engine
appengineauth.sendreport(xb.address_16, avgwattsused)
```

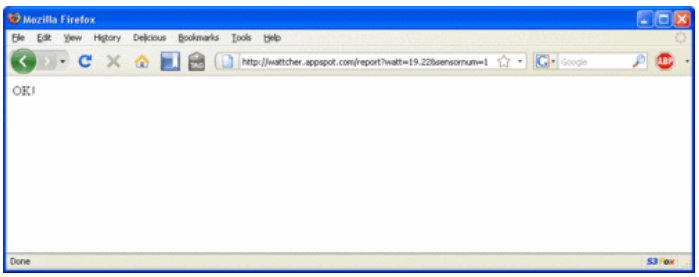
You can download the final script **wattcher.py - final** from the download page !

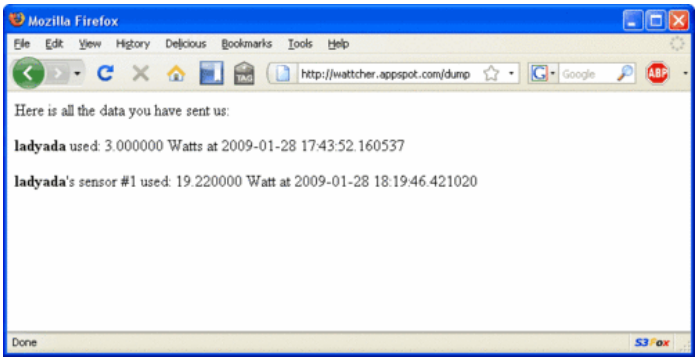
Don't forget to visit **wattcher.appspot.com** to check out the lastest readings

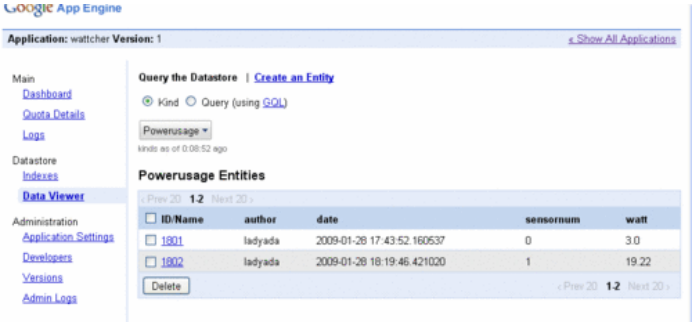


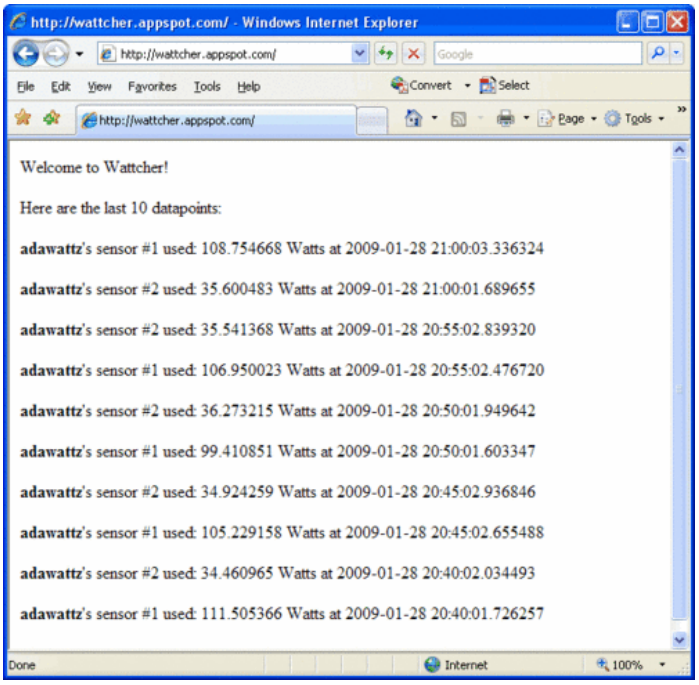












Step 17: Design - graph

Making pretty pictures

Data is great, but visualizations are better. In this step we'll manipulate our stored history so that we can make really nice graphs!

First we'll start by making our sensors named, so that its easier for us to keep track of which is what. Then we'll look at our graph options and data formats. Finally we'll reformat our data to make it ready for graphing

Configuring the sensor names

Its no fun to have data marked as "sensor #1" so I added a 'config' page where the app engine code looks at what sensor numbers have sent data to the database and then allows you to name them. Of course, you need to have the sensor on and sending data -first- before this will work

The configure screen looks something like the image below.

This code uses GET when it should really use POST. I'm kinda old and dont like debugging with POST so...yeah.

```
class Configure(webapp.RequestHandler):
def get(self):
# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

self.response.out.write('<html><body>Set up your sensornode names here:<p>')

# find all the sensors up to #10
sensorset = []
for i in range(10):
c = db.GqlQuery("SELECT * FROM Powerusage WHERE author = :1 and sensornum = :2", users.get_current_user(), i)
if c.get():
sensorset.append(i)
```

```
self.response.out.write('<form action="/config" method="get">')
for sensor in sensorset:
name = ""
currnamequery = db.GqlQuery("SELECT * FROM Sensorname WHERE author = :1 AND sensornum = :2", users.get_current_user(), sensor)
currname = currnamequery.get()

# first see if we're setting it!
if self.request.get('sensornum'+str(sensor)):
name = self.request.get('sensornum'+str(sensor))
if not currname:
currname = Sensorname() # create a new entry
currname.sensornum = sensor
currname.author = users.get_current_user()
currname.sensorname = name
currname.put()
else:
# we're not setting it so fetch current entry
if currname:
name = currname.sensorname

self.response.out.write('Sensor #'+str(sensor)+': <input type="text" name="sensornum'+str(sensor)+'" value="'+name+'"></text><p>')

self.response.out.write("""<div><input type="submit" value="Change names"></div>
</form>
</body>
</html>""")
```

Now we can have more useful data in the history dump

Now we can see that Phil is mostly to blame for our power bill!

Google Visualizer

So we have data and we'd like to see our power usage history. Graphing data is a lot of work, and I'm lazy. So I look online and find that Google -also- has a visualization API! This means I don't have to write a bunch of graphical code, and can just plug into their system. Sweet!

OK checking out the gallery of available visualizations , I'm fond of this one, the Annotated Time Line

Note how you can easily see the graphs, scroll around, zoom in and out and each plot is labeled. Perfect for plotting power data!

Data formatting

Theres a few restrictions to how we get the data to the visualization api and our best option is JSon data. As far as I can tell, JSON is what happened when everyone said "wow, XML is really bulky and wasteful". Anyhow, theres like 4 layers of framework and interpretive data structions and in the end there was a pretty easy to use library written by the Google Visualizations team that let me 'just do it' with a single call by putting the data into a python 'dictionary' in a certain format.

Lets go through the code in sections, since the function is quite long

```
class JSONout(webapp.RequestHandler):
def get(self):

# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

# assume we want 24 hours of data
historytimebegin = 24
if self.request.get('bhours'):
historytimebegin = int(self.request.get('bhours'))

# assume we want data starting from 0 hours ago
historytimeend = 0
if self.request.get('ehours'):
historytimeend = int(self.request.get('ehours'))

# data format for JSON happiness
datastore = []
columnnames = ["date"]
columnset = set(columnnames)
description ={"date": ("datetime", "Date")}

# the names of each sensor, if configured
sensornames = [None] * 10
```

First up we get the user we're going to be looking up the data for. Then we have two variables for defining the amount of data to grab. One is "ehours" (end hours) and the other is "bhours". So if you wanted the last 5 hours, bhours would be 5 and ehours would be 0. If you wanted 5 hours from one day ago, bhours would be 29 and ehours would be 24. datastore is where we will corall all the data. columnnames and description are the 'names' of each column. We always have a date column, then another column for each sensor stream. We also have a seperate array to cache the special sensor names.

onto the next section! Here is where we actually grab data from the database. Now app engine has this annoying restriction, you can only get 1000 points of data at once so what I do is go through it 12 hours at a time. The final datastore has all the points but its easier on the database, I guess. One thing that's confusing perhaps is each column has a name and a description. The name is short, say "watts3" for sensor #3, but the description might be "Limor's workbench". I don't even remember writing this

code so maybe you can figure it out on your own?

```
# we cant grab more than 1000 datapoints, thanks to free-app-engine restriction
# thats about 3 sensors's worth in one day
# so we will restrict to only grabbing 12 hours of data at a time, about 7 sensors worth

while (historytimebegin > historytimeend):
if (historytimebegin - historytimeend) > 12:
timebegin = datetime.timedelta(hours = -historytimebegin)
timeend = datetime.timedelta(hours = -(historytimebegin-12))
historytimebegin -= 12
else:
timebegin = datetime.timedelta(hours = -historytimebegin)
historytimebegin = 0
timeend = datetime.timedelta(hours = -historytimeend)

# grab all the sensor data for that time chunk
powerusages = db.GqlQuery("SELECT * FROM Powerusage WHERE date > :1 AND date < :2 AND author = :3 ORDER BY date", datetime.datetime.now()+timebegin,
datetime.datetime.now()+timeend, account)

# sort them into the proper format and add sensor names from that DB if not done yet
for powerused in powerusages:
coln = "watts" + str(powerused.sensornum)
entry = {"date": powerused.date.replace(tzinfo=utc).astimezone(est), coln: powerused.watt}
if not (coln in columnset):
columnnames.append(coln)
columnset = set(columnnames)
# find the sensor name, if we can
if (len(sensornames) < powerused.sensornum) or (not sensornames[powerused.sensornum]):
currnamequery = db.GqlQuery("SELECT * FROM Sensorname WHERE author = :1 AND sensornum = :2", account, powerused.sensornum)
name = currnamequery.get()

if not name:
sensornames[powerused.sensornum] = "sensor #"+str(powerused.sensornum)
else:
sensornames[powerused.sensornum] = name.sensorname

description[coln] = ("number", sensornames[powerused.sensornum])
#self.response.out.write(sensornames)

# add one entry at a time
datastore.append(entry)
```

Finally at the end of all the looping, we call the magic function that turns the dictionary into JSON, wrap it in the proper Google Visualization package, then spit it out!

```
# OK all the data is ready to go, print it out in JSON format!
data_table = gviz_api.DataTable(description)
data_table.LoadData(datastore)
self.response.headers['Content-Type'] = 'text/plain'
self.response.out.write(data_table.ToJSonResponse(columns_order=(columnnames),
order_by="date"))
```

If you were to visit **http://wattcher.appspot.com/visquery.json?user=adawattz@gmail.com&bhours=1** it would output something like this:


```
google.visualization.Query.setResponse({'version':'0.5', 'reqId':'0', 'status':'OK', 'table': {cols:
[{id:'date',label:'Date',type:'datetime'},{id:'watts1',label:'Limor',type:'number'},{id:'watts5',label:'Workbench',type:'number'},{id:'watts2',label:'Adafruit',type:'number'},{id:'watts
[{c:[{v:new Date(2009,1,25,21,20,2)},{v:64.8332291619},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,20,3)},,{v:230.122099757},,{v:null}]},{c:[{v:new
Date(2009,1,25,21,20,3)},,,{v:65.4923925044},{v:null}]},{c:[{v:new Date(2009,1,25,21,20,4)},,,,{v:48.6947643311}]},{c:[{v:new
Date(2009,1,25,21,25,3)},,{v:228.409810208},,{v:null}]},{c:[{v:new Date(2009,1,25,21,25,3)},{v:67.3574917331},,,{v:null}]},{c:[{v:new
Date(2009,1,25,21,25,3)},,,{v:66.0046383897},{v:null}]},{c:[{v:new Date(2009,1,25,21,25,4)},,,,{v:47.3892235642}]},{c:[{v:new
Date(2009,1,25,21,30,2)},{v:84.9379517795},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,30,3)},,,,{v:99.7553490071}]},{c:[{v:new
Date(2009,1,25,21,30,5)},,{v:229.73642288},,{v:null}]},{c:[{v:new Date(2009,1,25,21,30,6)},,,,{v:66.6556291818},{v:null}]},{c:[{v:new
Date(2009,1,25,21,35,2)},,,{v:67.3146052998},{v:null}]},{c:[{v:new Date(2009,1,25,21,35,3)},{v:96.2322216676},,,{v:null}]},{c:[{v:new
Date(2009,1,25,21,35,3)},,{v:226.678267688},,{v:null}]},{c:[{v:new Date(2009,1,25,21,35,4)},,,,{v:158.428422765}]},{c:[{v:new
Date(2009,1,25,21,40,3)},,{v:232.644574879},,{v:null}]},{c:[{v:new Date(2009,1,25,21,40,4)},,,,{v:153.666193493}]},{c:[{v:new
Date(2009,1,25,21,40,6)},,,{v:66.7874343225},{v:null}]},{c:[{v:new Date(2009,1,25,21,40,12)},{v:95.0019590395},,,{v:null}]},{c:[{v:new
Date(2009,1,25,21,40,21)},{v:95.0144043571},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,40,23)},,,{v:66.8060307611},{v:null}]},{c:[{v:new
Date(2009,1,25,21,45,2)},,,{v:66.9814723201},{v:null}]},{c:[{v:new Date(2009,1,25,21,45,3)},,{v:226.036818816},,{v:null}]},{c:[{v:new
Date(2009,1,25,21,45,3)},{v:99.2775581827},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,45,4)},,,,{v:154.261889366}]},{c:[{v:new
Date(2009,1,25,21,50,4)},{v:102.104642018},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,50,4)},,,,{v:155.441084531}]},{c:[{v:new
Date(2009,1,25,21,50,5)},,,{v:67.0087146687},{v:null}]},{c:[{v:new Date(2009,1,25,21,50,5)},,{v:230.678636915},,{v:null}]},{c:[{v:new
Date(2009,1,25,21,55,3)},{v:103.493297176},,,{v:null}]},{c:[{v:new Date(2009,1,25,21,55,3)},,,,{v:151.309223916}]},{c:[{v:new
Date(2009,1,25,21,55,4)},,,{v:66.9174858741},{v:null}]},{c:[{v:new Date(2009,1,25,21,55,4)},,{v:227.765325835},,{v:null}]},{c:[{v:new
Date(2009,1,25,22,0,3)},,,{v:67.0004310254},{v:null}]},{c:[{v:new Date(2009,1,25,22,0,3)},,,,{v:150.389989112}]},{c:[{v:new
Date(2009,1,25,22,0,3)},,{v:230.892049553},,{v:null}]},{c:[{v:new Date(2009,1,25,22,0,4)},{v:92.2432771363},,,{v:null}]},{c:[{v:new
Date(2009,1,25,22,15,3)},{v:97.5910440774},,,{v:null}]},{c:[{v:new Date(2009,1,25,22,15,3)},,,,{v:143.722595861}]},{c:[{v:new
Date(2009,1,25,22,15,4)},,,{v:64.4898008851},{v:null}]},{c:[{v:new Date(2009,1,25,22,15,4)},,{v:222.357617868},,{v:null}]}]}});
```


Anyways, you can kinda see the data, also note its actually a function call, this stuff is really kinky!

Now go to the Google Visualizations Playground and enter in that URL into the sandbox

And you can see the visualization itself pop out! (this is just a screen shot so go do it yerself if you want to mess around)

OK go mess around, adding and changing **bhours** and **ehours**

Wrapping up the visualization

OK we're nearly done. Now we just need to basically grab the code from the sandbox and make it a subpage in our app engine...like so:

```
class Visualize(webapp.RequestHandler):
def get(self):

# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

historytimebegin = 24 # assume 24 hours
if self.request.get('bhours'):
historytimebegin = int(self.request.get('bhours'))

historytimeend = 0 # assume 0 hours ago
if self.request.get('ehours'):
historytimeend = int(self.request.get('ehours'))

# get the first part, headers, out
self.response.out.write(
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>Google Visualization API Sample</title>
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load("visualization", "1", {packages: ["annotatedtimeline"]});

function drawVisualizations() {
)

# create our visualization
self.response.out.write(new google.visualization.Query("http://wattcher.appspot.com/visquery.json?user= +
account.email()+&bhours= +str(historytimebegin)+").send(
function(response) {
new google.visualization.AnnotatedTimeLine(
document.getElementById("visualization")).
draw(response.getDataTable(), {"displayAnnotations": true});
});
)

self.response.out.write(}

google.setOnLoadCallback(drawVisualizations);
</script>
</head>
<body style="font-family: Arial;border: 0 none;">
<div id="visualization" style="width: 800px; height: 250px;"></div>
</body>
</html>)
```

The first part is pretty straight forward, get the user name or login. Then we will assume the user wants 1 last day of data, so set **bhours** and **ehours** . Then we literally just print out the code we copied from Google's Visualization sandbox, done!

Viz Viz Viz

The only thing I couldn't figure out is how to get 3 visualizations going on at once (last hour, day and week) with the above code. It just kinda broke. So for the triple view I had to use iframes :(

```
class VisualizeAll(webapp.RequestHandler):
def get(self):

# make the user log in if no user name is supplied
if self.request.get('user'):
account = users.User(self.request.get('user'))
else:
if not users.get_current_user():
self.redirect(users.create_login_url(self.request.uri))
account = users.get_current_user()

self.response.out.write(
<h2>Power usage over the last hour:</h2>
<iframe src ="graph?user=adawattz@gmail.com&bhours=1" frameborder="0" width="100%" height="300px">
<p>Your browser does not support iframes.</p>
</iframe>

<h2>Power usage over the last day:</h2>
<iframe src ="graph?user=adawattz@gmail.com&bhours=24" frameborder="0" width="100%" height="300px">
<p>Your browser does not support iframes.</p>
</iframe>
```

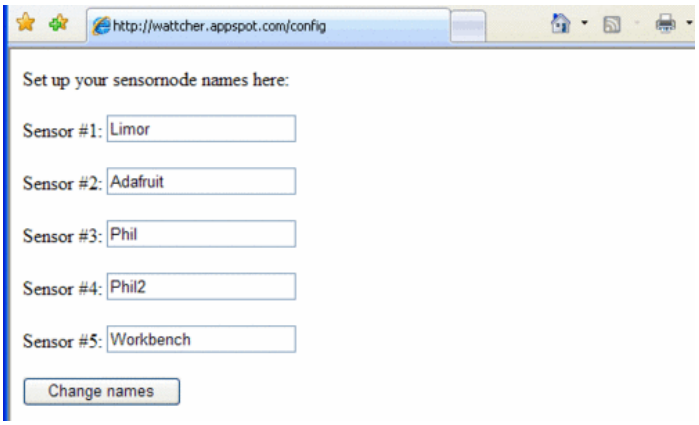
```
<h2>Power usage over the last week:</h2>
<iframe src ="graph?user=adawattz@gmail.com&bhours=168" frameborder="0" width="300%" height="500px">
<p>Your browser does not support iframes.</p>
</iframe>
```

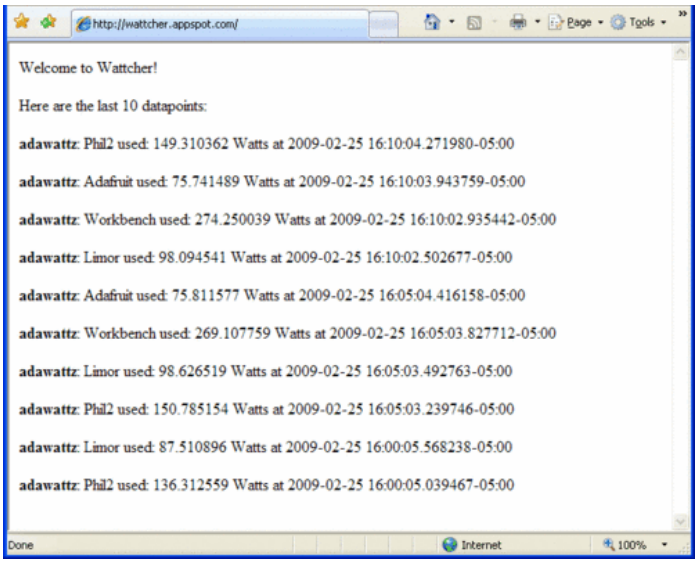
)

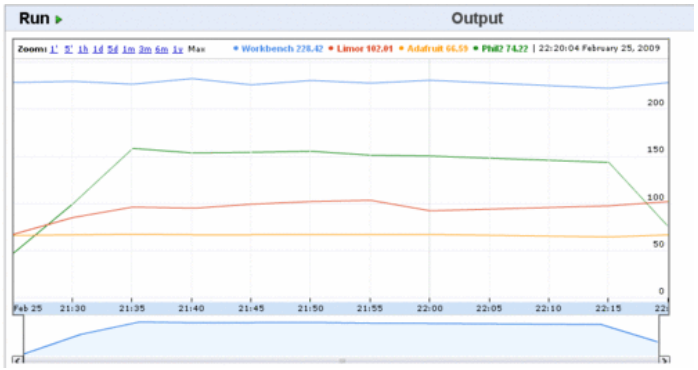
Anyhow, it works just fine.

Timecodes!

The final thing that wont be reviewed here is how I got the date and times to be EST instead of UTC. As far as I can tell, its kind of broken and mysterious. Check the code if you want to figure it out.







Step 18: Resources

Other power monitoring projects!

Get some good ideas here!

- "Carbon Penance" a power monitor by Annina Rust that punishes the user
- Jason Winter's Real-Time power monitor
- Mazzini's project pushes data onto Pachube
- Pachube has lots of other projects!
- Furnace monitoring, using a DAQ board and phototransistor

Power monitoring products
Wanna just buy it?

- Black & Decker home electric meter watcher
- DIY KYOTO's power-clip Wattson is pretty
- CurrentCost

Websites & Software

- Myenergyusage.org (one fellow upgrading his Wattson's software by hand)
- Wattzon.org (social energy information)

Step 19: Download

Software

- Zip file of the most recent python scripts this is what you want if you've built a tweet-a-watt and you want to get your project running

X-CTU profiles

- Receiver, connected to computer
- Transmitter, embedded in the Kill-a-Watt. Change the unique ID if you have more than one!

Design files

All this stuff (other than the XBee library and the AppEngineAuth library, which are not written by me) is for you in the Public Domain! These are for debugging and design purpose and show how the project is put together. If you just want to "run the code" see the "software" above

- XBee library
- AppEngineAuth library for python
- Wattcher.py - graph just the mains data
- Wattcher.py - graph mains and wattage data
- Wattcher.py - Reports averages every 5 minutes
- Wattcher.py - Sends data to Wattcher Google App

For the latest code, please check the google code repository where you'll find any new code . And hey, are you good at this sort of code? I could really use some help. It works OK but it could look and run much better so please commit patches and hang out on the forum!

Related Instructables

Getting started with the Maxbotix sonar sensor - quick start guide by adafruit

Proto Shield by adafruit

Tilt Sensor Tutorial by adafruit

Musical MIDI Shoes by thobson

Self Destructing Message by foxxtrotalpha

Hacking an Arduino ISP Shield for AtTiny45 / AtTiny85 by rleyland

Make your pet dishes tweet! by thoughtfix

The Best Arduino by msuzuki777

Bubblesteen Bubble Machine

by **belliedroot** on April 27, 2010

Author:belliedroot Bernard Katz Glass

I am a glass sculptor with a shop and gallery located in the Manayunk section of Philadelphia. Besides being a dad and running my business, I have a strong interest in electronics, and physical computing.

Intro: Bubblesteen Bubble Machine

Is it a 3D Spherical Atmosphere Encapsulated Phosphorous Printer? **YES!**

Is it a CNC Anti Gravity transparent Orb Machine! **YES!**

Its **The Bubblesteen Bubble Machine!** The spherical miracle that kids and cats have been waiting for. It comes complete with robotic edge detection(when a bubble hits an edge it pops, thus the edge has been detected).

Turn up the sound and watch Lester the cat battle it out with the Bubblesteen!

* No animals were harmed during testing.

This project came about after playing around with pan and tilt using servos. Most of the pan and tilt senereos I saw involved using webcams or some type of camera, which pan and tilt is perfect for. There are some good instructables and how-to's on the web for this very thing.

I may not have an available camera, but I did have some bubble mix :)

http://bernardkatz.com/

Step 1: Things you will need

This list is mostly for the electronic and mechanical stuff. How you create your own Bubblesteen will depend on your creativity and what you have laying around.

I will also include small tips on the materials I used and things to be careful of

1. 1. Arduino Duemilanove $30.00
2. 1 motor shield $19.50 (www.adafruit.com) * It is made to fit the Duemilanove
3. 2 micro servos- I used Hextronik HXT 500 $3.50 each (www.hobbyking.com)
4. 1 DC toy motor- something between 3v and 12v - easy to find, motor shield docs will help you decide if what you may already have will work.
5. 1 thing of bubble mix. - find at CVS or a dollar store. Some work better than others

These things I used, but are not critical. This is where your own creativity will need to come to play.

1. 1 roll of perferated metal tape- any hardware store
2. nuts and bolts of various sizes - thread count not critical :)
3. diamond plate- local scrap yard
4. aluminum channel- local scrap yard
5. 1 threaded rod hanger/ plate
6. earthquake putty or museum wax

7. 5 minute epoxy
8. 1/4" acrylic sheet- about 6" x 6" worth
9. acrylic adhesive

The tools you need will once again depend on what you build yours out of.

Image Notes
1. This stuff is almost as good as duct tape when it comes to versitility

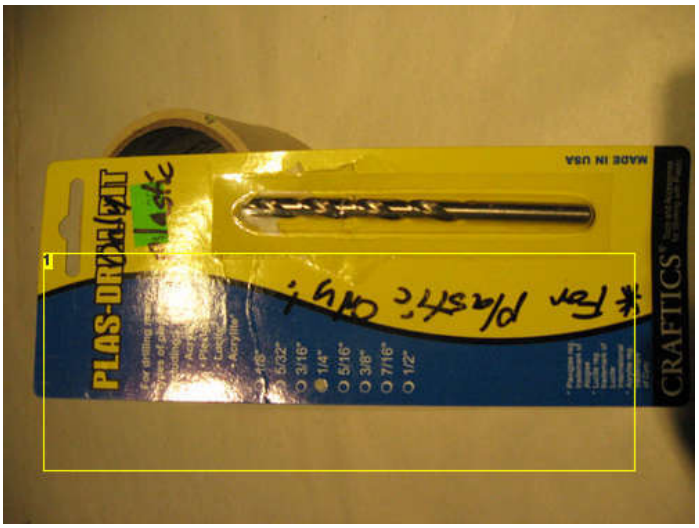


Image Notes
1. I highly recommend using special drill bits that are made for drilling plastic. Less chance of cracking the plastic. Can find on the web or local plastics supplier

Step 2: Dealing with the micro controller

I am going to assume that you have some experience with the Arduino. If not, you can check out either the web or this site for starting with arduino.

I will say that Ladyada's site www.adafruit.com is great for tutorials and buying arduino related stuff. In fact, you should refer to her site about using the Adafruit motor shield anyway.

The instructions on using the motor shield will tell you where to hook in the servo and DC motor, so I will not go into those details.

The code I used is posted below.

It is not the most elegant and for the most part hobbled together, but it works. Make sure you have the library for the motor shield

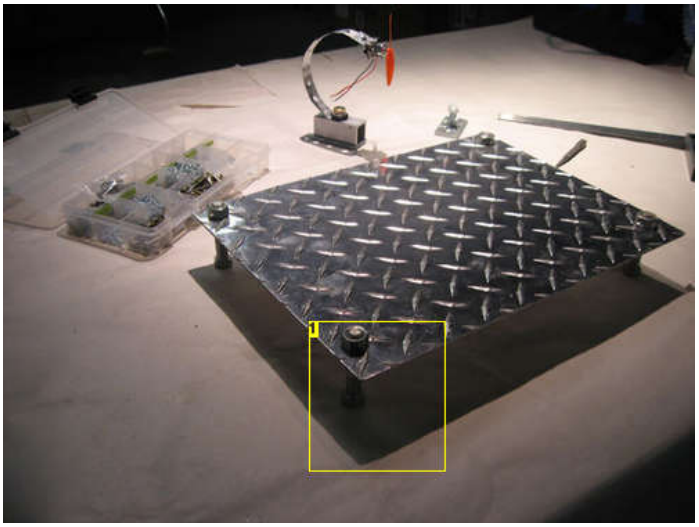
File Downloads

bubble_servo_DCMotor_april24b.pde (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'bubble_servo_DCMotor_april24b.pde']

Step 3: Putting it together

I had some diamond plate at the shop, so this became the platform.

Tip # 1 - The reason the arduino will be mounted below the platform is so it won't get **wet.** I am pretty positive a wet microcontroller doesn't work very well. Stuff will spash around!



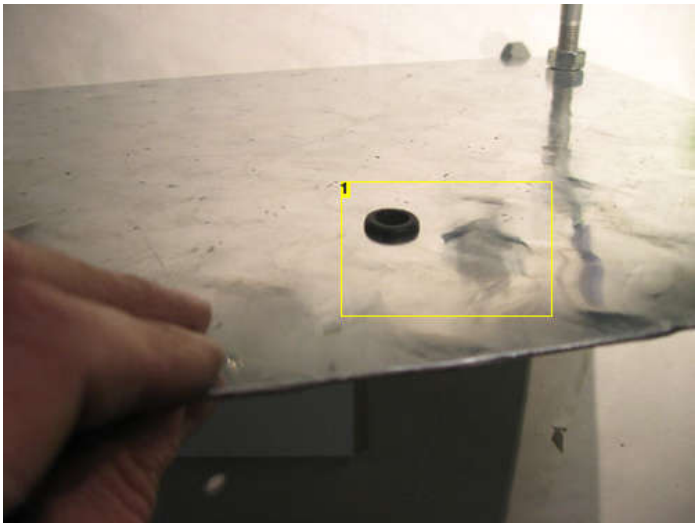
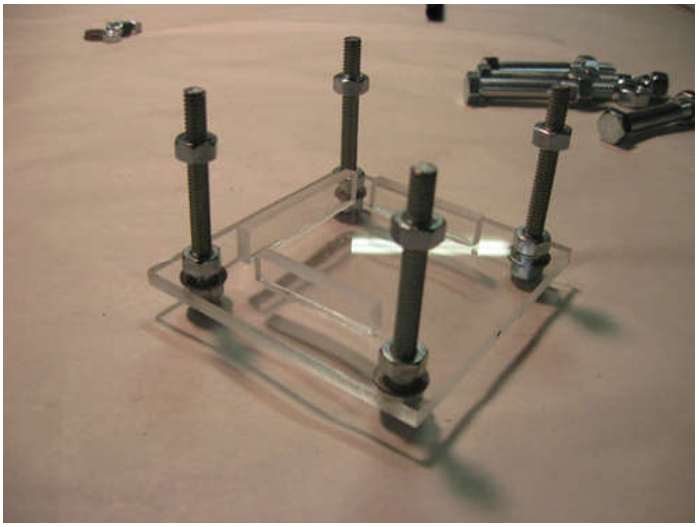


Image Notes
1. Bolts to raise the platform

Image Notes
1. gromet to protect servo and DC motor wires and to keep liquid from leaking to micro.

Step 4: Arduino & motor shield platform

These are just photos showing the construction of the microcontroller platform. I decided to have an on-off switch and a power jack.



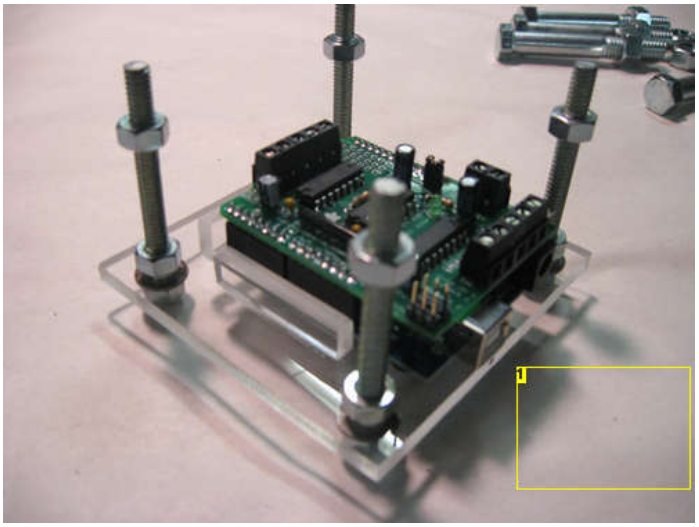


Image Notes
1. Arduino with motor shield

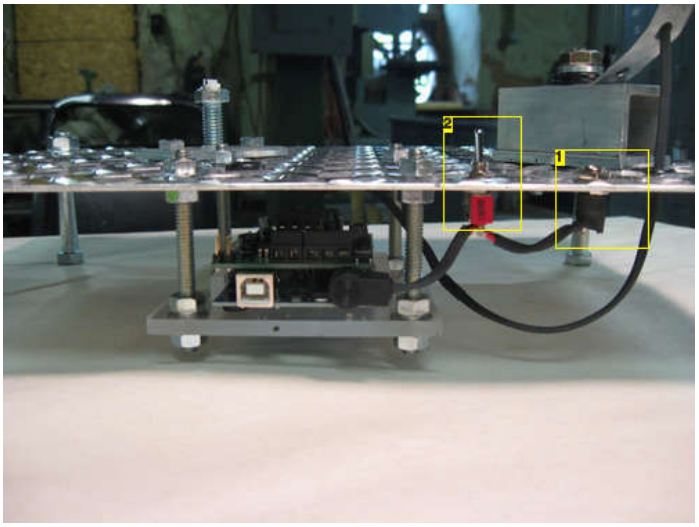


Image Notes
1. power jack
2. switch

Step 5:

Tip # 2 - I used museum wax to hold the servos together. It is fairly strong, but removable in case you need the servos for something else.

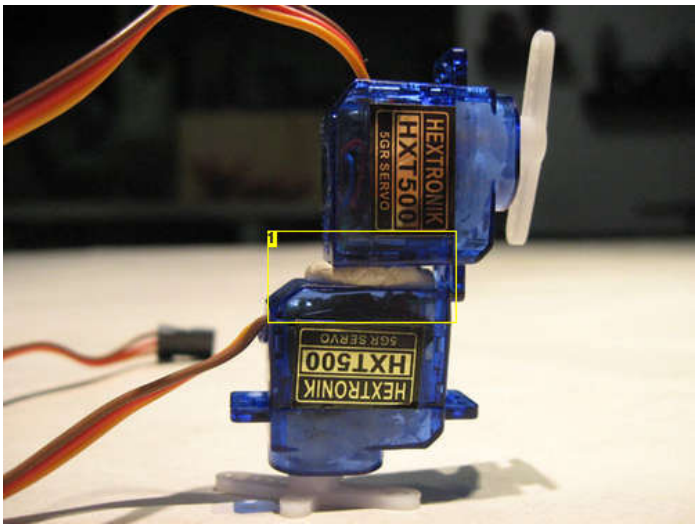


Image Notes
1. museum wax or poster putty

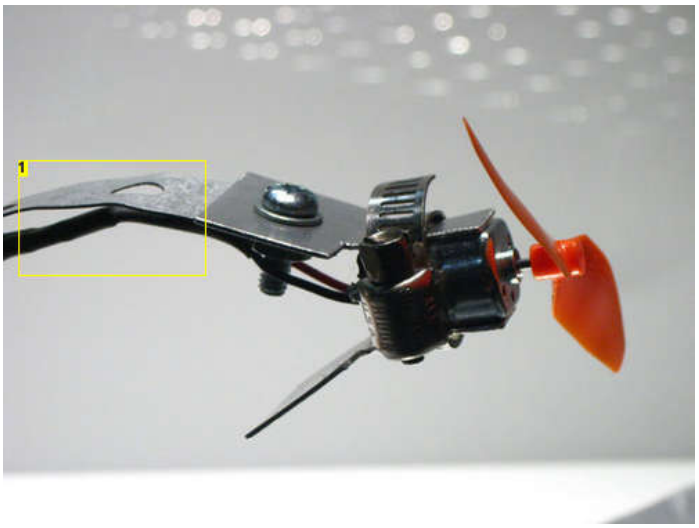


Image Notes
1. metal tape. easy to adjust height

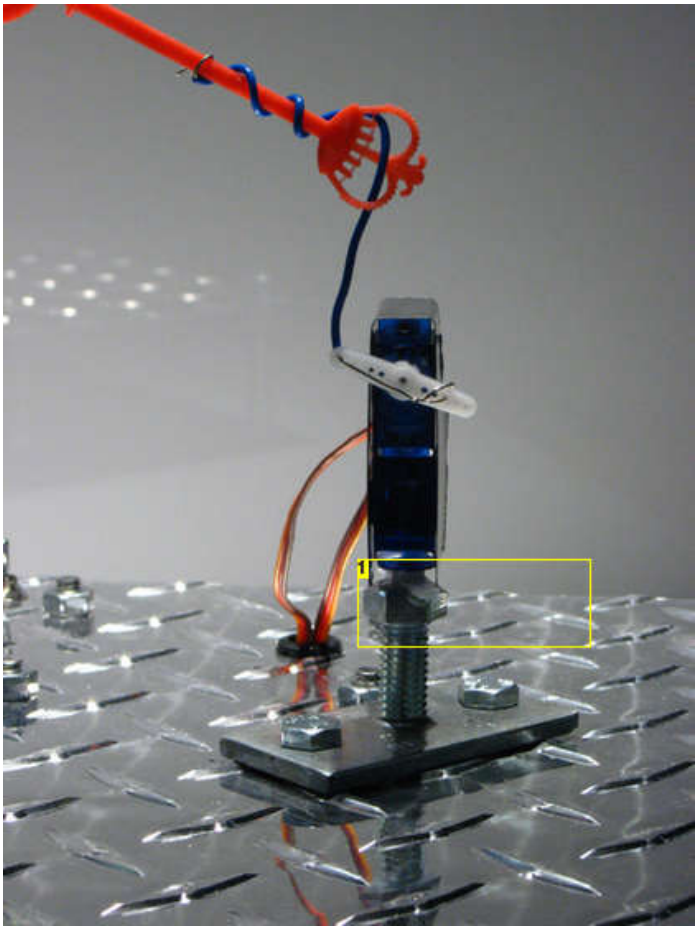
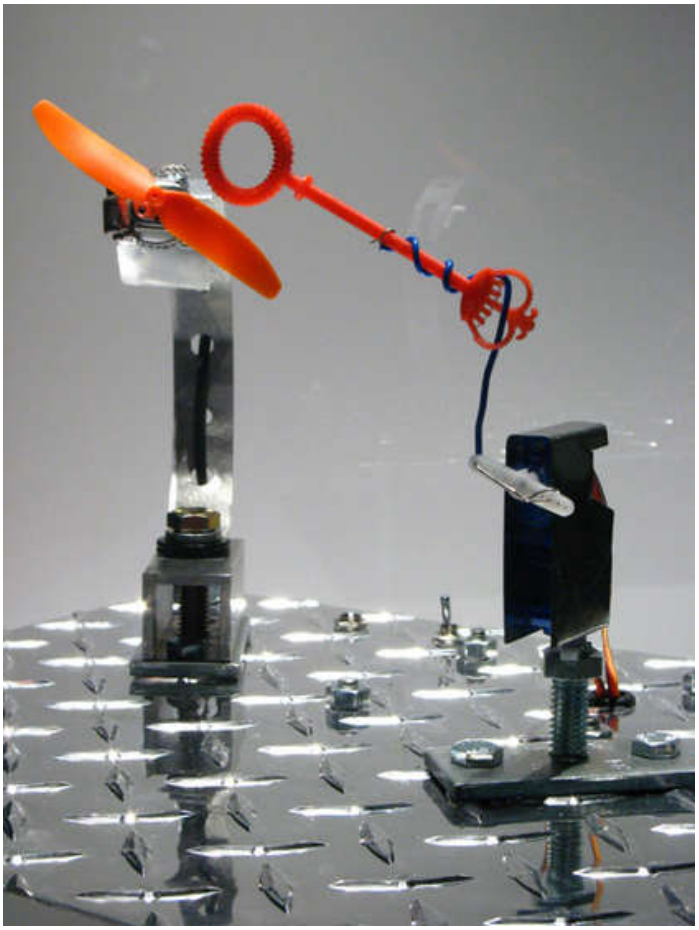


Image Notes
1. epoxy the servo horn to bolt



Step 6: Additional photos

That is about it. Here are some additional photos.

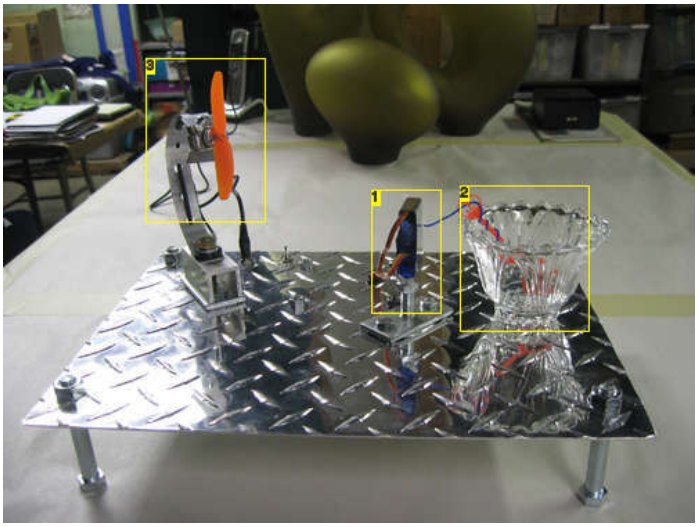
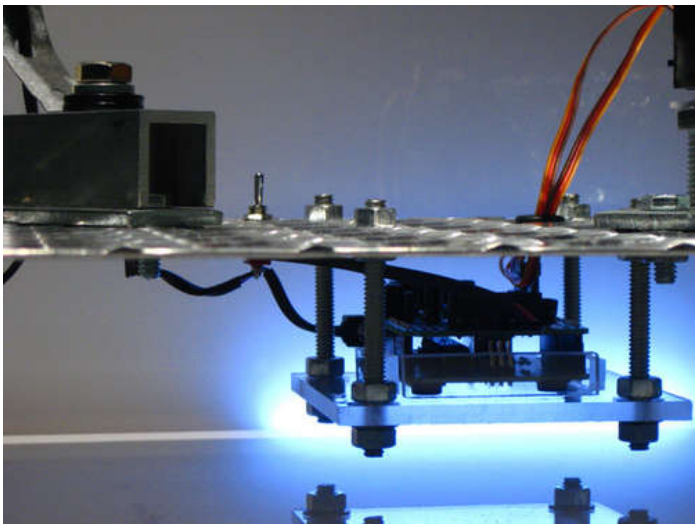


Image Notes

1. servos
2. bubble dipper and glass cup for bubble mix
3. small DC motor with propeller



Related Instructables

Arduino 4wd robot with ping sensor "J-Bot" by JamecoElectronics

Robotic Arm with Servo Motors by dustynrobots

Arduino-based line follower robot using Pololu QTR-8RC line sensor by techbitar

JabberBot! The Arduino robot with an ATMega brain and bluetooth braun! :-) (Photos) by declanshanaghy

My Arduino Ping Display Robot by Cello62

Bubblebot: Gigantic Bubble Generator by zvizvi

Robotic Talking Turret by RazorConcepts

Control an OWI Robotic Arm with Arduino by zlite

Arduino R/C Lawnmower (painted)

by **johndavid400** on May 19, 2009

Author:johndavid400 author's website

I have always been one to take things apart to figure out how they work, so most of what I own has been dismantled. If it can't be taken apart or hacked, i'd rather not have it. And I like to do things the cheapest way possible, because I like to do a lot of things and I don't have a lot of money.

Intro: Arduino R/C Lawnmower (painted)

What this is:

This instructable will show you how to make your Arduino into an R/C interface that you can use for just about anything requiring remote control. I will also show you how I built an R/C lawnmower using my Arduino, a cheap R/C transmitter and receiver pair, and a couple of electric-wheelchair motors from Ebay. I have used this interface to control anything from basic LED's to Bipolar stepper motors, mini-robots, lifeless R/C cars from the thrift store, and even a 100lb lawnmower (all with appropriate motor controllers). It is very flexible and easy to change and very simple to set up.

See a slightly different version of the Lawnbot400 in my new book "Arduino Robotics" , as well as a DIY Segway and several other bots.

Check it out in MAKE magazine in the April 2010 issue (#22) or here:

UPDATE 3-24-10

New wheel-barrow bucket mounted on top with hinges so it can dump its contents.

UPDATE 3-10-10: NEW CODE

And new video of the Lawnbot400 moving a bunch of dirt from my truck to the flower beds across the yard, also I updated the code again.

.

I added some new code to the project that is safer, including a manual kill-switch and a Failsafe switch.

To implement the Failsafe, I used another Atmega168 (or an Arduino), to control a normally-open 60amp power relay. The relay disconnects the power to the motor-controller unless receiving a "good" signal from the 2nd microcontroller. This signal is updated 2 times every second and is either ON or OFF. If the bot gets out of range, it loses power to the motors. If I flip the kill-switch on the Transmitter, it loses power to the motors. This is also a handy way to disable it remotely if anything were to go near it that wasn't supposed to. The updated code for both microcontrollers is on the CODE page.

In addition to the failsafe, I changed the way the code reads the PPM signals to make it more reliable. Also, I realized that I was only able to run the bot at 80% speed with the old code, so now it is quite a bit faster and has more power (it can carry me across the yard @ 155lb).

Check out this new video of me riding the Lawnbot400, my wife driving it over a bunch of branches, then me making do some wheelies. Don't worry, the mower was turned off this time since the grass didn't need cutting, we were just having fun.

Disclaimer:
DANGER!!! This is a VERY dangerous piece of equipment if not handled appropriately. Since all the electronics have been home-built and the Arduino code is new, you MUST be very careful while operating anything heavy with this code. I have had 1 or 2 times during testing - and before adding a secondary failsafe - that the main Arduino jammed up and I temporarily lost control of the mower for a few seconds!!!! Though I have added several filters to discard unwanted signals and I rarely have any issues, an un-manned lawnmower IS STILL A POTENTIAL DEATH TRAP and I assume no responsibility for anything that happens as a result of your use of this code or this tutorial. This is meant as a guide for people who not only have the ability to build such a contraption, but the responsibiltity to operate it safely as well. Any suggestions or ideas on how to make this a safer project is always gladly accepted. Having said that, it's also awesome.

Background:

Most R/C equipment comes packaged for a single specific use, which makes it easy to use but is very limited in what you can do with it. So using the Arduino as an interpreter between the R/C system and the motor driver, I can use any motor controller that I want (depending on the size of the motor and power required), reprogramming the Arduino to supply the required signals.

What I ended up with:

After successfully hacking a few R/C cars from the thrift store, I got bored driving them around the driveway and I was having a hard time convincing my wife that there was any usefulness in the revived toy car. So I decided it was time to make my biggest chore at home, a whole lot easier and actually put my Arduino to work, and thats how I ended up building an R/C lawnmower.

While designing the lawnmower, I thought it would be cool to learn about the electronics that made it move, so I designed and built my own motor speed controller (or H-bridge) to power the lawnmower. I looked around at every H-bridge design I could find before deciding to go with a Mosfet h-bridge that uses both N-channel and P-channel Mosfets.

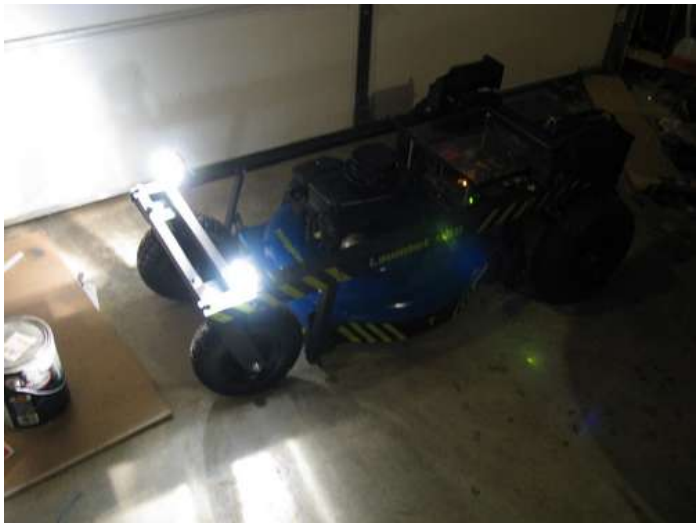
I built several different motor driver boards for this project, the first two were on Radio-Shack perf-board and the next 4 were designed using EagleCad and etched to a piece of copper-clad PCB, using the toner-transfer method. The most recent board is the one I use to mow the lawn as it has the ability to stay cool even while operating for long periods of time (30-40 mins straight) at 10-20amps and 24vdc. FWIW, I had to burn up a lot of Mosfets to find this out. If you want to see any of my other motor controllers, go to www.rediculouslygoodlooking.com and check out the Mosfet shield.

Here is what I bought already assembled:
FM R/C transmitter and receiver pair from ebay = $40
Arduino = $30
I already had a used push-mower = $60

Here is what I bought and assembled into the Lawnbot400 (as I call it):
(2) electric-wheelchair motors from ebay = $40 ea
(2) 12v marine deep cycle batteries - Walmart - $60 ea new (used batteries might work)
36" pieces of 2" angle-iron (2) and 1" square-tubing (2) from Home Depot = $8 ea
36" pieces of 1" angle-iron (2) and 1" flat steel bar (2) from Home Depot = $5 ea
(a lot) of nuts, bolts, washers, lock washers 3/8" or 1/2" with drill bit = $20
(2) caster wheels from Harbor Freight Tools = $14 ea
(2) drive wheels from Harbor Freight Tools = $8 ea
(36") 5/8" threaded rod with several 5/8" nuts and washers from Home Depot = $8
(2) sprockets from Allelectronics = $5 ea
#25 roller chain and a few universal links from Allelectronics = $10 for 3'
sprockets from Electronics Goldmine = $1.50 ea
(24) mosfets from Digikey = $1 ea
(there were quite a few small parts for building the H-bridge, they are listed later on)


Lawnbot 400


Lawnbot 400




Lawnbot 400

Image Notes
1. the front left mower deck hanger
2. the rear left mower deck hanger

Image Notes
1. the Triple8 motor controller with 24 mosfets, each set of 3 is bolted together and each mosfet is heatsinked. It has 3x as many Mosfets as it's little brother, but essentially the same circuit.
2. the predecessor to the Triple8, only 8 mosfets total (just enough to complete a dual h-bridge). Though it would run the Lawnbot400 around for about 10 minutes, it would end up getting hot after some use.

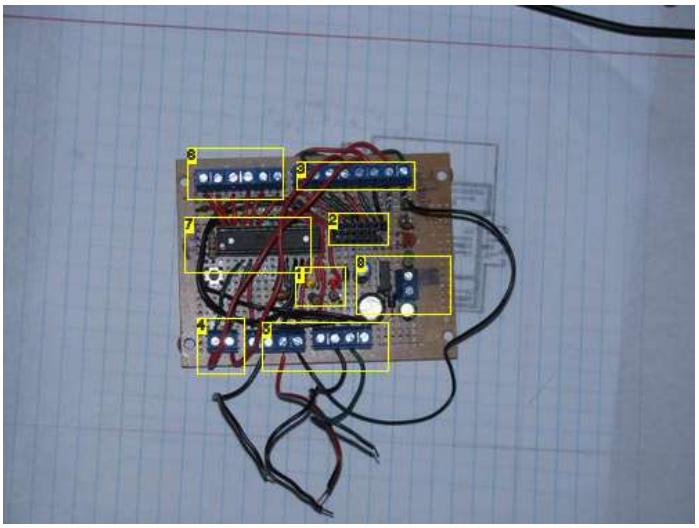


Image Notes
1. the 2 neutral indicator LED's (1 red and 1 yellow) hard-wired to digital pins 12 and 13. Anytime I center one of the control sticks on the lawnbot400, one of these lights turns on.
2. the female headers used to plug my R/C receiver directly onto (they use standard .1" spacing like perfboard you can buy at Radio Shack)
3. these are the breakout screw-terminals used to route the R/C receiver signals to the Atmega168. I am only using 2 of the 6 R/C channels right now, so the other 4 can be used for extra servo's or whatever else.
4. digital pins 2 and 3 of the Atmega168, used for the External Interrupts on those pins to capture the R/C signals from the receiver.
5. These are the screw-terminals for the signal wires leading to the H-bridge motor controller. I only need 4 wires to run my motor controller, but there are 3 extra digital pins that are unused by the current code.... Any ideas for their use?
6. all 6 analog pins are unused! I might add some sensors to automate the Lawnbot400 one day.
7. the Atmega168, it's reset button, and a kind-of hidden 16mHz crystal oscillator (together make a bare-bones Arduino).
8. 5-35v power terminal and onboard 5v regulator for powering the Atmega and R/C receiver. Plus a bunch of capacitors and a reverse polarity protection diode.

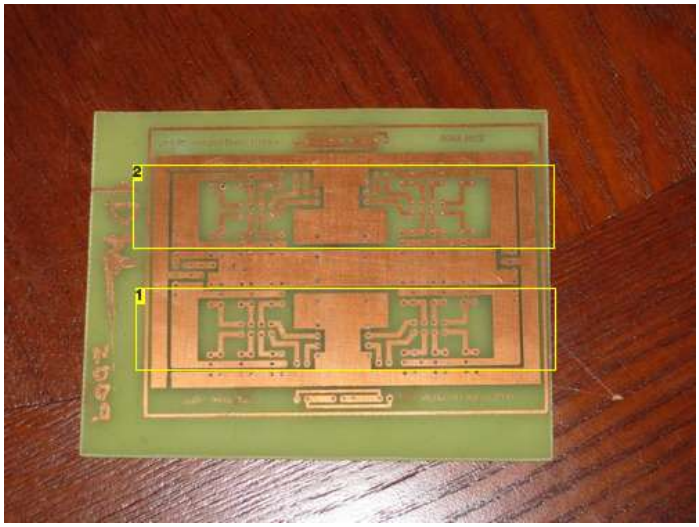


Image Notes
1. this is the 2nd H-bridge, notice that the motor screw-terminals for each motor will be on opposite sides of the board.
2. This is the 1st H-bridge

Step 1: Setting up

1. Get R/C transmitter and receiver (I have tested FM and AM systems and they both work)
2. Upload code to Arduino (it is on the last page)
3. Make sure you are getting a good signal

You will need an R/C radio transmitter(Tx) and receiver(Rx) pair, which is the most expensive part of the project, but can be used for every future project you might have involving R/C. I went with a 6-channel FM system, but I have tested a 27mHz AM transmitter/receiver and it works just as well. The beauty of the Arduino is that if you want to adjust the deadband or the motor-speed at turn-on, (unlike commercial ESC's) it is all easy changed in the Arduino IDE.

Once you have your radio, all you need to do is upload the code to your Arduino, plug in the 2 channels that you want to use from your radio receiver into Digital pins 2 and 3 of the Arduino (these are the 2 external interrupt pins on the Arduino) and you are ready to control whatever you want. If you don't have a batter pack for the receiver, you can run jumper wires from the Arduino +5v and GND to the R/C receiever for power, you only need to supply a single channel with GND and +5v (it is not necessary to power every channel).

Upload the code using the Aruino IDE (I am using version 0016 on Ubuntu).

I started by controlling 3 LED's with 1 channel on a breadboard. I wired a red LED to be Forward (digital pin 9), a yellow LED for Reverse(digital pin 5), and a green LED for Neutral (digital pin 12). This allows you to adjust the code to fit the needs of your radio system. You will have smooth 0-100% PWM control of both LED's and the neutral light will turn on when the control stick is centered. If needed, you can widen the deadband for Neutral, but doing so will increase the speed at turn-on (which starts at 0%, so that would likely be desirable). See pictures.

The code has 4 PWM outputs for motor control:

channel 1 Forward = Arduino digital pin 9
channel 1 Reverse = Arduino digital pin 5
channel 2 Forward = Arduino digital pin 10
channel 2 Reverse = Arduino digital pin 6

2 outputs for Neutral indicator lights:

channel 1 = digital pin 12
channel 2 = digital pin 13

The 2 INPUTS from the R/C receiver should go to:

channel 1 = digital pin 2
channel 2 = digital pin 3

If you are interested to see your readings, turn on your Serial Monitor in the Arduino IDE (set to 9600bps) and you can see the actual real-time pulse readings for each channel, they should read:

full forward = 2000 (2 milliseconds)
center = 1500 (1.5 ms)
full reverse = 1000 (1 ms)

These readings reflect the number of microseconds that the pulse signal from the R/C receiver stays HIGH (or at 5v). The typical Servo signal that comes from an R/C receiver is a pulse whose length varies from approximately 1 ms to 2 ms with 1.5 ms being Neutral (which should also be the position that the control stick returns to when you let it go). The transmitter reads the position of the control stick and sends that pulse length about once every 20milliseconds. So it is constantly updating for precise control (for more info, look up PPM on wikipedia). If you push the transmitter control stick forward, the reading should go up to 2000, if you push it backward it should go down to 1000. You can also use a voltage meter at this point to see that Digital Pins 5, 6, 9, & 10 will be changing from 0-5v depending on the position of the control sticks on the R/C transmitter.

If you care to know, the code uses the Arduino's 2 external interrupts to capture when the Rx signal pin changes states (goes from HIGH to LOW or vice versa), when it does at the beginning of each signal, it calls the interrupt function which reads the digital state of the pin and if HIGH, it records the microseconds value on the Arduino system timer0. It then returns to the loop until the pin goes LOW, at which point it subtracts the previously recorded microsecond value from the new current microsecond value to determine how long the pulse stayed HIGH (which tells us the position of the Transmitter control stick). It then does that over and over really fast.

I have the values constrained from 600-2400 in the Arduino code to keep things simple. Once it receives the signal and constrains it, it maps that value to be proportionally between 0 and 511, where 255 will be Neutral. The code then determines when the value changes and uses a function to determine the appropriate 0-255 PWM value in the appropriate direction and each direction has it's own PWM output pin to control the H-bridge.

On a side note:

To make things easier, I built an Arduino-based breakout board using Radio-Shack perf-board, a 28pin DIP socket, a 16mhz oscillator, and a bit of wire. I also added a set of female-headers in such a way that I can plug my R/C receiver directly onto the breakout board. For secure connections while mowing grass, I added screw-terminals on each Output pin and each of the 6 channels from the receiver. It also has a built in 5v regulator to power both the Atmega168 from the Arduino and the R/C receiver (which gets power when you plug it onto the breakout board). So you just route jumper wires from the channels you want to use on the receiver, to the Atmega digital pins 2 and 3. I also added 2 LED lights that are hard wired to the digital pins 12 and 13 for the Neutral lights for each channel so I can easily see when I am in neutral.

Since this bot is a Tank steer setup with 1 drive motor on each wheel, the coding is very straightforward where the left stick controls the left motor and the right stick controls the right motor. Both sticks forward means lawnmower goes straight forward, both backward and it goes in reverse. If you push the left forward and the right backward, it does a zero-turn circle. As you can imagine, mowing the grass is really fun now.

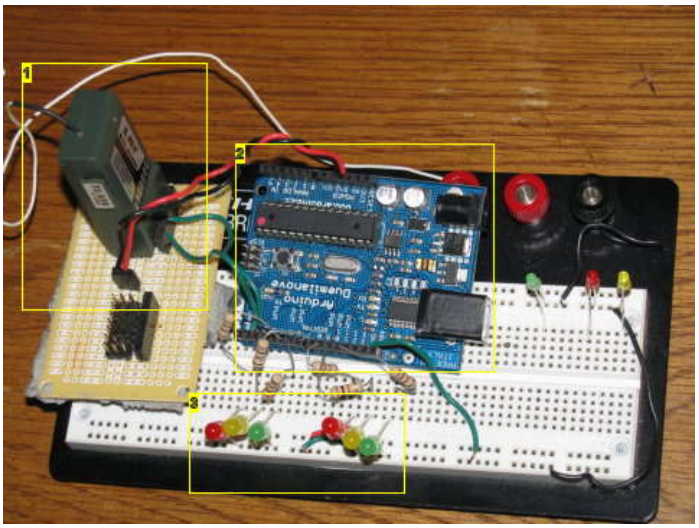


Image Notes
1. this is my receiver plugged into a breakout board I made for it using perfboard.
2. the Arduino receiving R/C servo signals and translating them into forward/reverse PWM values.
3. each set of LED's is controlled by it's own channel from the R/C receiver. Forward will turn on the green light, reverse the Red light, and neutral will light up the Yellow light. This is the easiest way to test the setup.

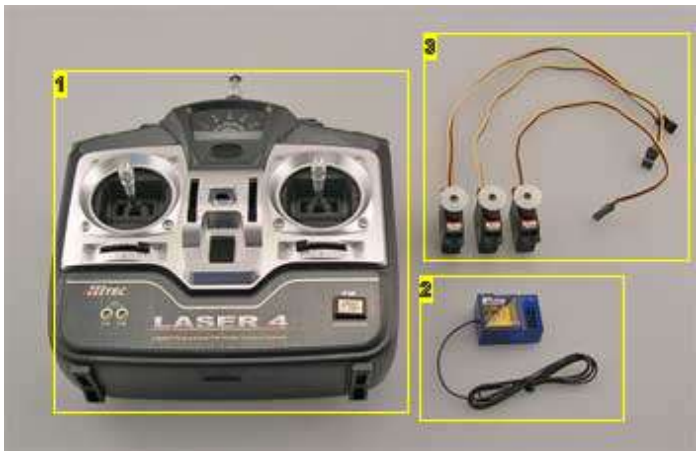


Image Notes
1. this is a typical R/C transmitter with 4 channels, the one I got is a knockoff of this one, but looks very similar.
2. this is a typical R/C receiver. Mine has it's connector pins on the end of the unit instead of the top, enabling me to plug my receiver directly onto the control board.
3. these are typical servo motors. They can be controlled directly by the R/C receiver and are useful for many things.

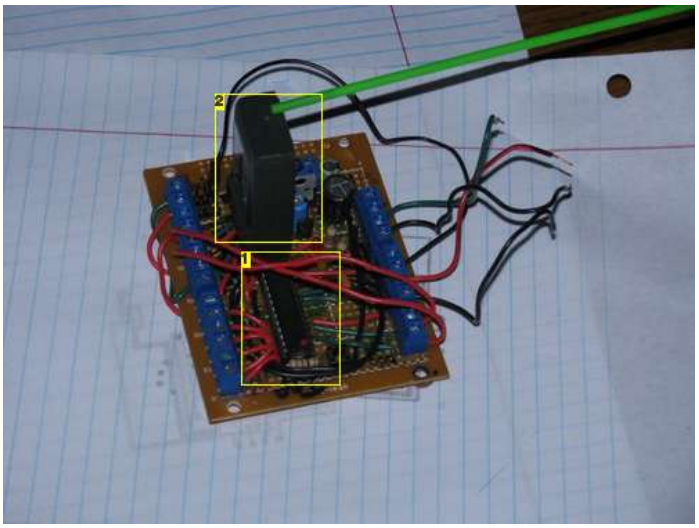


Image Notes
1. the Atmega168 from my Arduino (I bought a few extras to use for projects like this). I remove it when I need to re-program it in the Arduino.
2. my R/C receiver plugged into the control board. Notice the green antenna coming out.

Step 2: The Motor Driver

I built several motor drivers before finding a design that worked for my needs. For what it's worth, there are several nice products already out there that are fully assembled and require a lot less work if you are not interested in building your own electronics. The Open Source Motor Controller is an open source design that has been under constant community improvement for several years now and can handle up to 160amps at 36vdc! But they are over $100 and only control 1 motor. The Sabertooth 2x25amp motor controller is nice and controls 2 motors, but it is $125.

So I thought I would just make an extremely simple dual h-bridge that could handle at least 25 amps at 24vdc continuous and handle surges of up to 100amps for a few seconds. Once I found out that you can parallel Mosfets and multiply their current carrying capacity accordingly, I thought I would come up with a simple design and slightly complicate it by adding more mosfets until I had enough to handle the current that I needed. Digikey has a good selection of Mosfets to choose from and good filters to narrow it down by what you need, so I spent a lot of time looking for Mosfets that were rated for around 50amp and could handle over 30 volts. Also, they have to be cheap because my plan is to use a bunch of them. I decided on the FQP47P06 p-channel and the FQP50N06L n-channel Mosfets from Fairchild Semiconductor, which I bought from Digikey.

If you are wondering what an H-bridge is, find out here: en.wikipedia.org/wiki/H-bridge and this will all make more sense to you.

The design is simple: 2 P-channel mosfets control the high-side switches and 2 N-channel mosfets for the low-side switches. But instead of using 1 mosfet for each switch, lets use 3. Now we have 12 mosfets per H-bridge (3 mosfets x 4 switches) and theoretically the ability to carry 150 amps (that is not accurate though). The board is as small as I could make it with nothing touching. Each set of 3 mosfets have heatsinks and are bolted together to help dissipate heat. Also, there is an 80mm cooling fan mounted directly above mosfets to further keep them cool. The mosfets are very good at handling sudden changes in direction and speed changes.

Since there are 24 mosfets in total (8 groups of 3) I dubbed it the Triple-8. It is running at the Arduino default PWM frequency of 1kHz (I plan on playing with that to get the frequency higher). The board has 4 inputs, 2 for each bridge. If you bring an input HIGH, that side of the bridge goes HIGH.

Ideally, you would control the board by holding 1 input LOW and applying a PWM signal to the other input. This allows for easy speed control. I have written into the code that if you bring digital pin 7 HIGH, the code switches to Relay mode and either turns the mosfets all the way ON or all the way OFF. This is far more difficult to control, but is useful sometimes.

If you are interested in building your own H-bridge you can download the eagle file to etch a pcb and the schematic to show where everything goes. You can get everything to make this dual h-bridge at Radio-shack (including the copper clad), except the Mosfets and a special resistor network I used to save space. I bought most of the parts from Digikey though because it was cheaper and arrives to my house in 2 days.

Here are the parts needed for this motor driver:

(12) FQP47P06 - P-channel mosfet 47a 60v - Digikey - $1.73 ea
(12) FQP50N06L - Logic level N-channel mosfet 52a 60v - Digikey - $1.04 ea
(4) 2n7000 - Logic level N-channel mosfet 200ma 60v - Digikey - $0.26 ea
(8) 4606X-1-470LF-ND - 47ohm bussed resistor network - Digikey - $0.25 ea
(6) ED1609-ND - 2 position screw terminal - Digikey or Radio Shack- $0.46 ea
(24) CF1/84.7KJRCT-ND - 4.7k 1/8w resistor - Digikey or Radio Shack - $1.78 (for 50pk)
(1) PC9-ND - 3"x4.5" 1-sided copper-clad .064" 2oz copper - Digikey or Radio Shack- $4.66
(4) P5575-ND - 1000uf Capacitor or similar - Digikey - $1.19 ea
(1) 330ohm - 1kohm resistor 1/4w - for power LED, doesn't have to be exact
(1) power LED any color you like, I use the 3mm size to save space

Maybe something smaller?

If you are going to use this for something smaller than a 100lb lawnmower, you can look up one of the many H-bridge circuits and build your own smaller motor controller with as few as 4 mosfets (or BJT transistors) or even use a packaged IC H-bridge like the l293d (dual 1 amp) or the l298n (dual 2 amp).

Or if anyone is interested, I will post a schematic and Eagle .brd file for a smaller version of this H-bridge that only requires 8 mosfets total (everything else is the same), and it can handle about 10amps at 24vdc.

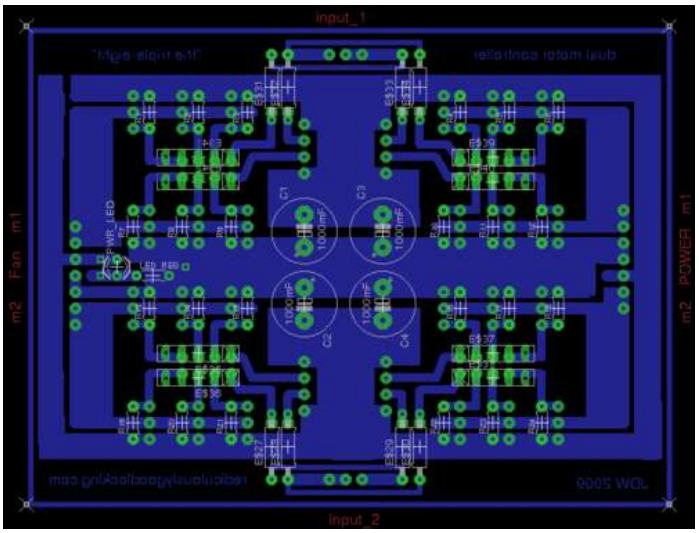
Etching:

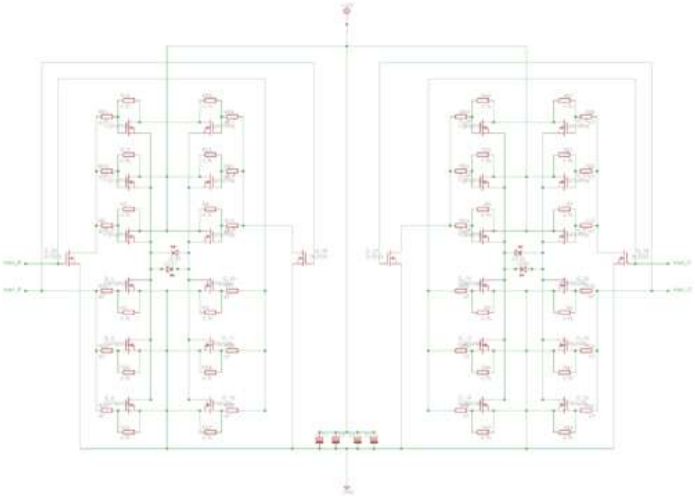
I am not going to go into all the details of PCB etching, because there are already many excellent instructables on that topic. So once you download my .BRD file of my

motor controller, all you need to do is print the .brd file onto some magazine paper using a laser printer, and iron that onto a piece of clean copper-clad. Then etch it with your favorite etchant solution (I use 2 parts Hydrogen Peroxide to 1 part Muriatic Acid and it works perfectly). And remove the toner with Acetone when done etching.

For ease of assembly I designed this board to be Single-sided and to use only through-hole components, no surface-mount stuff to mess with! Yay for you.

You can get the .brd files for the various h-bridges at www.rediculouslygoodlooking.com





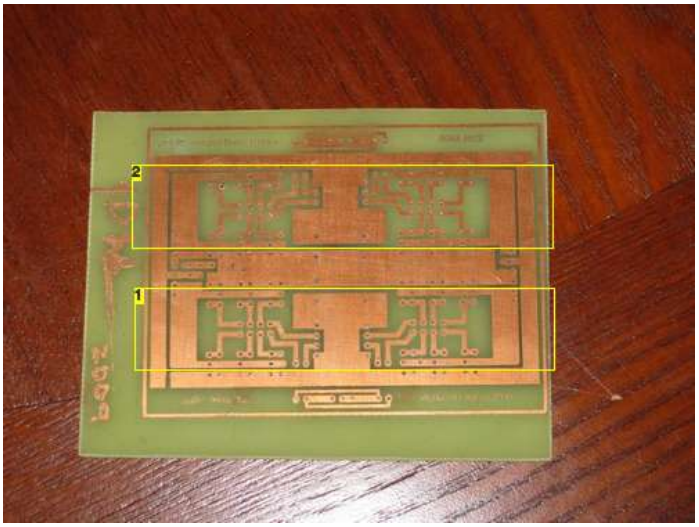


Image Notes
1. this is the 2nd H-bridge, notice that the motor screw-terminals for each motor will be on opposite sides of the board.
2. This is the 1st H-bridge

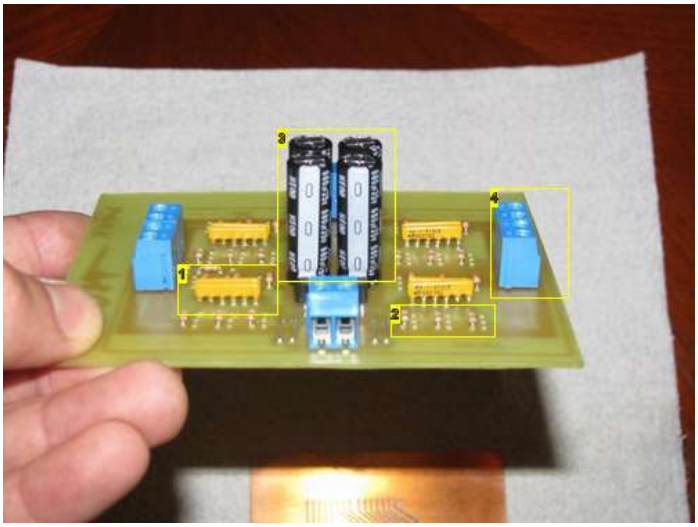


Image Notes
1. bussed resistor networks 47ohm. They have 1 input and 5 outputs, this board only uses 3 of the outputs.
2. pull up/down resistors 4.7k ohm, these keep the Mosfets turned off when not being used.
3. capacitors, I used (4) 680uF 50v, but you can substitute others that fit.
4. screw terminal connectors for motor terminals and power

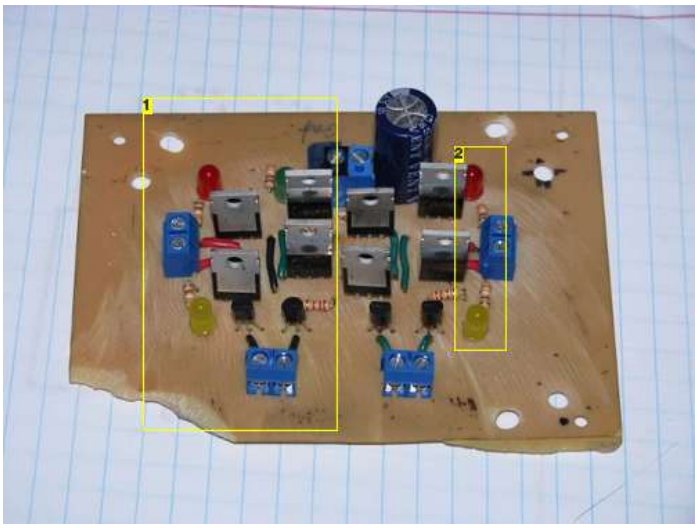


Image Notes
1. this is 1 complete h-bridge to control 1 DC motor. The 2 smaller mosfets toward the bottom are used as signal-inverters to control the High-side p-channel mosfets.
2. each h-bridge has it's own set of direction lights to determine the direction of the current.

Image Notes
1. the Triple8 motor controller with 24 mosfets, each set of 3 is bolted together and each mosfet is heatsinked. It has 3x as many Mosfets as it's little brother, but essentially the same circuit.
2. the predecessor to the Triple8, only 8 mosfets total (just enough to complete a dual h-bridge). Though it would run the Lawnbot400 around for about 10 minutes, it would end up getting hot after some use.

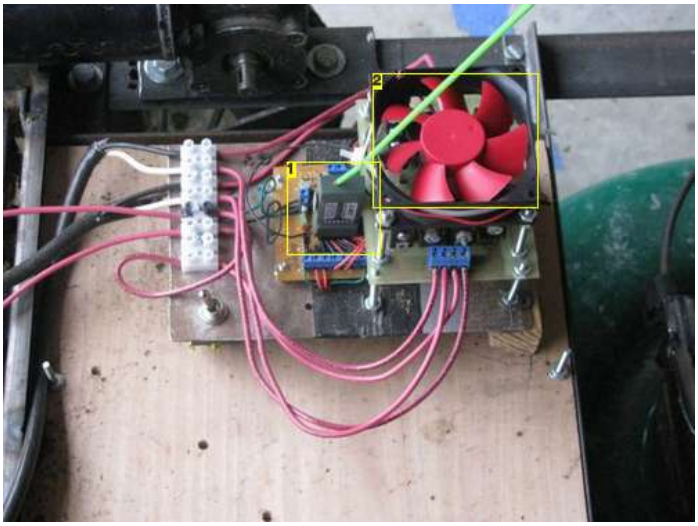
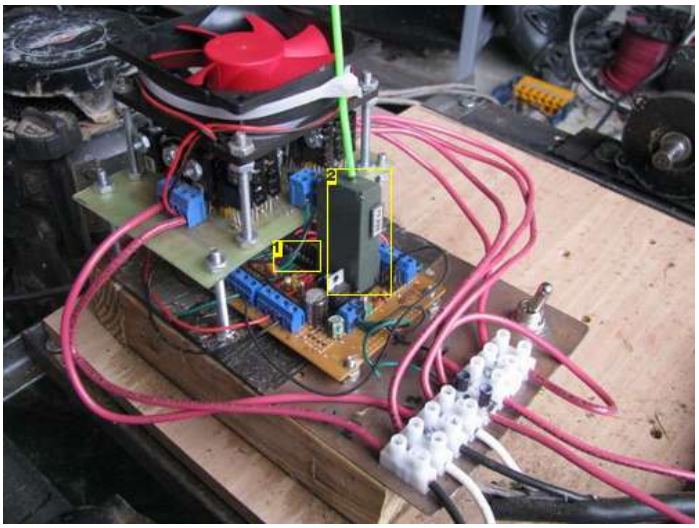


Image Notes
1. R/C receiver plugged into Arduino breakout board
2. cooling fan for motor controller (h-bridge)



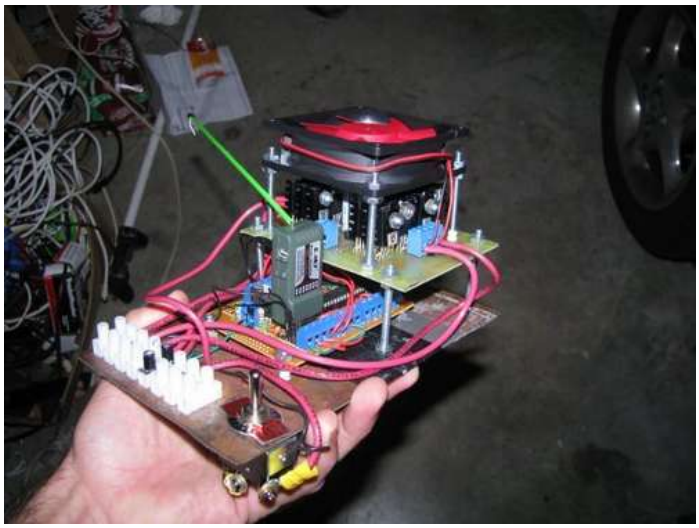
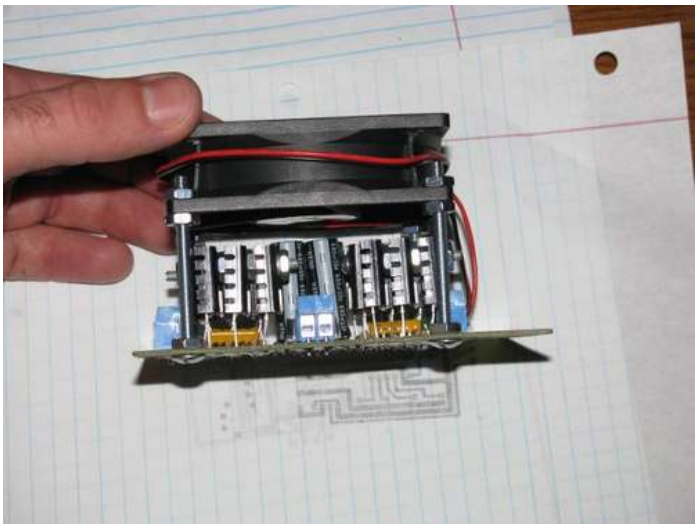
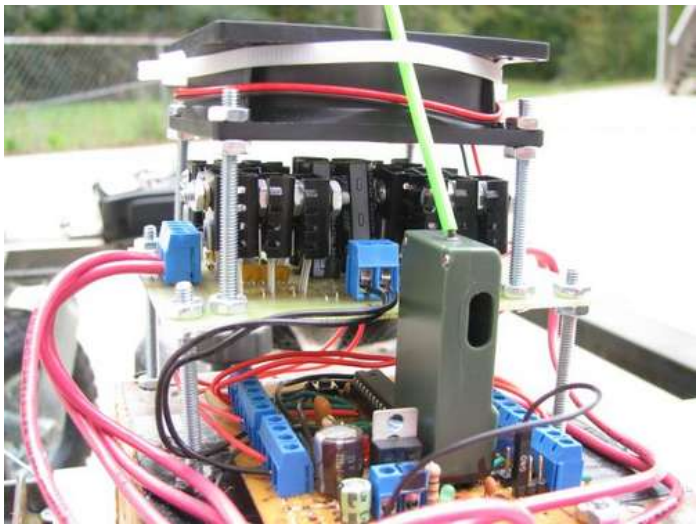


Image Notes
1. Atmega168 microcontroller programmed in the Arduino, then transferred to this home-made breakout board for permanent use.
2. The R/C receiver is plugged directly onto my home-made breakout board which supplies the +5v and GND needed for power as well as a breakout screw-terminal for each channel. This receives the signals from the remote-control (R/C transmitter) and sends them into the Atmega168 for processing.





Step 3: The Wheels

First you need to mount the drive sprockets to the wheels.

The EASY way:
If you are smart and have more money, you can find a set of wheelchair motors that have the wheels mounted to them.

The CHEAP way:
I could not find any in my price range, so I went with just the motors, then bought wheels, then sprockets. Believing it would not be strong enough to mount the wheels directly to the motors, I opted to mount the drive wheels on an axle, then the motors to the frame, and use chain to transmit the power. A picture is worth 1000 words, so look at them carefully.

Mount the sprockets to the wheels:

I had to place the sprocket on the center of the wheel and drill 3 holes through the sprocket and then through the wheel itself. Once the sprocket is lined up and properly centered, I placed the 3 bolts through the sprocket and wheel and tightened them up as much as possible. I then welded the sprocket to the wheel hub to keep it centered.

The wheels from Harbor Freight Tools have built in bearings for a 5/8" shaft, hence the 5/8" threaded-rod we are going to use as an axle.

Repeat this process for both wheels.

There is more detailed info tagged in the pictures.

Image Notes
1. The bolts coming from around the axle are the 3 bolts that hold the sprocket onto the other side.

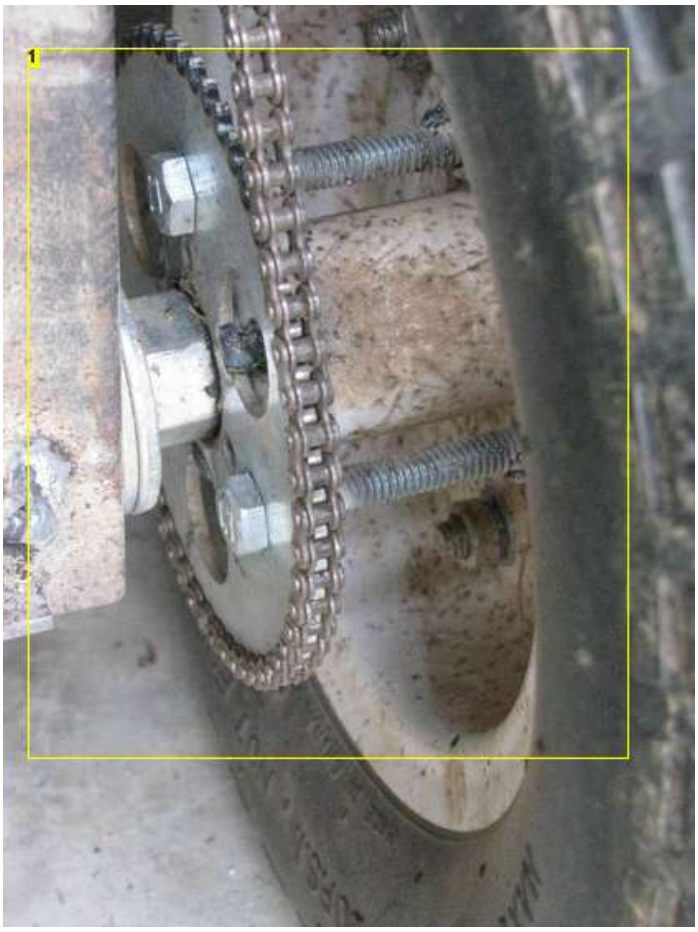


Image Notes
1. The drive sprockets are about 6.5" in diameter and had no holes to mount them. I had to drill 3 holes and mount bolts through the sprocket into the wheel. I then added a small bead of weld to keep it centered around the axle.

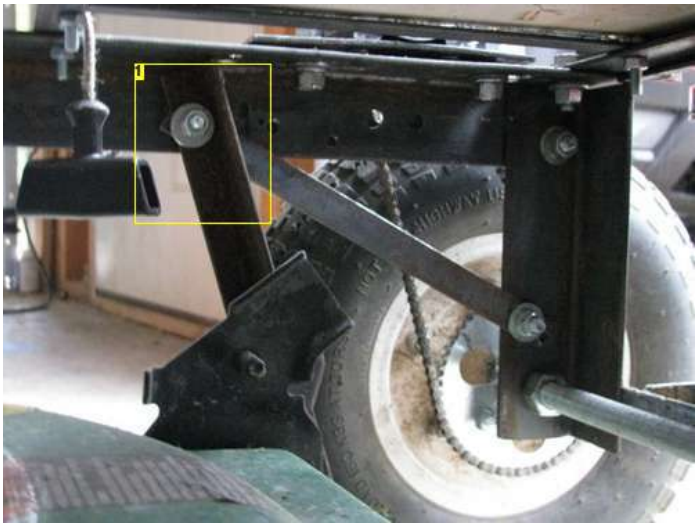


Image Notes
1. save a bolt on each side by using the same one that you used to bolt the frame riser brace into the frame.

Step 4: The Frame part A

This is the difficult part to explain. You will likely have to have some mechanical ability and a good set of tools to build a large metal frame from scratch. And since this was a prototype, the dimensions are not all perfect, but luckily they don't need to be.

The frame will be custom measured for your particular lawnmower, so I won't be giving you exact measurements.

Tools needed to build a frame:
measuring tape
angle-grinder
ratchet set
crescent-wrench
a level
electric drill
bolts, nuts, washers, and lock washers of either 3/8" or 1/2" diameter and 3/4"- 2" long
drill bits the size of the bolts you are using
1" and 2" angle-iron (36" long pieces) you'll need both
1" square tubing (36" pieces, steel)
1" flat steel bar (36" long pieces)
the 4 wheels you got from Harbor Freight Tools (2 drive wheels and 2 caster wheels)
5/8" threaded rod (36" long) and several 5/8" nuts/washers

First you need to plan out the frame of your bot. Since I was attaching a lawnmower, I started by measuring the height that the lawnmower stood off the ground and took some basic measurements to see how big the frame needed to be. My frame turned out to be about 24" wide (this distance must match the width from the center of the rear lawnmower wheels) and 48" long (long enough for the front caster wheels to swing 360 degrees without hitting the front of the mower deck) and about 18" tall. Since we want the height of the mower-deck to be adjustable, we are going to attach the mower to the frame by removing the lawnmower wheels and using angle-iron to suspend the mower-deck from the frame of the bot.

1. I started out by using 2 of the 36" pieces of angle-iron (2" wide) for the main part of the frame running long-ways.
2. Cut the rear-piece of angle-iron the width of the rear of the mower (this measurement will be from the center of the left-rear wheel to the center of the right-rear wheel).
3. Drill holes in the ends of the angle-iron and bolt the rear-piece to the adjacent pieces from step 1, making sure they are straight.
4. Cut two front-pieces using 1" square steel tubing, the same length as the rear. We need 2 in the front to bolt the caster wheels to.
5. Drill holes and bolt these 2 pieces to the front of the angle-iron from step 1. You have to measure the holes from the 2 front caster wheel's mounting plates and drill the pattern into the front square tubing bars. Then bolt the wheels through those holes onto the front of the frame.

I later added another set of 2" angle-iron bars to the front caster wheel assembly to make the length of the bot adjustable at the front (see pics)

Now we should have a rectangular frame with the front wheels attached.

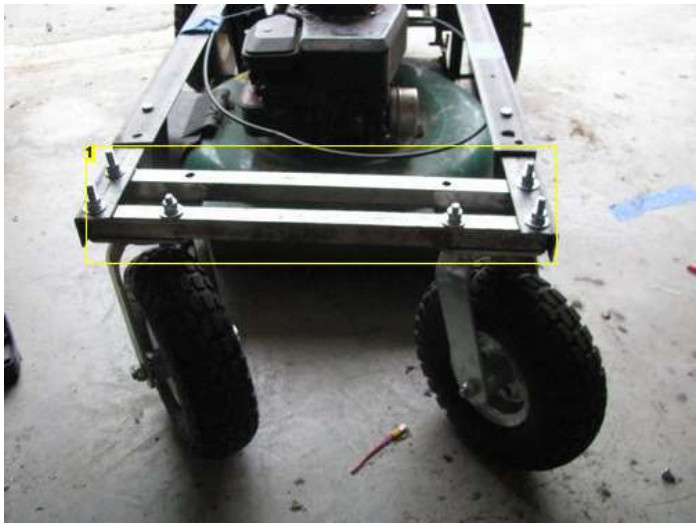


Image Notes
1. the front 1" steel square tubing that the front caster wheels attach to.

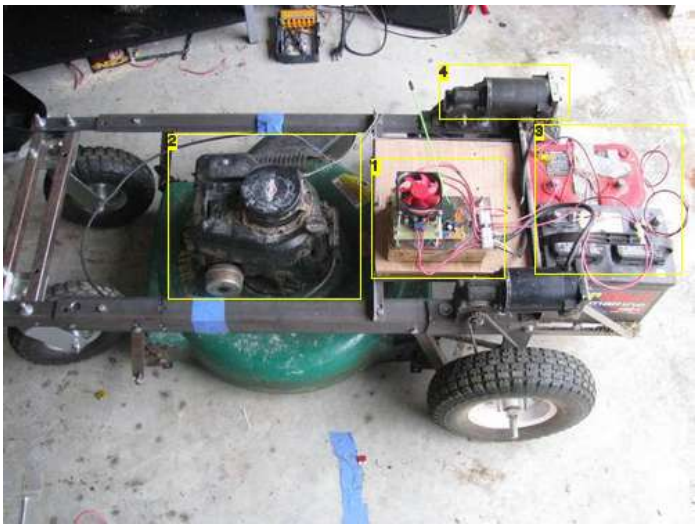


Image Notes
1. Motor controller and Arduino
2. push mower
3. (2) 12v batteries (deep cycle marine is the best)
4. electric wheel-chair motors

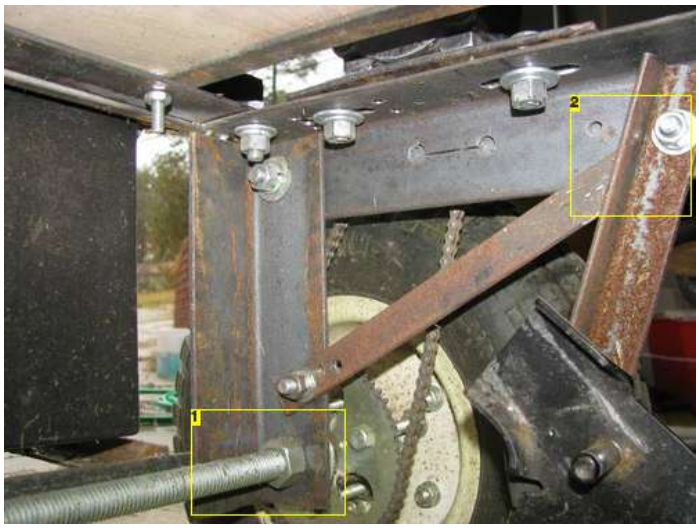


Image Notes
1. you need 1 nut on the inside of the frame riser bar to, and 1 on the outside to hold it securely to the axle.
2. I bolted the support bar in with the rear lawnmower-deck hangers to save a bolt on each side.

Image Notes
1. the rear bar should be the same width as the center of the rear wheels on your push-mower (must be measured before you remove the wheels).
2. the main frame bars.
3. the support brace

Image Notes
1. one of the main frame bars from step 1, which is 2" angle-iron.
2. the other main frame bar from step 1

Image Notes
1. the front left mower deck hanger
2. the rear left mower deck hanger

Step 5: The Frame part B

We now need to see how far down to mount the drive axle to make the frame level. So raise the rear of the frame up until the top of the frame is level with the ground (use your level). Now measure the distance from the top-rear of the frame to the ground, this is the frame height.

Now we need to take into account the height that the wheels will raise the axle off the ground. So measure the distance from the center of the rear drive wheel to the ground (the wheel's radius). Subtract the wheel radius from the frame height and we will have the correct distance from the top of the frame to the drive axle, which we will call the frame-riser height (we need to cut these pieces next). They are going to connect the rear of the frame down to the axle which the wheels will be mounted on.

6. We are going to add 2" to the frame-riser measurement (so we have a little to work with) and cut the 2 frame risers (mine were about 10-12" long).
7. Now drill (2) 5/8" holes, 1 at the bottom of each frame riser (about 1" from the bottom), this is where the drive axle will go through.
8. Drill 2 holes at the top and bolt the frame risers to the rear of the main-rectangular frame with the frame-risers pointed down.
9. Now feed the threaded-rod through the bottom holes of the frame risers and use 4 nuts to secure the frame risers to the drive axle (1 nut on each side of each frame riser, tightened down).
10. put the rear wheels on the axle and use 1 more nut on each wheel to secure them to the axle (these wheels have built in bearings). The sprockets should face inward toward the frame.

Now we should have a frame that stands on it's own with 4 wheels. However, the rear axle is not completely secure yet. We will need to add 2 braces from the bottom of the frame risers (near the axle) to the main part of the frame in order to keep the frame risers positioned properly. These braces can be flat steel and do not need to be very thick, they are just keeping the frame risers from moving.

Measure about 2" above each axle and drill a hole, then measure how far down that hole is from the top-rear of the frame and measure the same distance from the rear of the frame toward the front. Drill another hole on each side at this measurement. The support braces will need to be measured to be bolted in through these holes on each side (see pictures). The placement of the support braces is less important, meaning you can bolt them in wherever is convenient, as long as they are present.

Image Notes
1. the rear bar should be the same width as the center of the rear wheels on your push-mower (must be measured before you remove the wheels).
2. the main frame bars.
3. the support brace

Image Notes
1. The drive sprockets are about 6.5" in diameter and had no holes to mount them. I had to drill 3 holes and mount bolts through the sprocket into the wheel. I then added a small bead of weld to keep it centered around the axle.

Image Notes
1. The bolts coming from around the axle are the 3 bolts that hold the sprocket onto the other side.

Image Notes
1. one of the main frame bars from step 1, which is 2" angle-iron.
2. the other main frame bar from step 1

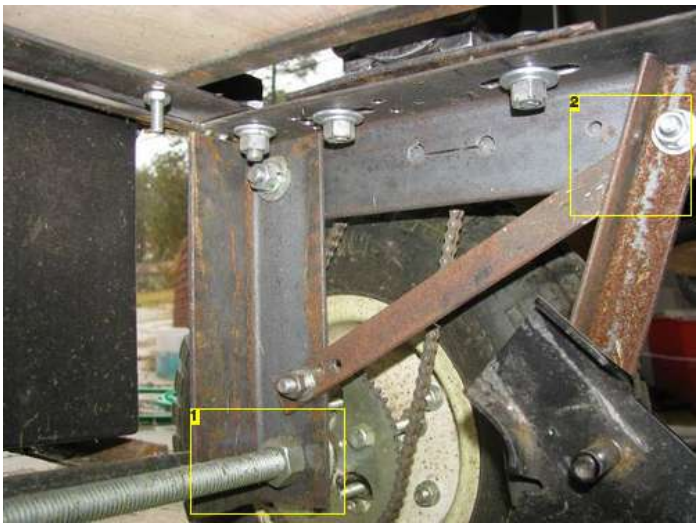


Image Notes
1. you need 1 nut on the inside of the frame riser bar to, and 1 on the outside to hold it securely to the axle.
2. I bolted the support bar in with the rear lawnmower-deck hangers to save a bolt on each side.

Step 6: Mounting the motors

This was the most difficult part to plan out on the frame. We need the motors to be adjustable so we can adjust the tension of the chain, however they just have 4 holes in the bottom of each motor and nobody makes a mounting plate that I could find.

The simplest way I could come up with was to mount the motors to an 8" long piece of 2" angle-iron, and then mount that piece of angle iron to the frame through some specially cut holes that allow the motor mount to travel forward and backward (but not side to side) along the frame.

Make the motor mount plate:

Cut an 8-10" section of 2" angle-iron, depending on how much room your motors need to mount. Mine only needed about 4", so I made it 8" to have plenty of room for the mounting bolts. Drill a hole about 1.5" from each end of the top of this bar, this is where the mounting bolts will go through the frame.

Mount the motor to the motor mounting plate:

Now you have to find the center of your motor mount plate (the 8" long piece of 2" angle iron) and measure the mounting holes on your DC motors. Use a sharpie marker to plot the hole pattern from the motor, centered onto the motor mount plate. My motors have (4) 1/4" diameter tapped holes in a rectangular pattern on the bottom of the gear box.

Drilling and cutting the adjustment holes on the frame:

Next you need to drill and cut the holes in the frame to let the motor mounting plate become adjustable. I cut these holes using a dremel tool and a cutoff wheel. You have to line up the motor mounting plate (with motor mounted preferrably) onto the frame rail and use a sharpie marker to mark where the holes will need to be on the frame rails. Start as far back as you can (without hitting any other bolts underneath the frame), and mark the center of each hole. Then move the motors forward 2" and mark the holes again. You want to cut the holes out of the frame so that the motor mount plate (with bolts going through the frame), can move forward or backward about 2". The holes in the frame are the width of the bolt and about 2" long. I drilled 1 hole at each end and used the dremel to cut out the rest.

The holes drilled in the motor mount plate are just single holes for the bolt to fit through, the holes through the frame were cut with a Dremel tool with a cutoff wheel to make channels for the motor mount bolts to travel forward/backward through. You want the 2" angle-iron motor mount bracket to set as much on top of the main frame rails as possible, the bolts (which you can't see with the motors mounted) that hold the motors to the motor mount plates will keep the motor mount plate from laying flat against the frame bars. Go ahead and mount the motors loosely to the frame using 2 bolts on each.

Cutting and connecting the chain:

Now get your 10' of #25 chain and wrap it around the main drive sprocket on the wheel. With the motors pushed all the way toward the back of the frame (closest to the drive wheel sprockets), wrap the chain around the motor drive sprocket and mark where they overlap. You need 2 of the universal chain links from to connect the 2 loose ends. Cut the 2 pieces of chain and connect them to each side with the universal links to connect them.

Tensioning the chain:

Push the motor mounts forward until there is good tension with the chain, and tighten up the bolts that hold the motor mount plates to the main frame.

Now you can generate electricity. Connect a voltage meter to 1 set of motor terminals and push the bot around.

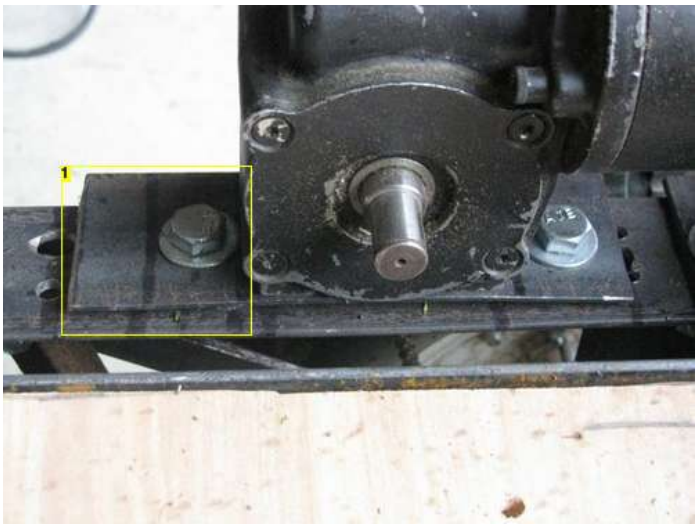


Image Notes
1. notice the motor is mounted to this piece of 2" angle-iron and that is mounted to the frame with these bolts. They allow the motor to slide forward/backwards on the frame when loosened.

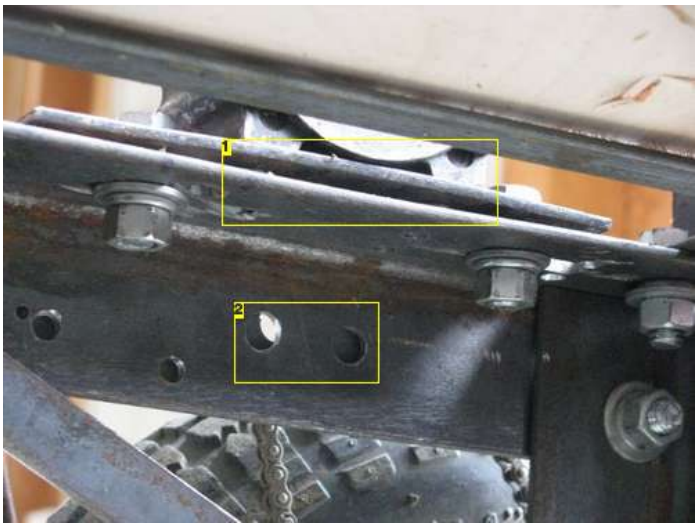


Image Notes
1. notice the gap between the motor mount plate and the main frame bar. This is caused by the bolts that hold the motor to the motor mount plate.
2. These are 2 unfinished holes for a 3rd mounting hole which I later deemed unnecessary.

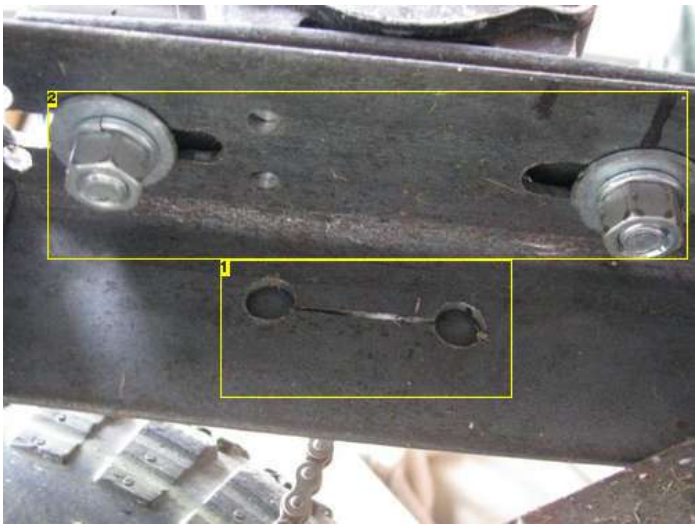


Image Notes
1. This is how to make the motor mount slide holes. Drill 2 holes where you want the ends of the track to be. Then use a Dremel with a cutoff wheel to cut a straight line between the tops and bottoms of each hole. They should end up looking like the ones above with bolts in them.
2. Tighten up these bolts when you get proper tension with the chain.

Step 7: Mounting the mower deck

Next we need to mount the mower deck to the frame. Remember we made the frame wide enough that the edges of the frame would be centered on the lawnmower wheel shafts.

All we have to do is cut 4 pieces of 1" angle-iron equal lengths so that the mower deck hangs evenly from the frame.

So measure the height of the frame from the top to the ground. Now measure how high the mower sits off the ground from the center of the wheel shafts (when the original wheels are on the lawnmower and all the height adjusters for each wheel are in the middle position). Now subtract the height the mower sits of the ground from the frame height, and cut 4 pieces of 1" angle iron to that length.

Now drill 1 hole in the end of each piece of angle-iron, about 1/2" from each end. The holes at the bottom will need to be the diameter of the lawnmower wheel shafts and the holes at the top will need to be bolted into the frame (hung at equal distances from the top of the frame).

Once you have all 4 hangers installed, you can install the mower deck and tighten up the bolts. Make sure you have at least 1/2" of clearance or more between the drive tires and the lawnmower wheel shafts.

You are almost ready to go.

Image Notes
1. the front left mower deck hanger
2. the rear left mower deck hanger

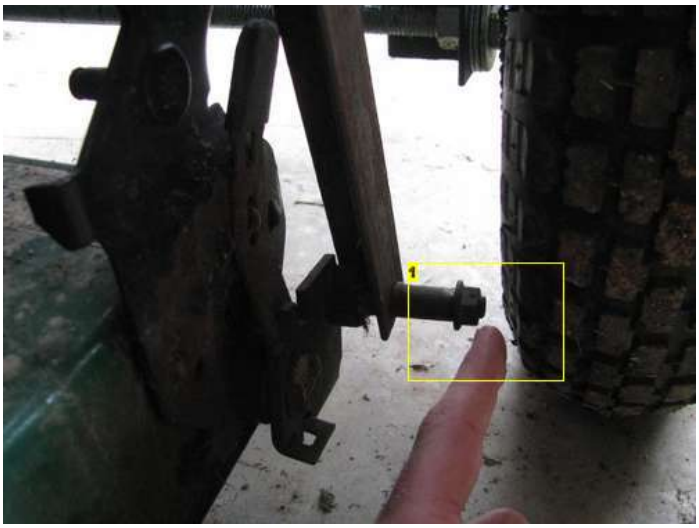


Image Notes
1. make sure to keep the old wheel shafts from touching the drive tires (leave 1/2" or so)

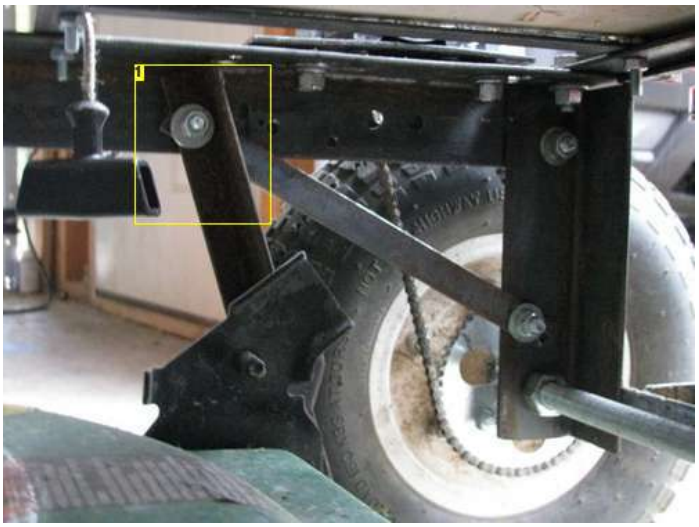


Image Notes
1. save a bolt on each side by using the same one that you used to bolt the frame riser brace into the frame.

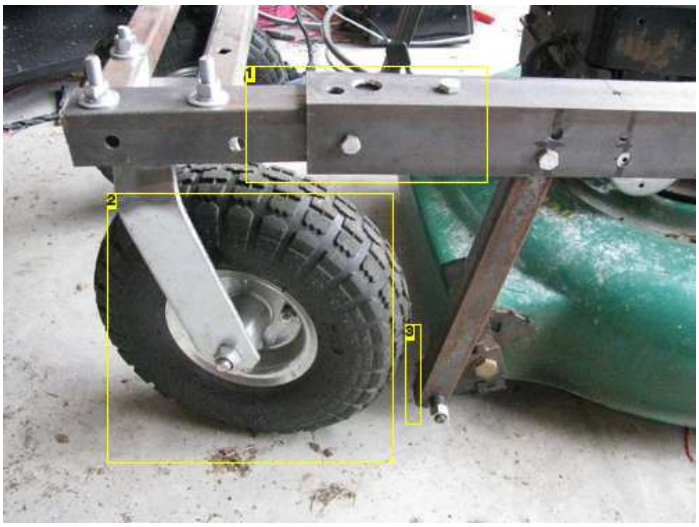


Image Notes
1. adjustable total length (for different model push mowers)
2. caster wheels with 360 degree turning
3. leave a gap or the front wheels will hit the mower deck!!!

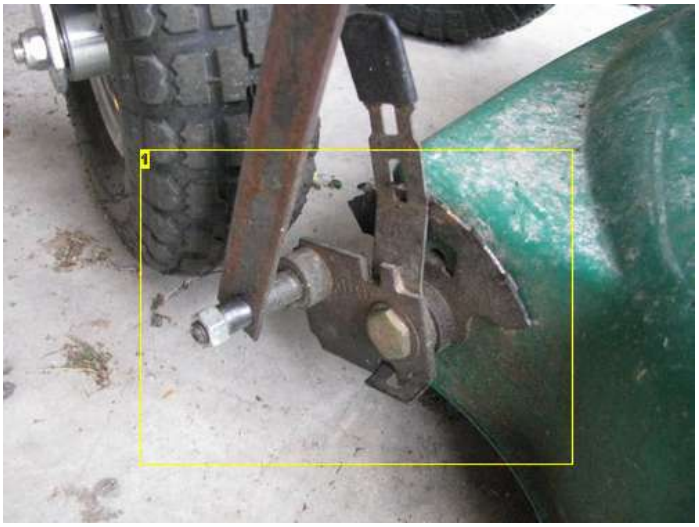


Image Notes
1. by mounting the lawn mower deck-hangers to the old wheel shafts, you can still

Image Notes
1. these are the 1" angle-iron lawnmower-deck hangers, they hold the mower-

adjust the mowing height of the mower deck without taking anything apart.

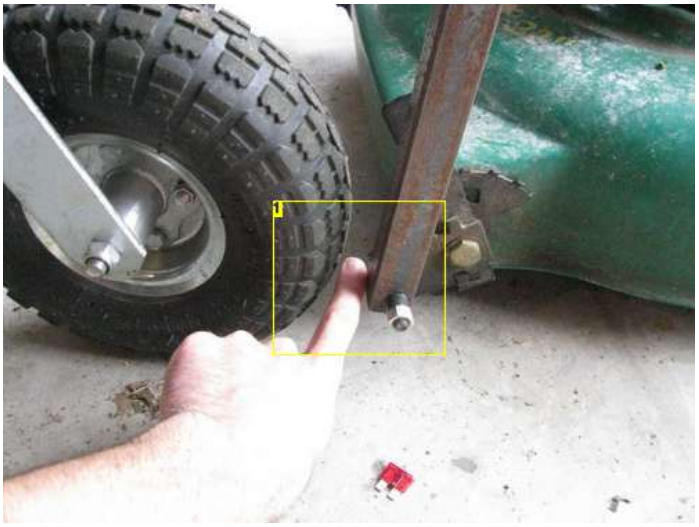


Image Notes
1. make sure the front caster wheels won't hit the mower deck when they swing around (leave at least 1/2" clearance)

deck to the main frame

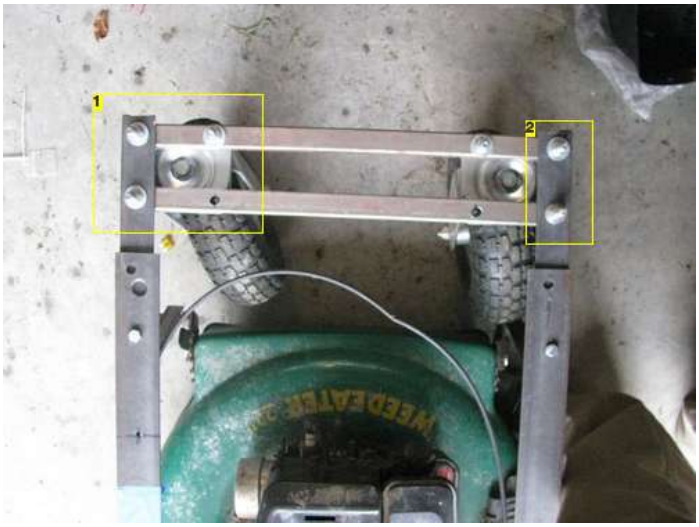


Image Notes
1. I only installed 3 of the 4 bolts on each front caster wheel.
2. these 2 bolts on each side go through the caster wheel mounting plate AND the frame

Step 8: Select and Install the batteries

This is the simple part. Go BIG. I only bought 1.. which I got at Walmart for $62.

I got 2 car batteries (actually 1 marine deep cycle and 1 gel-cell car battery) both 12vdc. They together keep my lawnmower running strong for the duration of my front and back yard (I have about 1/2 acre of grass to cut and it is somewhat hilly). I slacked while trying to learn about batteries and just went with the biggest ones I could find for the price (the gel cell is actually used). I initially thought 12vdc would work, but the added weight of the mower deck made it travel so slowly at 12vdc, that it would not quite make it up some larger hills, so 24volts was necessary. The 2 batteries are connected in series with each other.

The microcontroller is also powered by these batteries. I have never had any problems with the electronics not getting enough power, so I didn't see the need to have a separate power supply.

The batteries (due to their weight) are mounted behind the rear wheels. This GREATLY improves control of the bot because it counters the weight of the mower deck in front. Zero-turns are very easy now.

I needed a place to hold the 2 big batteries that were going to power the lawnbot, so I measured the 2 batteries and welded a small 1" angle-iron frame to hold them. It is welded to the rear of the frame behind the drive axle to maintain even weight distribution.

You can use bolts and 1" angle-iron to make a battery holding cage that is bolted to the rear of the bot, or you can use smaller batteries and secure them to the top of the bot. 12v 20ah Sealed Lead Acid batteries can be found online for around $35-45 each. Any battery rack that you can whip up will likely be just fine, as long as it can support the weight of the batteries it is carrying. I used a welder to speed up the process.

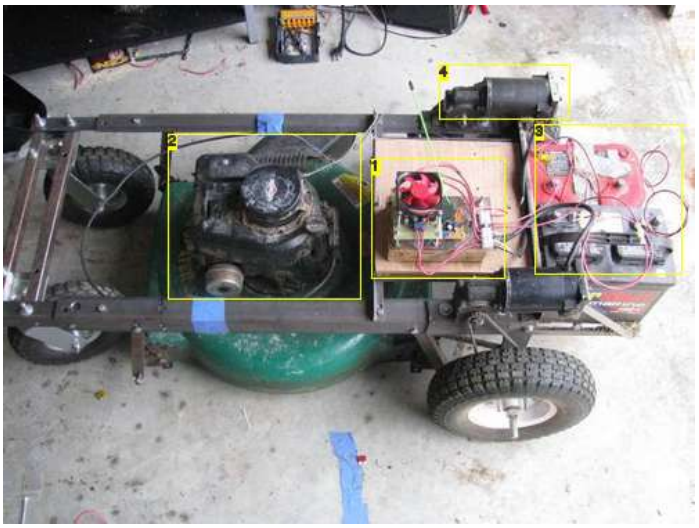


Image Notes
1. Motor controller and Arduino
2. push mower
3. (2) 12v batteries (deep cycle marine is the best)
4. electric wheel-chair motors

Step 9: Mount the electronics

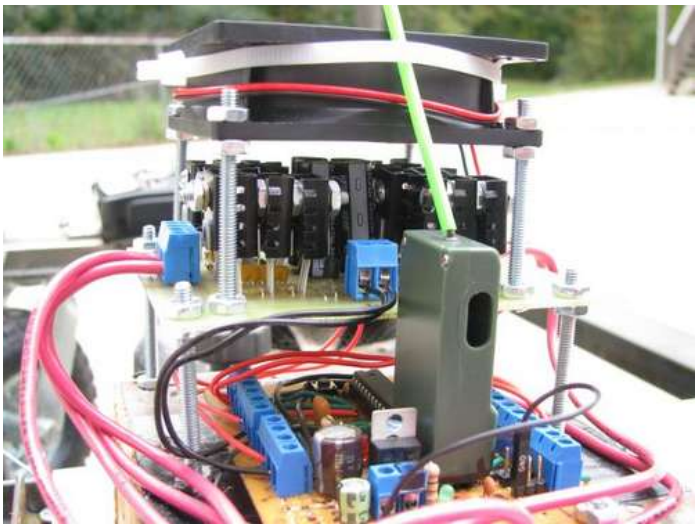
Connect the electronics to the motors and batteries. The motor drive board has 1 connector for the main battery power and 1 power connector for the 80mm cooling fan that I would highly recommend you install directly above the mosfets. There is spacing for some long skinny bolts to hold a cooling fan. I bolted the motor driver above the Arduino breakout board to save space.

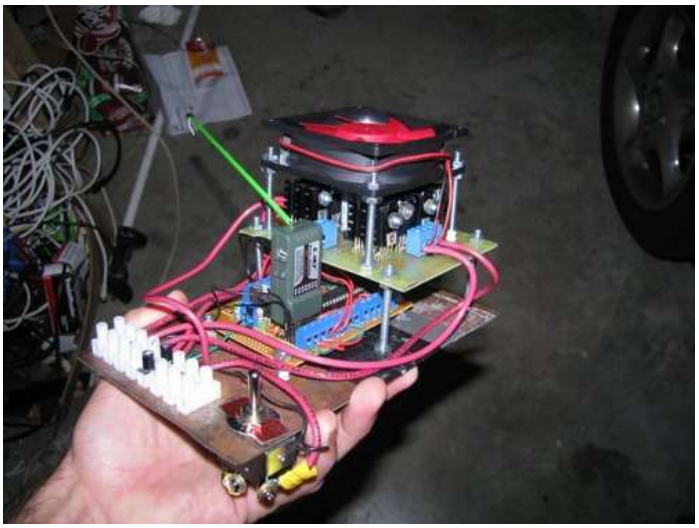
Also, you might want to use some smaller wire coming from the batteries to power the Arduino board, as the 10ga wire I used for main power and motors is a bit overkill for the microcontroller.

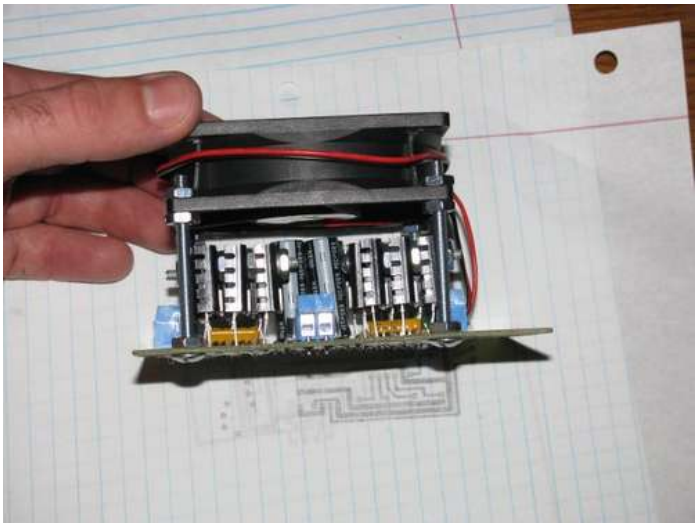
I installed a 30a 120v toggle switch from Radio Shack to switch the main power ON/OFF, this is my kill-switch. I also found a terminal-block for power distribution at Radio Shack for a few bucks. It is the white thing that all the wires go into in the pictures. This makes removing the electronics a whole lot easier.

It is very important that you wire everything up correctly. Otherwise you might blow up the motor controller.

So make sure that you check the code before connecting anything to verify that you haven't mixed any wires up.







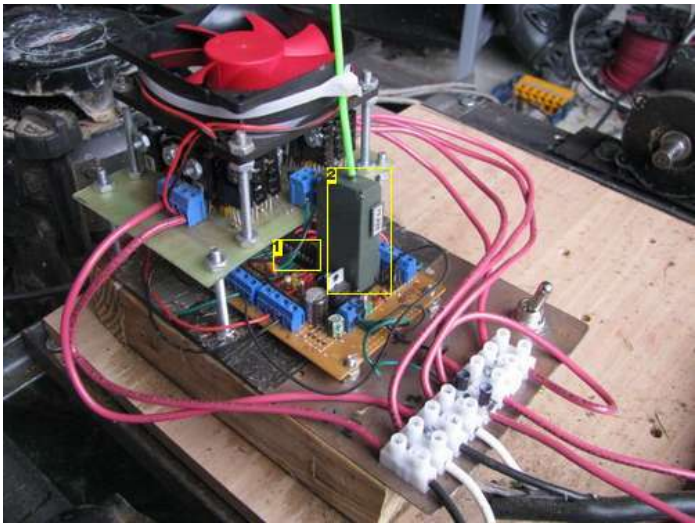


Image Notes
1. Atmega168 microcontroller programmed in the Arduino, then transferred to this home-made breakout board for permanent use.
2. The R/C receiver is plugged directly onto my home-made breakout board which supplies the +5v and GND needed for power as well as a breakout screw-terminal for each channel. This receives the signals from the remote-control (R/C transmitter) and sends them into the Atmega168 for processing.

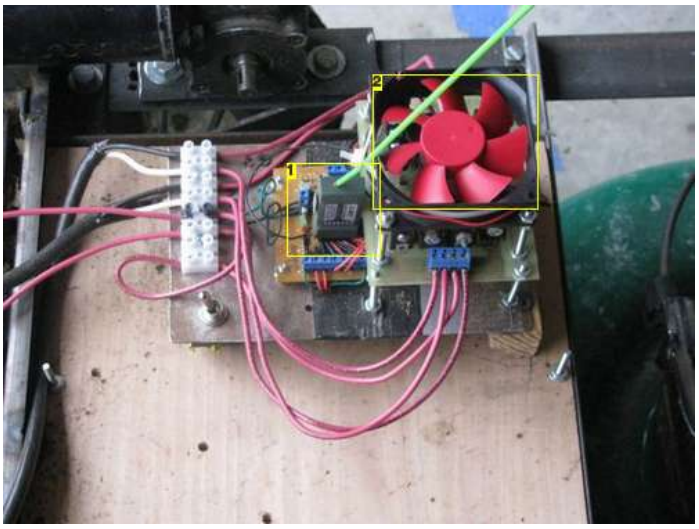


Image Notes
1. R/C receiver plugged into Arduino breakout board
2. cooling fan for motor controller (h-bridge)

Step 10: The Code

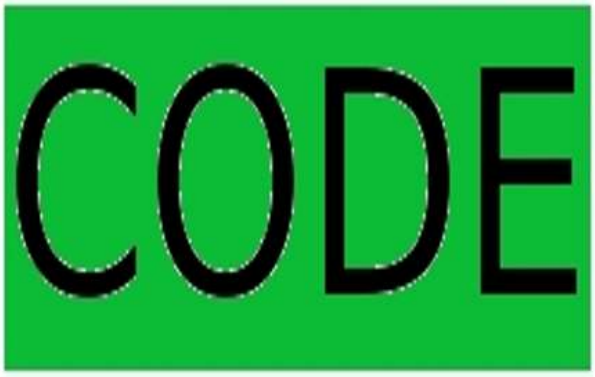
I changed the code so that the Interrupt Service Routines (ISR) would run more quickly and the sketch would spend less time in the ISR. This means less overhead which means more signals are processed and smoother operation of the bot.

I also added a 2nd sketch for the 2nd microcontroller to process 2 signals (you can add as many more as you want) using the pulseIn method instead of using interrupts. This only processes about 1/5th of the available signals from the R/C Receiver, but also severely decreases the chance of receiving a "BAD" signal. Also, since the power relay is setup to only be ON if the signal is "GOOD", when you go out of range, it automatically shuts off the power to the motors only.

The 2nd Atmega by default should have digital pin 4 used as the R/C servo signal input from the R/C receiver, digital pin 6 should control a 5v relay or N-channel mosfet that is used to switch the 60amp power relay ON/OFF. That is all that is needed, you can also use an LED on pins 12 and 13 to indicate whether the relay is ON or OFF.

You can also add 2 12v running lights from Walmart for a car... I use an N-channel mosfet directly tied to pin 9 of the 2nd Arduino to control the brightness of the lights using a hacked channel on my transmitter. This input from the receiver would go to digital pin 2. Check the code.

Download the .zip file on this page and upload the sketches. If you don't plan on adding the 2nd Atmega with the failsafe and killswitch, that is fine. You can still update the new code for just the main Atmega and it should run more smoothly.



File Downloads

Lawnbot400_code.zip (152 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Lawnbot400_code.zip']

Step 11: More Videos

here are a few more videos in case anyone wanted to see...

#2

#3

#4

#5

More Videos

Related Instructables

How To Change Lawn Mower Oil by toolrepair

How To Replace a Lawn Mower Blade by toolrepair

How To Build A Solar Charged Remote Control Electric Lawn Mower (video) by hastyhost

Electric Lawn Mower by susanta

Robocam - Homemade Video Robot (Photos) by talk2bruce

Remote Controlled Lawnmower (Photos) by Bob Bowie

remote start system for car truck and suv by lonemeno

Remote control lawnmower by pfoglietta

How to Build an Arduino Powered Chess Playing Robot

by **mJusticz** on March 4, 2011

Intro: How to Build an Arduino Powered Chess Playing Robot

Judging by the sheer number of chess related Instructables, I think it's safe to say the community enjoys the game. It can be difficult, however, to find someone who plays on the same level you do. To solve this dilemma, and to increase my playing skills, I built this arduino powered chess playing robot.

****UPDATE** PCWorld just posted about this project on their blog! Thanks, guys!**
This project is entered in the 13-18 division of the robotics week contest.

The board works like any other xy table, with a few key differences. First, the x axis has an extra servo attached to it, which raises and lowers a magnet. The magnet is attracted to pieces on the chess board above, allowing them to move. Second, embedded in the board are 64 magnetically activated reed switches, allowing the arduino to know the location of each piece.

What I love about this project is its adaptability. If you decide you're done with it as a chess board, it can instantly convert into a CNC mill by modifying a few pieces. I'll talk more about this possibility at the end.

All in all, while there was definitely some great novelty in watching the computer move pieces around with the board, this project was not as successful as I had hoped. The magnets were way too powerful, so extra pieces would often get drawn in when they shouldn't have. I think with some changes this could have been an even better, more functional project. However, I think this Instructable still serves as a pretty good guide to make your own functioning chess playing robot.

Step 1: Parts and Materials

You may have many of the parts for this project already, but if you don't, the whole list costs ? $350, depending on where you get your parts from. Many, many of them can be salvaged, so look to recycle before you buy!

- 1 Arduino Uno or Diecimila

We'll be using this arduino to drive our stepper motors and servos. You can pick these up just about anywhere online. I got mine from Adafruit. $30

- 1 Arduino Mega

This is the most expensive item in the project. It'll be dealing with the inputs from each chess square to let the computer know where you've moved. We're using the mega here due to its speed and number of inputs. Adafruit $65

- 1 Mux Shield

The mux shield (short for multiplexer) gives us even more inputs for our arduino mega. We'll need 64 inputs in total, one for each square. Sparkfun $25

- Motor Shield

The motor shield will be controlling our stepper motors and servo. You'll need to solder it together. Adafruit $19.50

- 1 Large chess board with pieces

This one is a little more self explanatory. We want a large chess board here because the pieces need to be able to move in between each other with disrupting others. Make sure you measure the diameter of the bottoms of the pieces. We'll need that in a moment. I'm not sure where mine is from, but you can pick them up from a flea market for a bargain. The playable area of my board is 24".

- 64 NO Reed Switches

Reed switches are magnetically activated switches. They'll help us find the location of moved pieces. NO stands for normally open, that is, the circuit is disconnected Digikey ?$30

- 16 10K 1/4 Watt Resistors

These are the pull up resistors for the built in digital pins. The mux shield, luckily, has integrated pull downs, so we don't need to worry about those. Digikey ? $2

- Roughly 90 feet of 30AWG Wire

This is the hookup wire for all of our sensors. Radioshack ? $16

- Neodymium Magnets to fit your pieces

This is where the measurements from the bottoms of your chess pieces come in handy. You'll need disc magnets to fit underneath each piece. For proper strength, they should be about 1/8" think. A great source for these is K&J Magnetics . ? $55

- 1 Large Neodymium Magnet

This magnet will be attached to the XY table underneath the board, to move each piece around. K&J Magnetics $19 Note: This was Waaaay too powerful. It would draw in pieces it shouldn't have. You'd be better off going with some smaller ceramic magnets, like you'd find at Staples or another office supply store.

- 2 Pairs of 24" Drawer Bearings

The size of your bearings will depend on the playable area of your chess board. These allow for the stepper motors to move back and forth underneath the board. Amazon ? $30

- 2 Stepper Motors

Stepper motors can move in very precise increments. In the late 90s they were in just about every piece of tech you could find. The best place to get these are in old dot-matrix printers. You can them at the flea market for next to nothing!

- 2 Vex Rack and Gear Sets

The rack gears allow the stepper motors to travel on the drawer bearings. See the Step 4 for a more detailed explanation. Vex Store $40

- 1 Standard Hobby Servo

This servo will be raising and lowering the powerful magnet below the board. You can find them at a hobby shop for ? $10, or Amazon ? $12

- 1 2' x 2' Perf Board

The perf board is super thin and will be the mounting surface for all of our reed switches. The price will vary greatly on this one, but I got mine from Home Depot for ? $5

- 1 2' x 2' x 1/2" MDF Board

Similar to the perf board, I got this from Home Depot for ? $5

- Various lengths of scrap 1"x2" wood

This wood forms the bridge between the X-Axis drawer bearings. Go behind any hardware store and you'll see dumpsters full of this stuff for totally free!

- 5 Minute Epoxy

This stuff is a godsend. It's used for just about everything in this project, from mounting motors to attaching the rack gears. I'm in love -- and I picked mine up from Radio Shack for $3

- 1 Wood Saw

You probably already have this one, but if you don't, I picked mine up at Ace Hardware for $10 a couple of years ago.

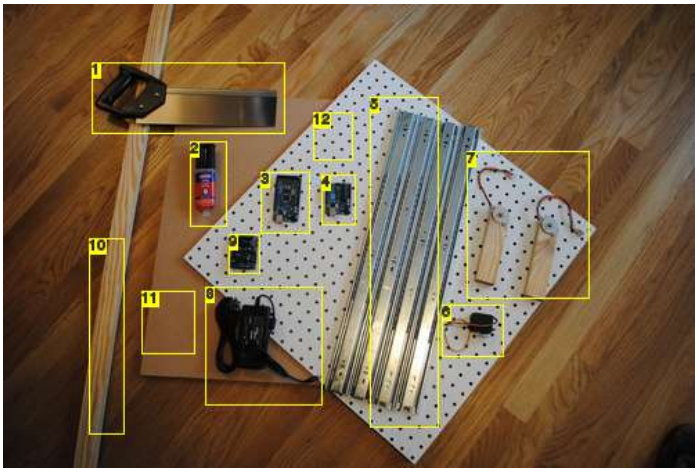


Image Notes
1. Standard wood saw
2. 5 minute epoxy
3. Arduino Mega
4. Arduino Uno. A Diecimila would also work.
5. 24" Drawer Bearing
6. Standard Servo
7. Two 24v 1A stepper motors. They are already mounted here. I describe the

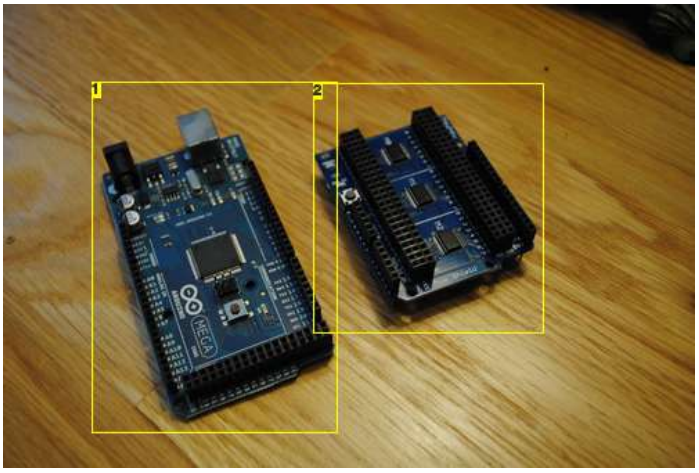


Image Notes
1. Arduino Mega... This thing is a beast!
2. Mux Shield. This gives us an additional 46 inputs on our mega.

process in the next step.
8. 24v 1A power supply
9. Adafruit Motor Shield
10. 1x2 lumber. Some of the better stuff I have found for free behind home depot.
11. 2' x 2' MDF
12. 2' x 2' pegboard

Image Notes
1. 24 gauge hookup wire. This stuff is pretty cheap -- a 90ft roll is about $5
2. Diagonal cutters -- these are great for clipping leads that are just a little too long.
3. Heat shrink tubing is great for cleaning up your connections. It is optional.

Image Notes
1. These are reed switches... metal encased in glass. They are a bit fragile. I would buy a few more than you need just in case.

Step 2: Design and Code Explanation

That parts list is a bit scary if you're not sure what everything is going to do, so here's how many of the pieces will be used.

You can see in the images below that each stepper motor can move freely about its axis thanks to the drawer bearings. On the Y Axis, each rail is connected with the wooden structure, so that the X Axis may sit atop it. Also on the X Axis is the servo that raises and lowers the powerful magnet, so that it may position itself before moving pieces.

Feel free to download the sketchup file and mess around if you're not sure of anything.

Another interesting element of this design is how to code talks with the arduino and motors. We need to address each square as a set of coordinates so that we may find slope and distance, however the traditional method of labeling squares A1, A2, etc. doesn't work particularly well in code. Standard (x,y) coordinates are much friendlier. Those coordinates, however, need to be in the form of a single number. What I ended up doing is assigning each square to a number, as you can see in image #3. Those numbers don't really work as coordinates on an 8x8 chess board, however, because we use a base 10 number system.

To solve that issue, we take the base 10 number of each square and convert it to base 8 using the modulus operator in C. 27, for example, is 33 in base 8, with the first digit being the x coordinate and the second the y. If you count over three squares and up three squares, voila! You end up on square 27. This converted coordinate system ends up looking like image #4.

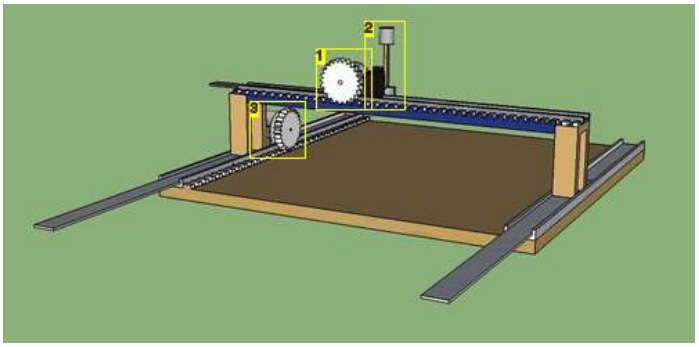


Image Notes
1. X Axis Stepper Motor
2. This is the servo that raises and lowers a powerful neodymium magnet.
3. Y Axis Stepper Motor

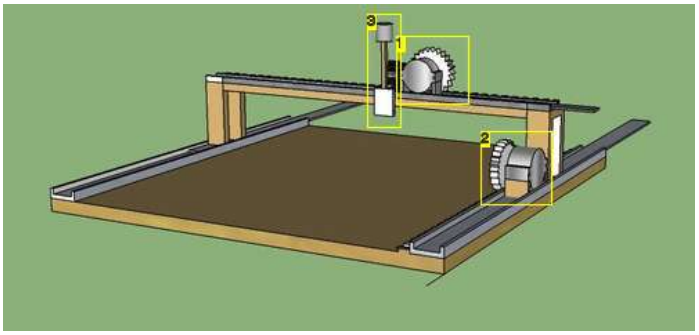
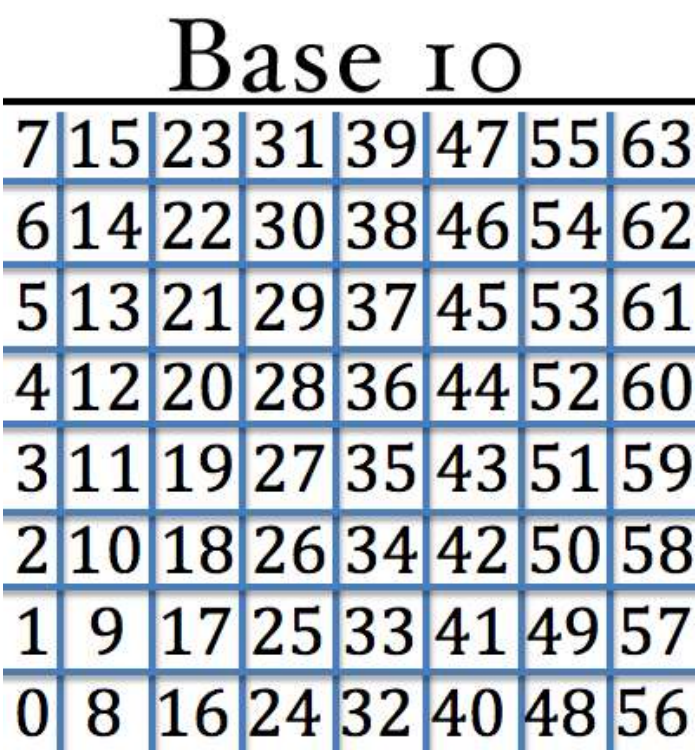


Image Notes
1. X Axis Stepper
2. Y Axis Stepper
3. Servo to raise and lower magnets (pseudo z-axis)



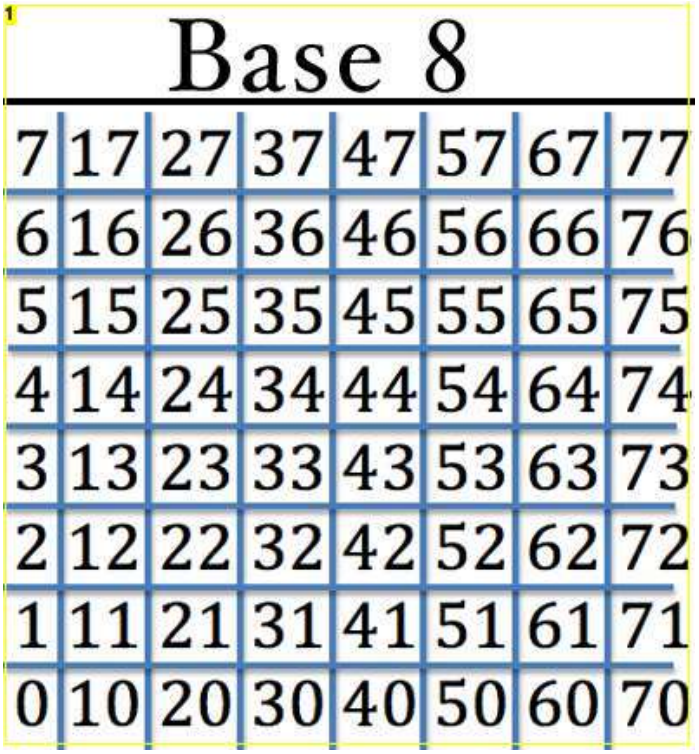


Image Notes
1. The first digit corresponds with the x axis and the second digit with the y.

File Downloads

XY.skp (347 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'XY.skp']

Step 3: Mounting the Drawer Bearings (Y Axis)

The drawer bearings are what allow the axes to move in their respective direction. The mounting instructions may vary slightly depending on the brand, but usually it's as simple as driving a couple of screws.

The only reason I've made this its own step is that **aligning the bearings perfectly is key.** Should you fail to do this, and they both point slightly outwards, they'll stop at some arbitrary point and refuse to move once you connect them. Save yourself a lot of trouble and use something you know is square as a reference for alignment. The corner of a book is perfect.

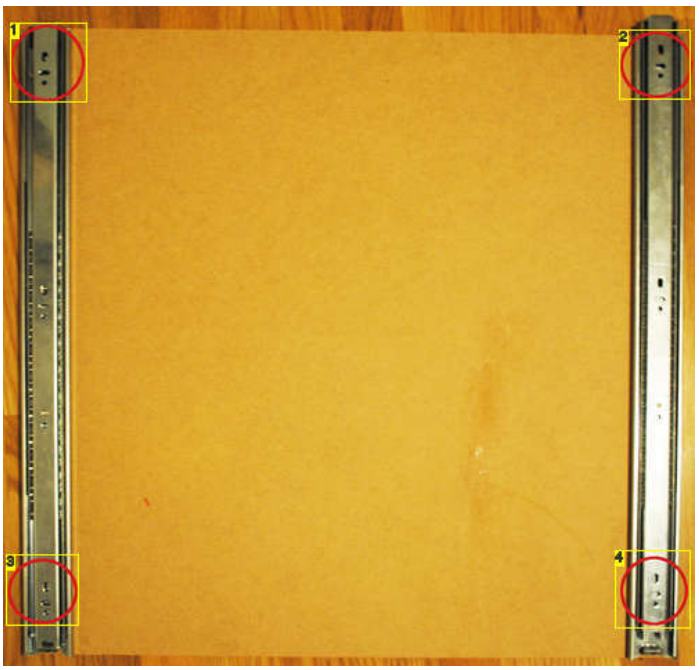


Image Notes
1. There is a screw behind this bearing. My screws came with the mounting

brackets.
2. Screw here
3. Screw here
4. Screw here

Image Notes
1. Nice and flush!

Step 4: Building the Motor Mount (Y Axis)

The stepper motors we'll be using have fantastic torque, but are circular. This means mounting them to our bearings later on will be nearly impossible, unless we build a square mount. To build one, find a hole saw with a similar diameter to your motor. You'll want to use a drill press rather than a portable drill for this, so I borrowed my school's.

Once you've cut the hole, slice the circle in half to get two mounts. This chipped the tips of my semicircle, so I used some 220 grit sandpaper to clean up the edge.

My steppers came with mounting screw holes, which line up well with the wooden frame. I used the smallest screws I could find. Mine fit so well that it wasn't necessary, but you might consider adding a bit of epoxy to strengthen the bond.

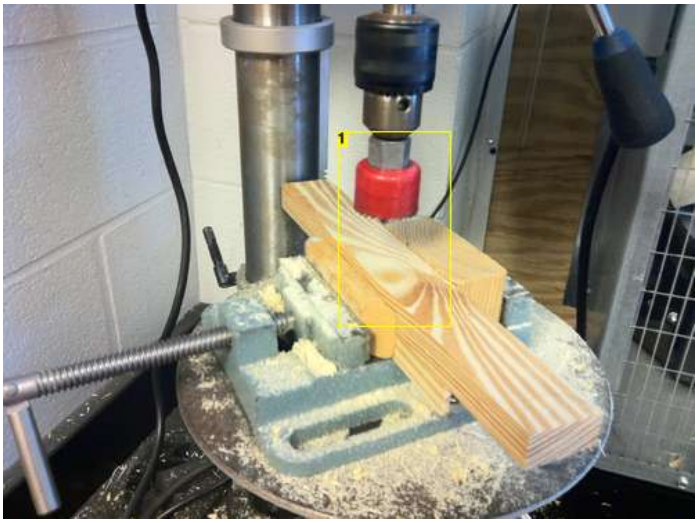
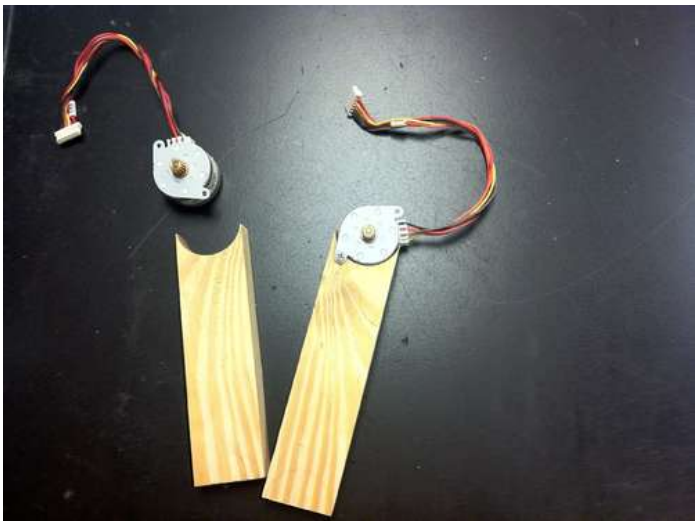


Image Notes
1. Go slowly or you risk chipping the wood on the other side.



Step 5: Installing the Rack Gears (Y Axis)

The rack gears are what allow the motor to latch onto a surface to pull itself along. Your physics teacher probably defined them as a way to convert rotational energy to linear.

Again, we use the epoxy to attach the gears onto the MDF. In addition to heavily applying epoxy to the board itself, make sure some is spread on the side of the drawer bearing, that way there is stability in two dimensions. Do your best to prevent epoxy from getting in places it shouldn't be -- you may gum up your motor.

It works out that the rack gears extend a little bit off of each end. This is a good thing -- it enables the gear to travel the full length of the board without running off. The stepper motor will be offset just enough that if the gears only covered the board's length the whole motor assembly would get stuck at one end.

Also install the circular gears onto your motor at this time. Mine had a set-screw, but you may wish to use some JB weld to hold your gear in place. If you go that route, the joint needs to fully cure before you try to use it, or you risk the gear popping off!

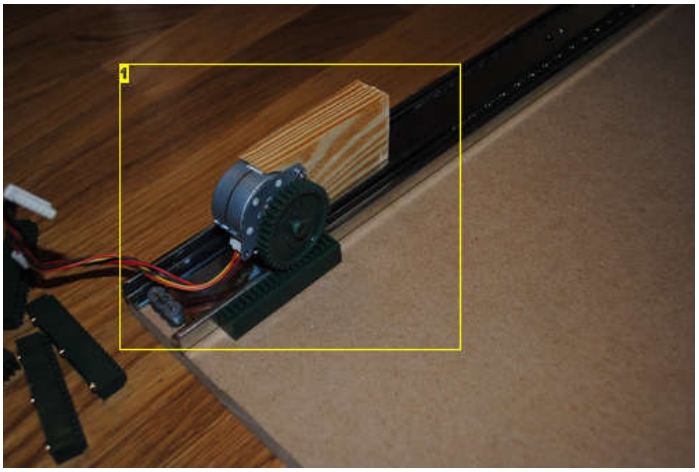


Image Notes
1. This is how the motor will fit into the rack gear once completed.

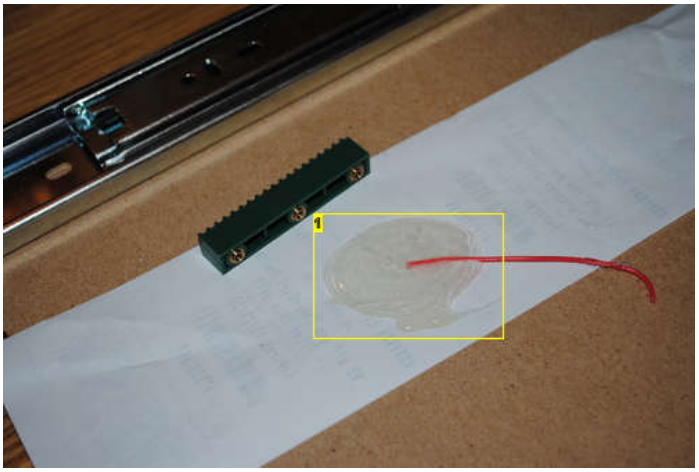
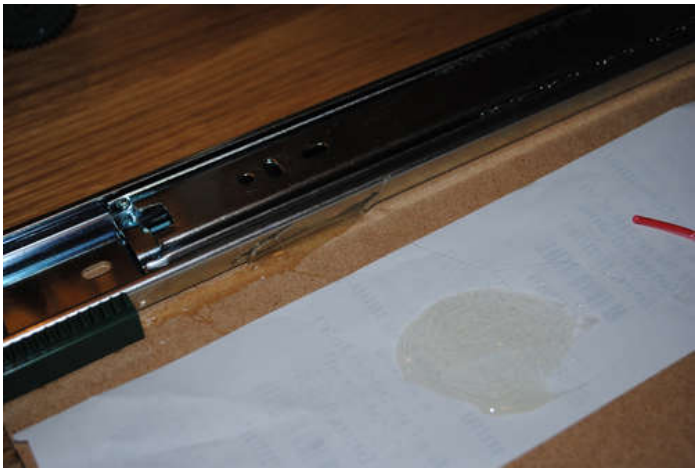
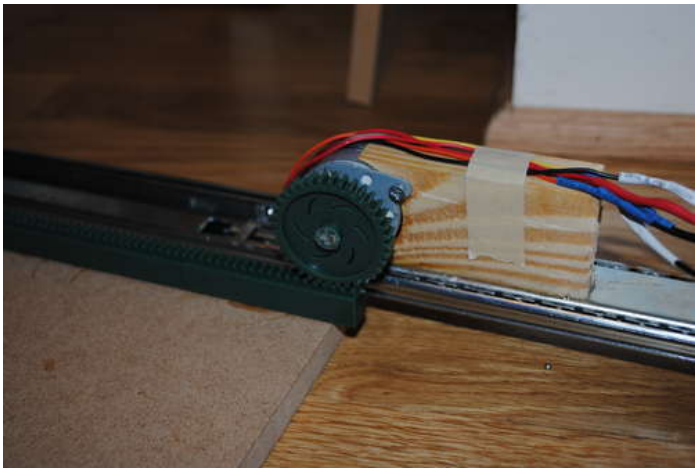


Image Notes
1. Make sure to mix this really thoroughly or it may not dry.





Step 6: Wiring and Mounting the Motor (Y Axis)

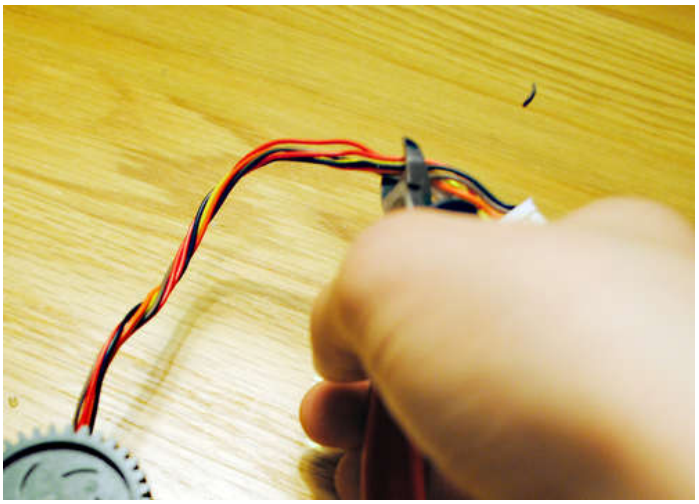
The leads that come attached to the stepper motors are very short. Because the Arduino is mounted off the board, the wires need to be at least the length of one side. That made mine about 2' 5" long. Heat shrink tubing is your friend here -- we're using enough power that it might arc if you're not careful.

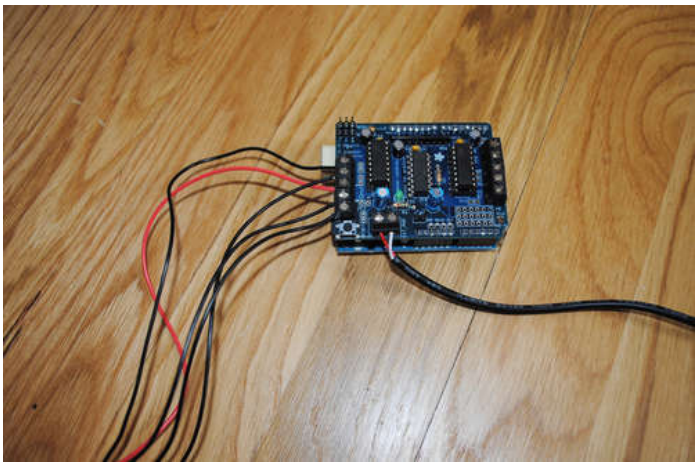
If your stepper motor has 5 wires, you're all set. If there are 6, however, it means you have to connect your center taps. Jason Babcock has a great tutorial on reverse-engineering your motors. In my case, however, the wires were the same color.

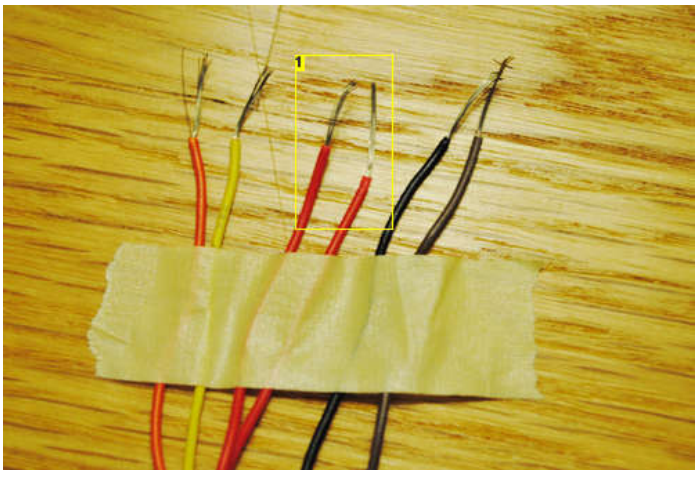
After extending the wires, the center taps go into the center of one of your motor hubs. The wires from one coil go to one terminal on the motor shield, and from the other coil to the other terminal. At this time we also hook up our 24v 1A power supply to the motor shield. If you get the polarity wrong on this, your motor shield is toast.

After trimming the motor mounting block to about 4 inches, it's time to attach it to our bearings. Mix up the epoxy, and liberally apply it to the area of the bearing the block will touch.

Also, if you have any pets, be sure to animal-proof the room you're working in. Cats seem to have an affinity for knocking over things that are drying.







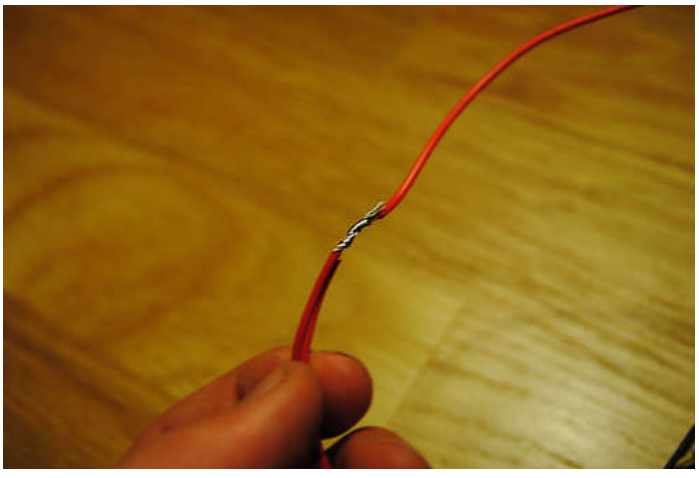
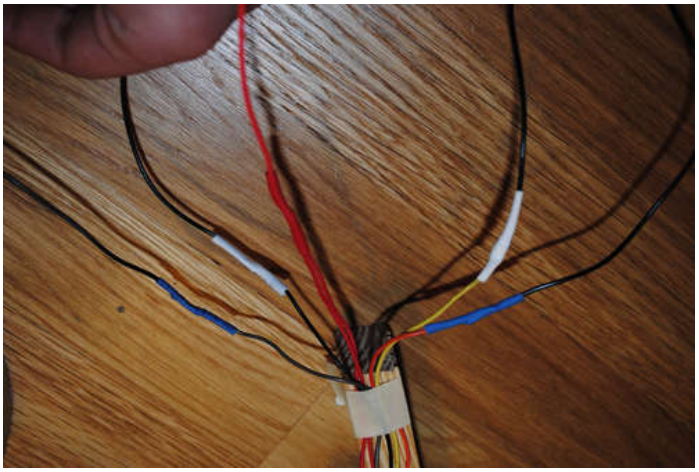
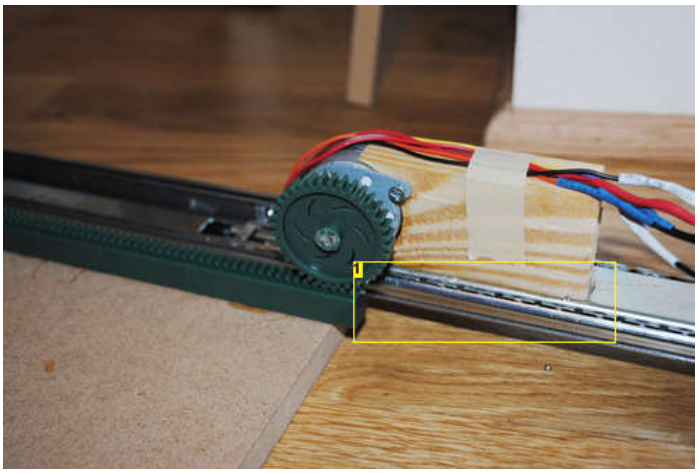


Image Notes
1. These two center taps get connected. Usually they will be the same color.





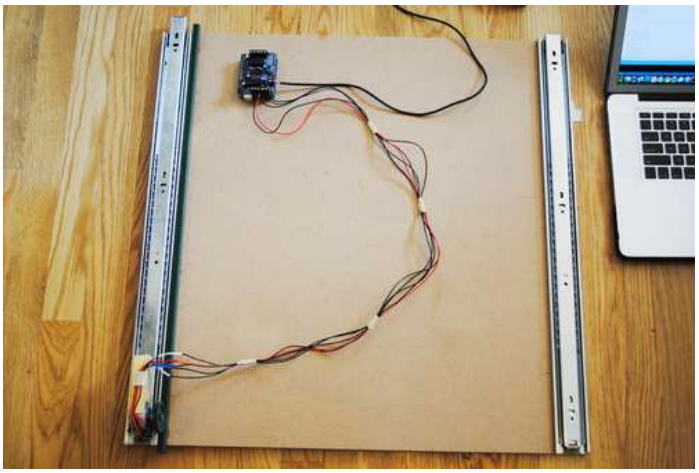


Image Notes
1. Make sure to keep the bearings epoxy free.

Step 7: Mounting the Crossbars (X Axis)

Atop the Y Axis sits a cross-bar which will hold the X Axis. To do this, we mount another block on the opposite Y Axis bearing. This will across from the motor mounting block.

Then we cut two 2' lengths of 1"x2" wood and mount them to the blocks with wood screws. Make sure these screws are in tight, or the bearings might not move at the same time. You might consider adding some wood-glue to lock them more tightly in place. While these cross bars need to be large enough to support our X Axis bearing, we want to use the least amount possible to avoid unnecessary weight. If the entire X assembly weighs too much, our Y Axis stepper motor won't have enough torque to move efficiently, or, if it's really heavy, at all.

Test that your bearings move evenly with each other. The crossbars should be as close to perpendicular with the bearings as possible.

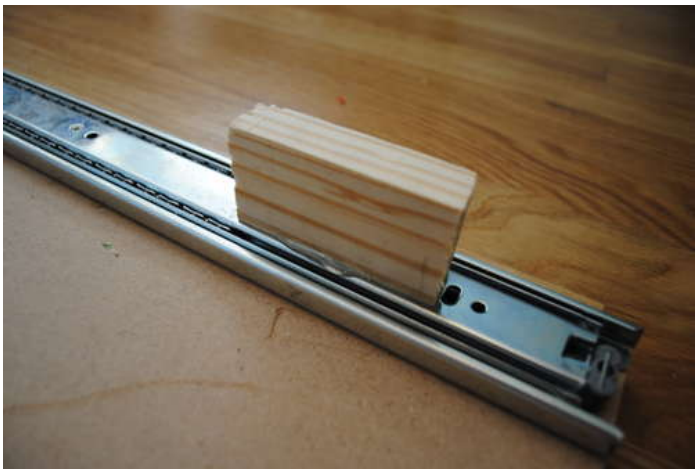


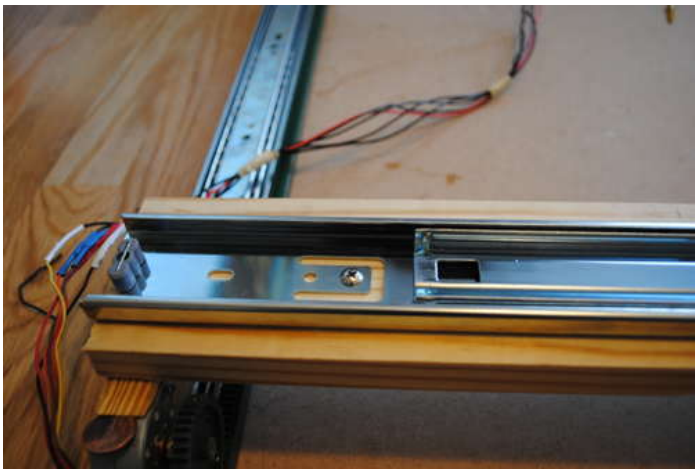
Image Notes
1. Make this as close to a 90 degree angle as possible.

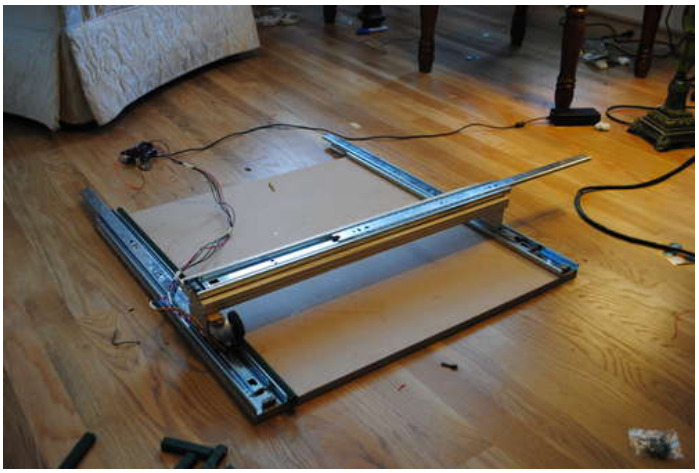
Step 8: Mounting the Drawer Bearing and Rack Gears (X Axis)

We are going to mount this drawer bearing in the same way we did on the Y Axis. Using the included mounting screws, attach it to the crossbars, making sure to leave room on one side for the rack gears.

This time, however, we have some options for mounting the rack gears. We can choose which side to mount them on. This is going to vary depending on how your motors are placed on their blocks. Once you've decided where to place them, though, they are glued in exactly the same way as on the Y Axis. Extend the rack gears slightly off of each end to make sure we have full travel on our steppers.

Again, make sure to try and keep any glue out of the sliding mechanism. If you do allow some glue to enter the mechanism, all is not lost. Fortunately for us, drawer bearings come in pairs, so we'll have one left over anyway. Mount it to the same screw holes and you're set! Just don't do it again! :)





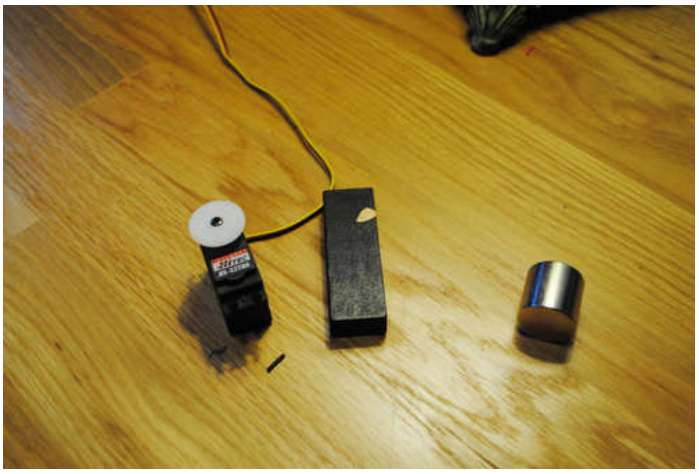
Step 9: Attaching the Magnet to the Servo (X Axis)

The chess board needs to be able to move into position and grab a piece. To accomplish this, we achieve a pseudo Z Axis by mounting a magnet to a servo on top of the X Axis. Use epoxy to attach your large magnet to a small piece of wood (Jenga blocks work wonderfully). Once that has dried, attach the wooden block to your servo.

If you're concerned about the torque of your servo, you might consider adding a counterweight to make lifting the magnet less difficult.

At this time you should also extend the leads on your servo motor. Do this the same way you did for your stepper, going one wire at a time and covering it with heat-shrink tubing or electrical tape.

Epoxy bonds best when there is a significant weight holding the two bonding surfaces together. Keep in mind that while 5 minute epoxy dries in 5 minutes, it may not cure for several hours.



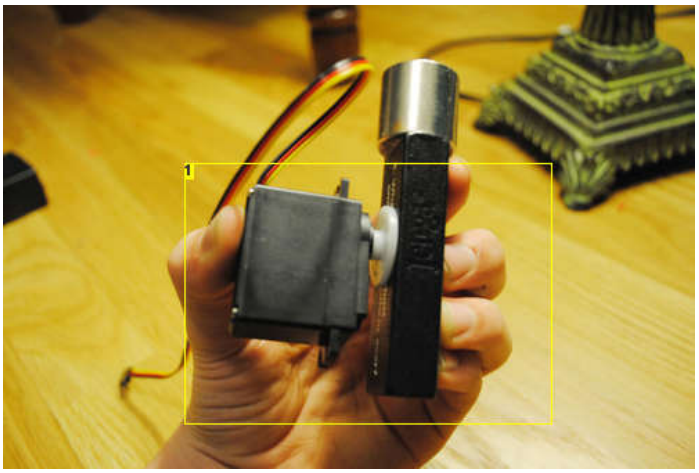


Image Notes
1. Clamp this as soon as you press it together for the strongest bond.

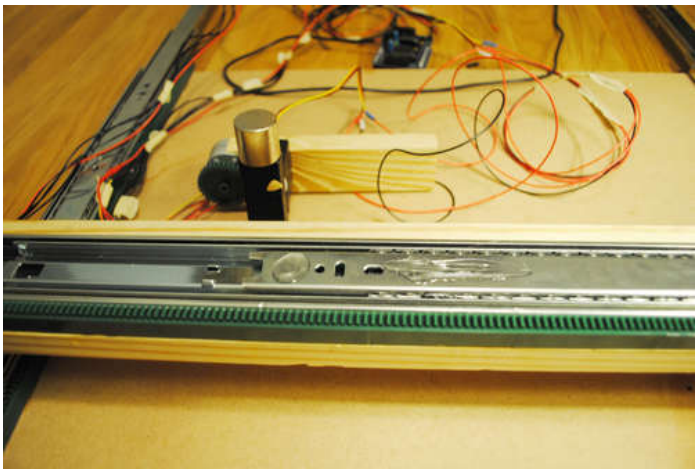
Step 10: Wiring and Mounting the Motor (X Axis)

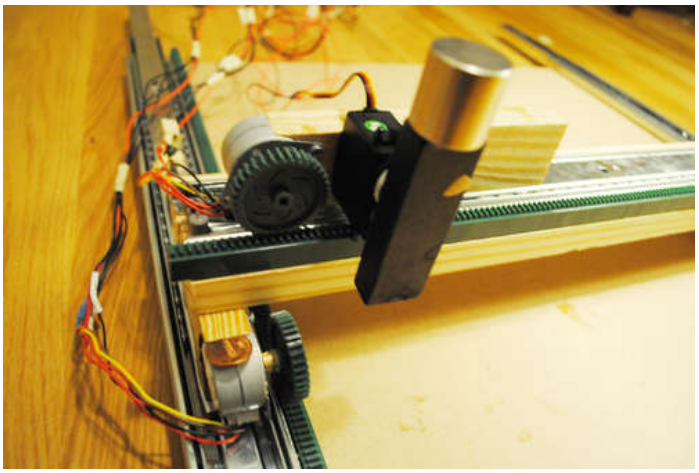
We mount and wire our X Axis motor as we did earlier, however we also need to attach our servo assembly. We need to mount the servo as closely to our motor as we can get without interfering with the gearing. This way we are guaranteed to have maximum coverage underneath the board.

Using a piece of wire or cotton swab, carefully apply epoxy to the block where you've decided to attach your servo. If you aren't careful, some epoxy may end up on your stepper motor, rendering it useless. It's a real pain to clear mixed epoxy out of motors, so it's worth the extra time it will take. Firmly hold your servo in place for several seconds, and leave it clamped to dry.

Once your servo is mounted, we can attach the motor block to our bearings. Being cautious as to not get glue in the sliding mechanism and, like earlier, apply the epoxy to the exposed metal the block will touch.

After that is dry, wire the motor like you did before, extending each lead. This time, however, we'll connect it to the other motor terminal. Congratulations, the XY Table is done!





Step 11: Wiring the Sensors

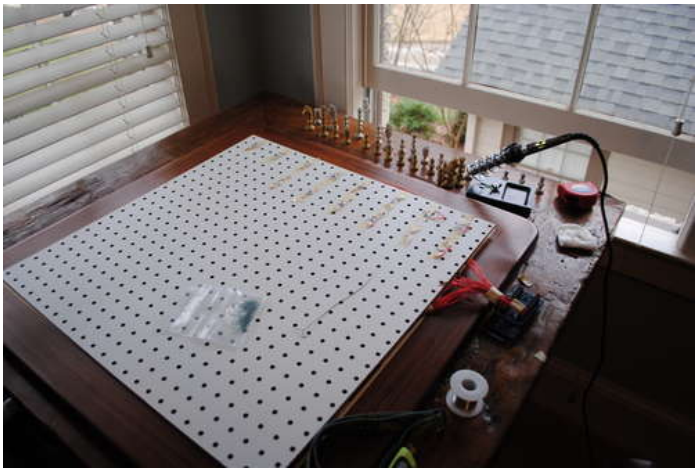
Each chess piece has a magnet embedded in its bottom, which makes detecting location really simple. Underneath each square is a magnetic reed switch hooked up to our Arduino mega. When a change is detected, the Arduino spits out the coordinates.

Pull up resistors make sure we don't get false readings from our sensors. 48 of the 64 switches won't need pull up resistors, because the multiplexer has them built in. Unfortunately, we still have to solder 64 sensors. To make this go a lot faster, tin your wires before you try to solder them to the switches. Basically, just add solder to the wire alone before soldering with it. Label each switch with tape as it is completed to avoid a wiring nightmare later!

Hook up each switch to the Arduino as you go along. The multiplexer has a built in ground next to each input, which is really convenient.

To wire the pullup resistors to the built in pins, connect one resistor end to 5V and one to the pin you're using. Then, skipping the resistor, connect one end of your switch to ground and the other directly to your pin.

Find a comfortable chair, because this is going to take a while!



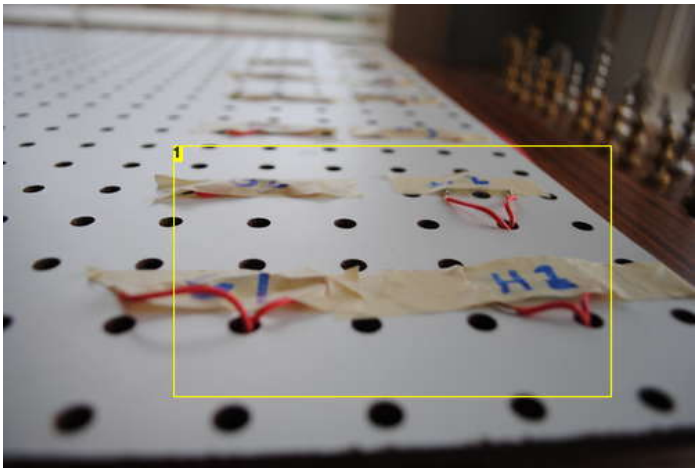


Image Notes
1. The holes in pegboard are at every square inch.

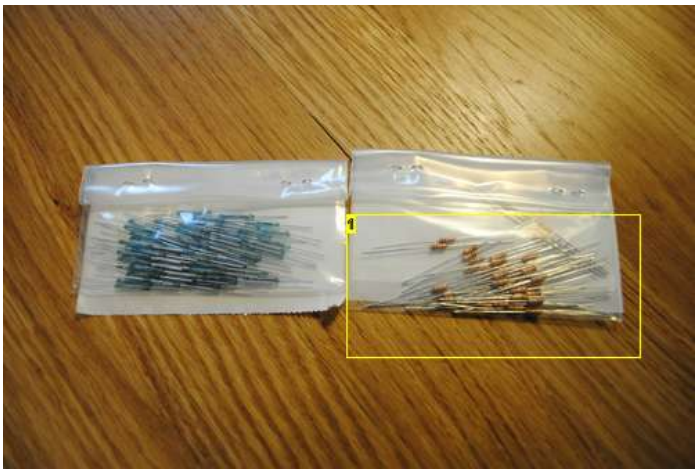


Image Notes
1. The mux shield has resistors built in. You only need 16.

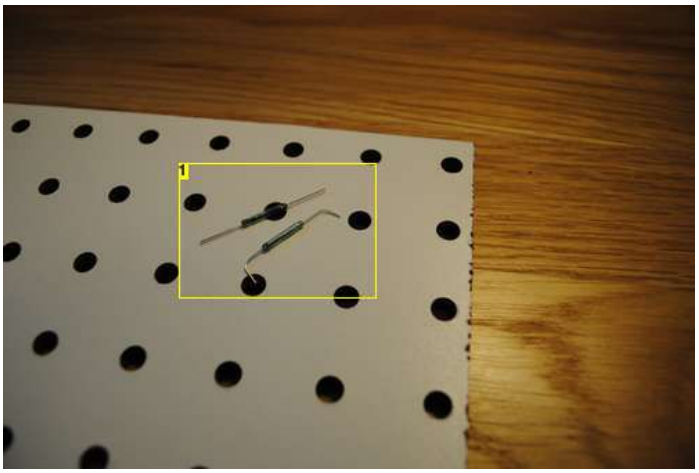
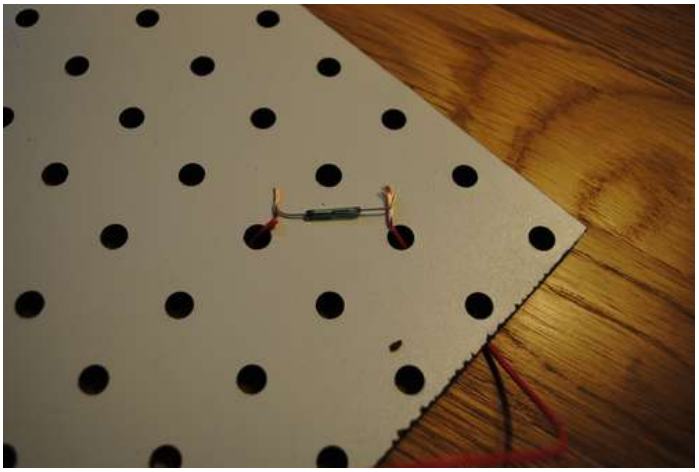
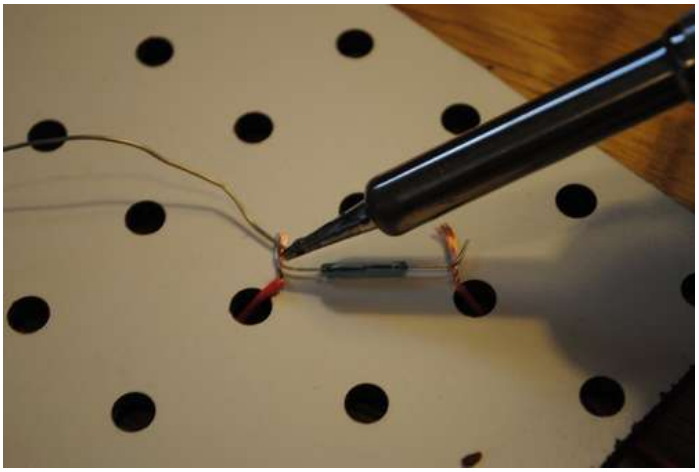
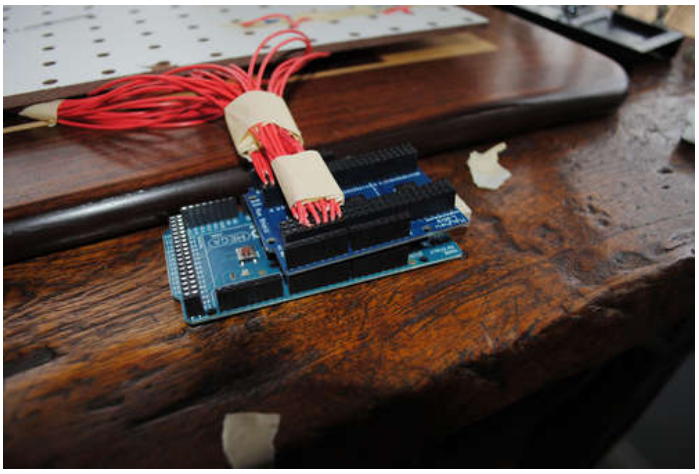
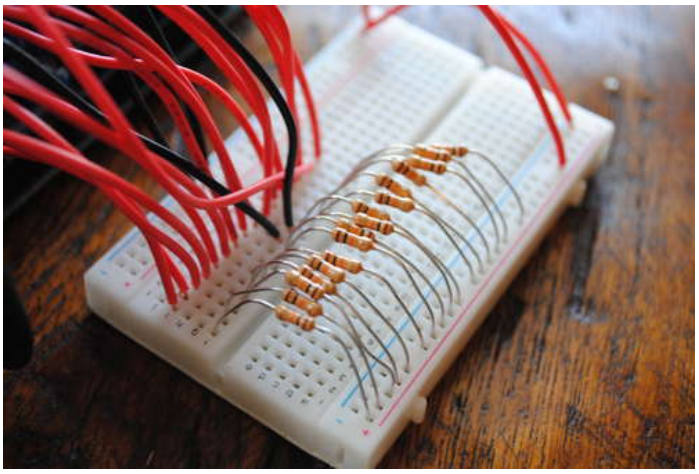


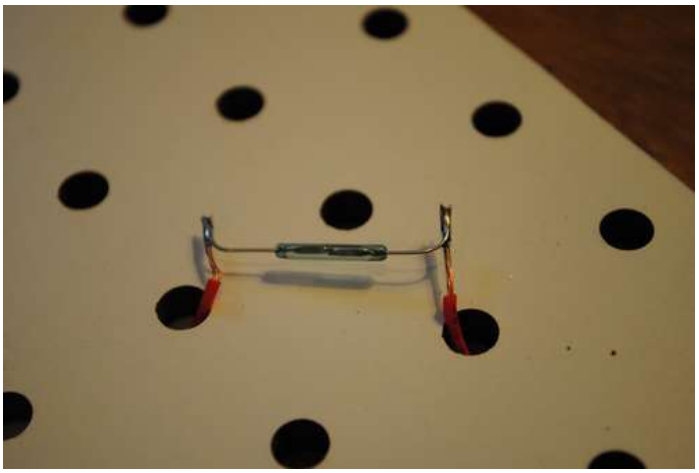
Image Notes
1. Be careful not to break the glass when bending the leads.

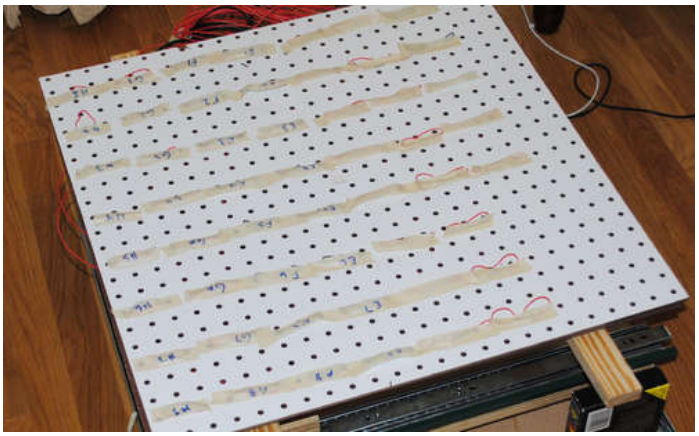












Step 12: Place the Magnets

Each chess piece has a magnet embedded in its bottom to trigger the reed switches. The only thing that's important here is that every piece on one side of the board has one polarity (like North), and every piece on the other side has the opposite (like south). If you'd only like the computer to play one color, make sure it is the opposite of the magnet attached to the servo, so they will attract each other.

Other than that, though, this only takes a few minutes.

Step 13: Code, Final Assembly + Reflection

The code for this project is surprisingly simple. It takes the reading from the chess board out of terminal, plugs it into a chess algorithm already built into Mac OS X (look inside the Chess.app bundle), and spits out the coordinates back into the Arduino window with some fancy applescript. The algorithm is a fantastic open source project called Sjeng , meaning this will work cross platform as well.

The final assembly for this project is fairly simple. Find something to hold up your sensor grid, and lay that down. Next, place your chess board and pieces on top. You're done! Run the software and give it a whirl.

If you ever get bored of chess, it's really easy to make this into a CNC project. Replace that magnet with an X-Acto knife, flip the entire assembly upside down, and you have a stencil cutter! Replace that X-Acto knife with a pencil and you have a draw-bot. Modify the servo to activate a dremel and you have a CNC machine! The possibilities are, please excuse the cliché, endless!

Reflection

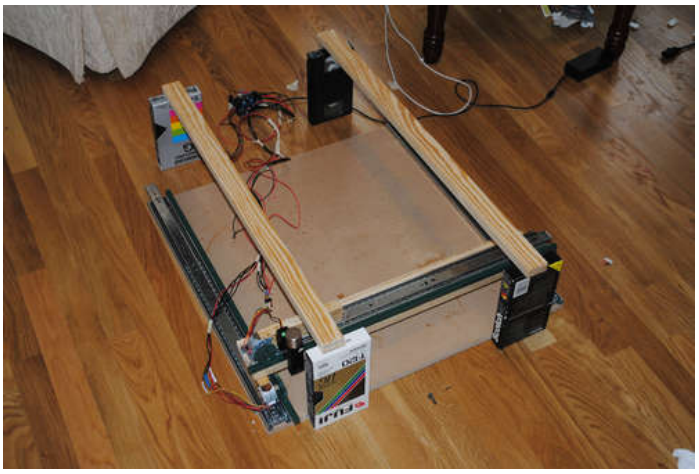
There are changes I would make to this project if I revisit it. First, the mounting of the gears should be improved; a few times the gears would pop off due to the heating of the stepper motors. Proper mounting with screws would fix this. Second, the board gets (fairly severely) out of calibration after a few moves, presumably due to skipped steps, and must be manually moved back to the origin to avoid misalignment. Some potentiometers allowing the board to know its absolute position would fix this. The rails, despite my best efforts, were misaligned, making the last few rows of squares very difficult for the stepper motors, causing them to skip steps and click loudly (bigger steppers would be better). Sometimes the motors would stop altogether and wouldn't be able to make it in those rear squares, rendering a game unplayable without human intervention. Aligning the rails better would fix this. The magnets are also waaaaaay too powerful, and often pull pieces in they shouldn't. This renders games unplayable if you're not actively helping the board out, so weaker magnets really are a must. Often what would happen was that when the large magnet flipped over, a whole bunch of pieces would get drawn towards it (and it's a real pain to place all the magnets back inside the pieces if you haven't attached them). If you don't switch to a smaller magnet, this will happen pretty much every time. The ones underneath the pieces are mostly fine - it's just the main magnet that is way too strong. Ceramic magnets like the ones people stick to whiteboards would work far better. Finally, the code could definitely be streamlined further; I completed this project when I was relatively new to C and Arduino. I'm sure there's a better way to do this than having an applescript copy and paste into the Arduino environment!

Thoughts on the Epilog Challenge

It's kind of a funny feeling when as a maker you have something that can, well, build other things. An epilog laser would help me in some way with literally everything I make. With this laser I would, on a regular basis, cut solder stencils for surface mount work, make more elegant housings for my projects, and engrave logos onto personal laptops and cell phones. As a student, it would provide me with a small extra source of income from engraving the gadgets of others. Rapid prototyping circuit boards would allow me to take my work to the next level, rather than having to spend several hours etching a PCB just to find out one trace is routed incorrectly. This laser, combined with help from the awesome community here on Instructables, would help me so much in my pursuit of a career in engineering, my ultimate goal.

I'll leave it up to the merit of my Instructable, though, to decide whether I deserve this laser, not the list of sappy thing's I'd do with it. :D Instructables and Epilog have done something great here, and the projects created by this fantastic contest will set a standard of quality for years to come.

I hope you've enjoyed reading about this project as much as I enjoyed making it. Make sure to leave a comment if this has inspired you to build anything, I'd love to see what you've made!



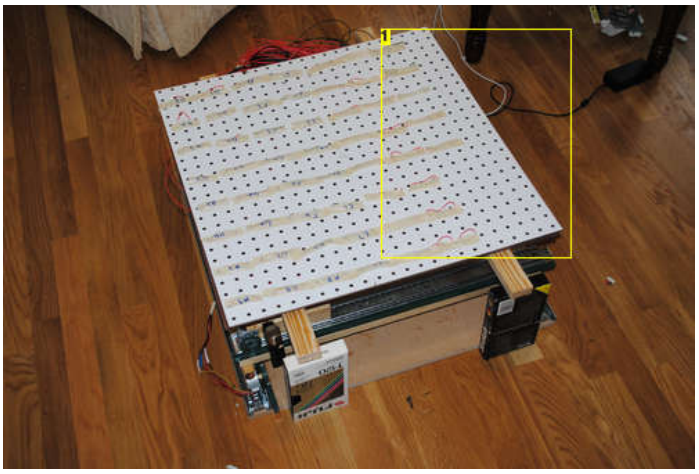


Image Notes
1. The sensors are not completed in this photo but need to be before putting on the chess board.

File Downloads

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

Related Instructables

Sandbox Project is now Online (Photos) by CarlS

Steampunk-Inspired Hardware Chess Set (video) by sparkyrust

Hard Drive Fridge Magnet by Everfalling

Build Laser Cutter (Photos) by fishy8082

Homemade 2'x4' Wood CNC Router by Kyles Woodworking

Internet Arduino Controlled T-Slot XY Table by CarlS

Full Contact Chess by KRA5H

Chess Board Safe With Hardware Chess Men (Photos) by Data643

SITWAY

by **mickydee** on February 7, 2012

Author:mickydee

My name is Roland MacDonald, my friends call me Mac
I am a retired but not bored engineer
My great joy is my workshop, a two car garage with heat and air.
I am a private pilot with 1800 hours flying time
I have built four Kit planes and restored two Cessna aircraft
My last contact with electronics was in the vacuum tube world.
I am really enjoying the new transistor world.

Intro: SITWAY

You are never to old to learn and try new things. I think one of the best days in my life was the day I discovered the Instructables web site. It opened up a whole new world to me. This is my third instructable . I really enjoy building anything that I can ride on or get in to.
I bought an Arduino Uno and was planning on building a balancing Robot. I was really impressed with the Balancing Skate Board that was published by Xenon John. It had most of the code that I would need to build a balancing something. That something evolved from a Robot to a Sit Down Segway clone, which I named the SITWAY. I want to at this time thank John for all the help and patience he showed me in building and testing this ible.

This is my second project involving a discarded electric wheel chair. The motors have great torque and are very reliable. They use 24 volts and have great range using two U1 type garden tractor batteries. You can't go any cheaper than that.

The build went pretty smoothly. Thankfully Xenon John pitched in and helped me modify his code to work with my wheel chair motors.
After running all the tests I felt were needed I elected to have a young neighbor take the first ride. It turned out to be a real blast. So far eight or ten people have ridden it, the youngest being 12, and the oldest 81 (me). The training wheels limit the speed by limiting the forward tilt. I plan to keep the rear training wheels on permanently because I don't need a lot o speed going backwards.

The SITWAY appears to be pretty safe, but it does not have all the built in backup systems that a real Segway has, I have only tested it on my smooth driveway at this time. I have driven over small objects, and it still stayed stable..Any one can learn to drive it in about 5 or10 minutes. With all the testing and driving we have done I have yet had to charge the batteries. The original wheel chair had a published range of 20 miles.. HAVE FUN!!!

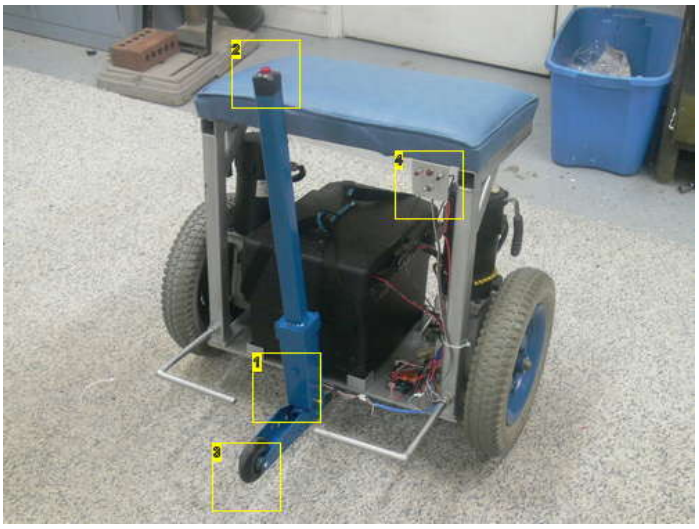


Image Notes
1. Pivot bolt. and tapped screws to adjust micro switch contact points
2. Dead man switch
3. Adjustable front wheel
4. Main power and balance trim switches

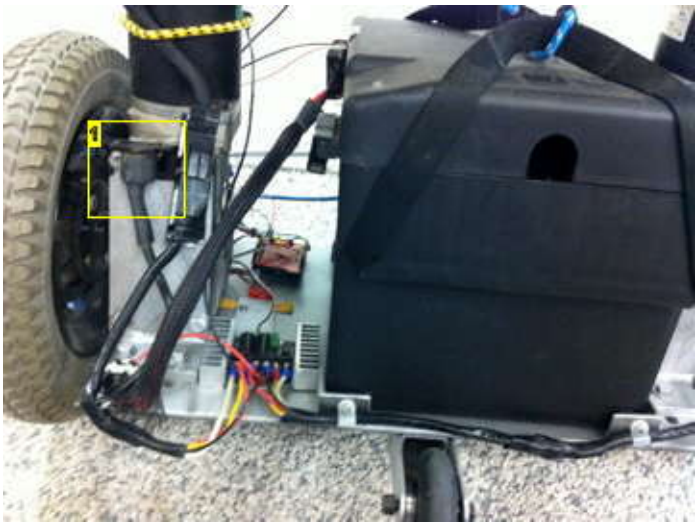


Image Notes
1. Optional 24 VDC charging plug

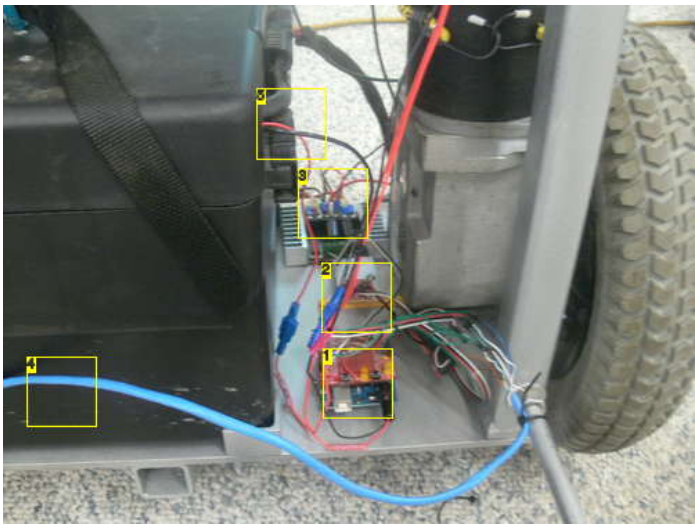


Image Notes
1. -Arduino Uno and shield
2. 5 degrees of freedom IMU
3. Sabertooth 2X25 Motor Controller
4. Cable to hand controller
5. 12 VDC tap from batteries

Step 1: MATERIALS AND COSTS

Item Source Cost

1. Donor electric wheelchair Various places $50 to $200 Depending on condition
2. Arduino Uno Maker Store $30
3. Arduino Uno Proto-shield Spark Fun $15
4. 5 degrees of freedom IMU Spark Fun $49
5. Sabertooth 2X25 Dimension Engineering $129
6. Two surface mount LED's Radio Shack $1.29
7. Two Momentary contact switch's (trim) Radio Shack $1.29
(normally open)
8. Two micro switches for Steering Radio Shack $3
(normally open)
9. Single pole single throw power switch Radio Shack $1
10 13X20X1/4" plate for base Local $10
11. 1/2" steel tubing for seat frame Local $10
12. Vinyl and foam for seat Local $4
13. Asst hook up wire Local $3
14. One can spray paint Local $3
Total $309 to $459

(note) I actually bought my used wheelchair for $35 at a yard sale
It was pretty beat up but the motors were good and even included
A 24 volt charger.My project cost less than $300.

Step 2: Salvaging parts from the donor wheelchair

Save the motors and drive wheels, they are usually one piece. Also salvage the connectors and wiring from the motors to the battery., leave the leads as long as possible. Keep the electronics if you want them for future use. Mine were trash. I never throw away wheels, they always come in handy. Most wheelchairs have two castoring wheels for steering, and two for stability when getting on the chair. Save the two stability wheels for training wheels on the project. Save the battery box and battery cover, you will use these. Dis-card the rest of the chair unless you think you will have use for it in the future. I threw most of it out to reduce clutter.

Step 3: Build the frame and mount the wheels and motors

I guess you could make this frame out of plywood, but I like to use steel. it's a lot stronger and welding is a lot of fun. The base is a 1/2" plate measuring 12X20". The uprights and seat frame is made of 1" steel tubing. Don't forget the 45 deg. braces in the corners. the four holes in the seat braces are for mounting the plywood seat support. The holes for mounting the motors are slotted about 6" long . At this point you do not exactly know where the C.G. will be. The motors can be adjusted fore and aft to adjust the C.G.The small wheels are used as training wheels and to keep the machine from falling over when not in use. Now is a good time to paint the frame You can make the frame any size you want. I designed this one to fit through an interior door. Cut the seat from 3/4" plywood . Pad and upholster a seat cushion to be bolted to the frame uprights.

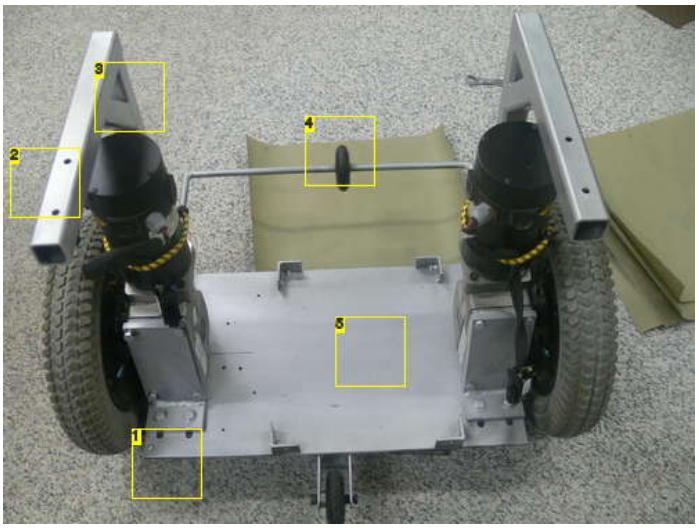


Image Notes
1. 6" slotted holes to mount motors and adjust the C.G.
2. holes to attach seat support
3. 45 deg. braces
4. foot rest and leveling wheel
5. battery position

Step 4: STEERING CONTROLLER

I originally used John's hand held steering controller, but found it was not practical for a sit down balancing machine. You need something very rigid to hang onto when riding. The machine does a good job of balancing if you don't fight it by trying to balance it your self..
The stick is rigid in the fore and aft position , and will move side to side in the lateral position. a compression spring centers the stick.
Two set screws provide stops, and two more act as limit switches for the two micro switches that control the steering.
This could be done simpler by mounting the micro switched on the outside of the stick, but I wanted to have them hidden inside the stick.
I have access to a vertical mill and I never miss a chance to use it. The micro switches are available from Radio Shack for 3 or 4 dollars.

The adjustable front training wheel serves two purposes. first it keeps you from pitching forward during any sudden stops, and secondly it limits the forward speed by limiting the pitch angle until you get comfortable with riding it. The rear wheel is fixed limiting reverse travel to a safe speed.

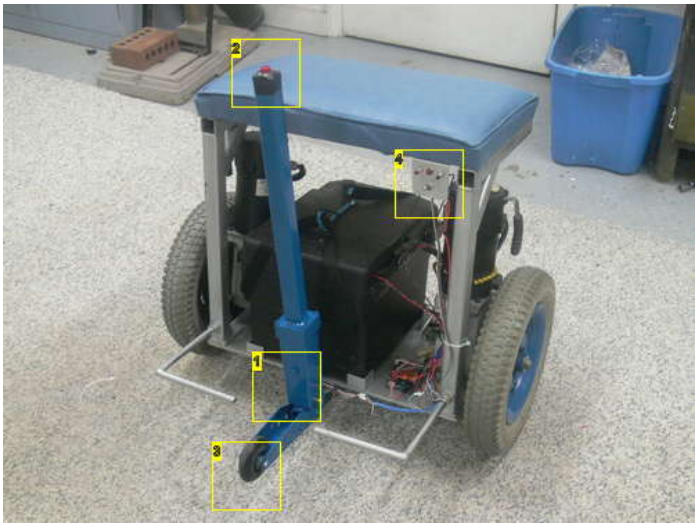
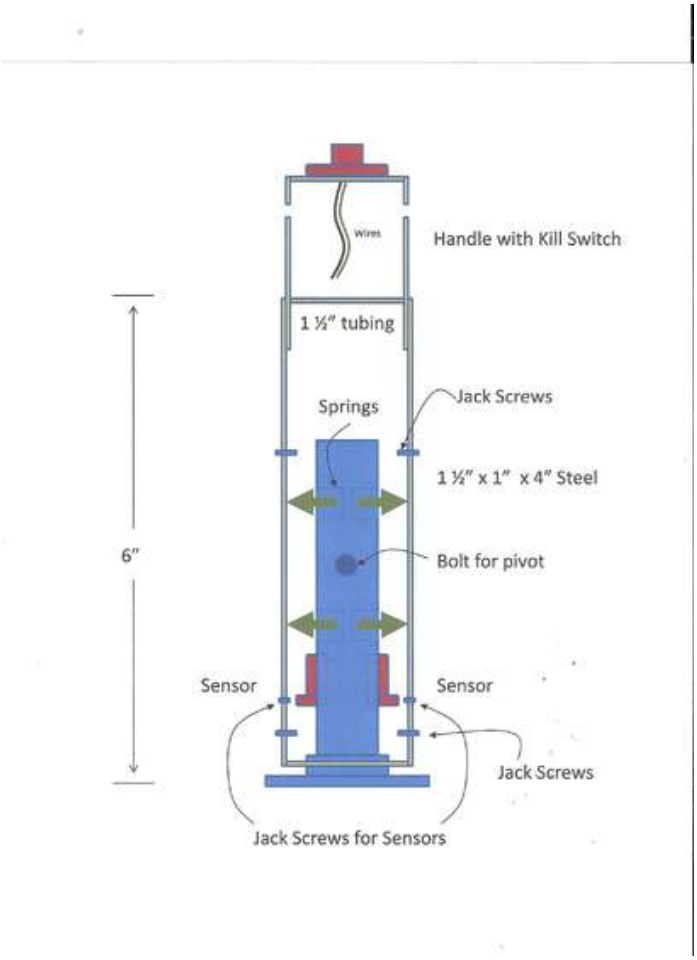


Image Notes
1. Pivot bolt. and tapped screws to adjust micro switch contact points
2. Dead man switch
3. Adjustable front wheel
4. Main power and balance trim switches



Step 5: ELECTRONICS

The electronics consist of the following
Arduino Uno
Shield
Sabertooth 2X25
5 Degree's of freedom IMU
Asst. LED's and switches
10K pull down resistors (5)
4 conductor cable and hook up wire

I don't like to solder directly to my Arduino. Instead I used a shield. This allows me to make solid solder connections instead of plugs that can come loose due to handling or vibrations.

A good place to begin is to solder the (5) 10k resisters to the shield. These are the pull down resisters for the balance trim, steering, and dead man circuits.

(Note) The following wires are connected to the Arduino digital pins
pin 9 is for the dead man switch circuit
pin 7 is for nose down trim circuit
pin 6 is for nose up trim circuit
pin 5 is for steer left circuit
pin 4 is for steer right circuit
The other end of the resisters goes to circuit ground
pin 13 connects to the S1 input of the Sabertooth Motor Controller

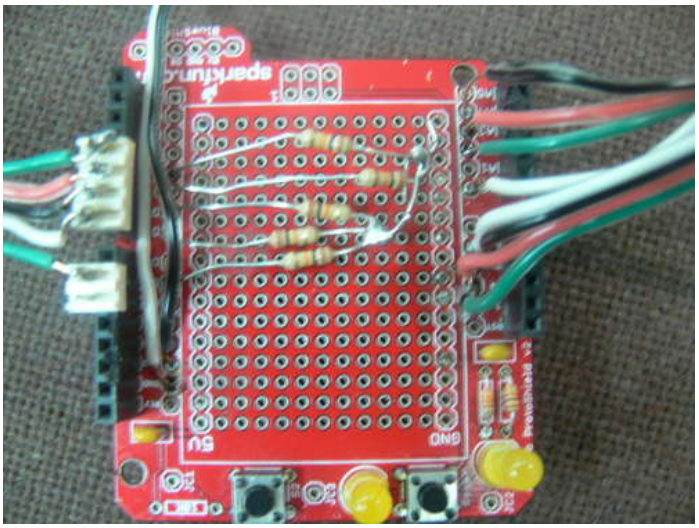
The following wires are connected to the Arduino Analog pins
pin 0 to Y Rate 4.5 on the IMU
pin 2 to X Rate on the IMU
pin 3 to Y Rate on the IMU
pin 4 to ZACC un the IMU
+5 volts to he Steering controller
+3.3 volts to the IMU (NOTE) do NOT apply 5 volts to the IMU
GND to the IMU

All the Analog connections can be soldered directly to the shield.
The Digital connections can be made either by plugging directly into the headers or using a connecter. I found some in the Sparkfun catalog that fit snugly into the headers (See above pic)

I found some four conductor ribbon cable at Radio Shack that worked well for me. It is stiff enough to hold it's shape, and the color coding makes life easier. You can use ribbon cable from an old computer just as well except for the color coding .

Mount the IMU to a small block either wood or phenolic to the floor of the machine at approximately the center line of the axles.
Be sure to mount it correctly.
The factory drilled mounting hole must be pointed towards the ground
The component side of the IMU must face forward
If you mount the IMU incorrectly bad things will happen.
The plate can be shimmed fore and aft to adjust for level balance

Fashion an aluminum plate to hold the power switch, and the trim and steer switches and mount it to be reached conveniently while seated. I installed a power indicator LED (with a Pull down resister) to verify Arduino power (12 volts)



Step 6: WIRING

Generally wheelchairs operate on 24 VDC.. Main power is obtained from two U1 type lead acid batteries. They are used in lawn mower or lawn tractors. They are cheap ($40) and easy to get .
There are four voltages used in this project
24 VDC for the motors
12VDC for the Arduino Uno
5 VDC for the steering and trim circuits
3.3 VDC for the IMU
24 VDC is connected to the Sabertooth 2X25 . A power switch is installed in the negative leg. Be VERY careful to maintain the correct polarity.
The Sabertooth will be permanently damaged if you hook it up with the wrong polarity. The warranty will also be voided.
The 12 VDC is obtained with a tap between the two batteries terminated with a plug for the Arduino power input. Do not use the USB circuit as a power source, strange things happen to the gyro when you use the USB for power.

The 5VDC and 3.3VDC is obtained from the Arduino Uno.
A 5 volt source powers the dead man, left and right turning, and both trim switches.
3.3 volts from the Arduino powers the IMU

Bolt the Sabertooth, the IMU mounted to a block, and the Arduino Uno and it's shield to the floor of the frame
Connect the motors to the Sabertooth. The left motor connects. to M1A and M1B. The right motor to M2A and M2B.
S1 on the Sabertooth connects to Arduino pin 13. Connect Sabertooth ground to Arduino ground. This completes the Sabertooth wiring for now. These connections will be verified during the Motor test procedure a little later. Be sure and set the Sabertooth DIP switches for Simplified Serial operation. Set switches 1,3,5, and6 to the on position Switches 2, and 4 are set to the off position. These settings support using lead acid batteries.

Route the five wires from Digital pins 4,5,,and 9 plus a 5 VDC source to the steering handle.
Connect the 5 volts to one side of the two momentary on steering switches and to one side of the dead man switch.
Connect the other side of the switches to the wires coming from Arduino pins 4, 5, and 9.

Route the wires from Arduino pins 6, and 7 to the little plate with the trim switches. Connect one side to 5 VDC and the other to Arduino pins 6, and 7.Route the wires from the main power switch to this plate and attach them to a 40 amp SPST switch. Install a surface mounted LED indicator and power it with 5VDC to ground through a 10k resistor.

Install the batteries using the battery box or covers you salvaged from the wheelchair and this pretty much completes the basic construction of the project.. If you saved the charging plug from the wheel chair, install it in a handy place and wire it up to the 24 VDC source. Pad and upholster a seat cushion to be bolted to the frame uprights.

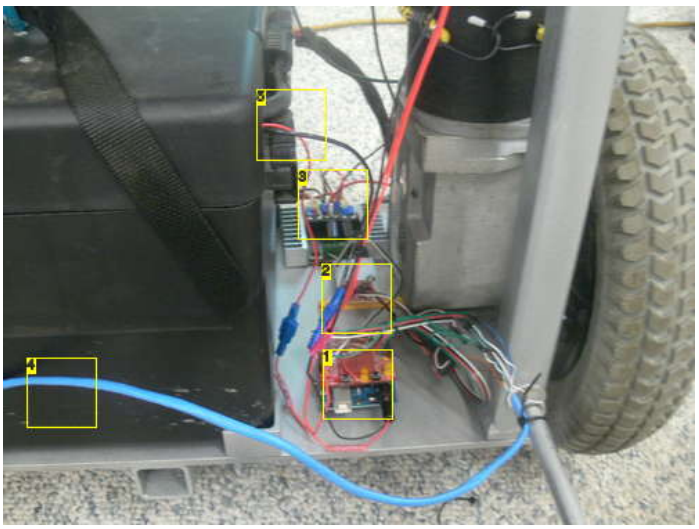


Image Notes
1. -Arduino Uno and shield
2. 5 degrees of freedom IMU
3. Sabertooth 2X25 Motor Controller
4. Cable to hand controller
5. 12 VDC tap from batteries

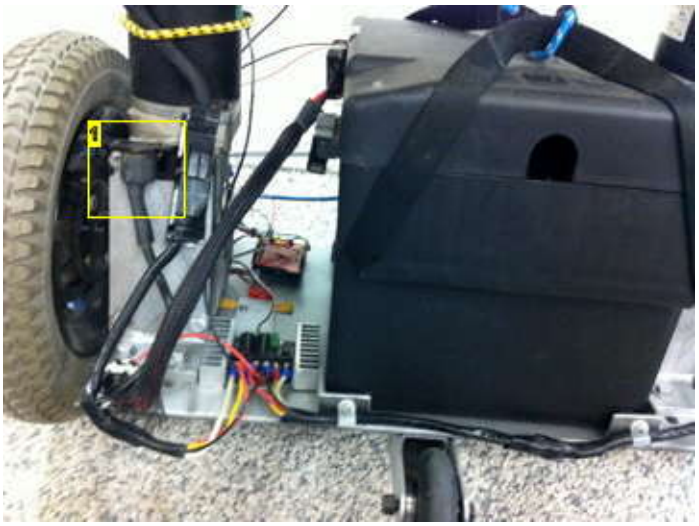
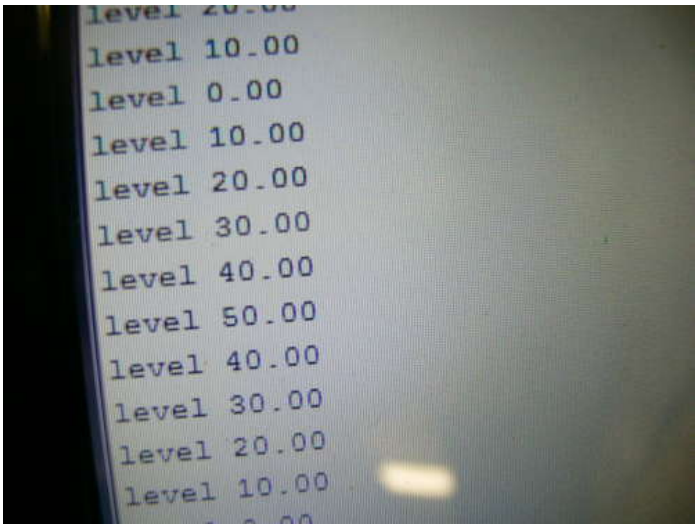
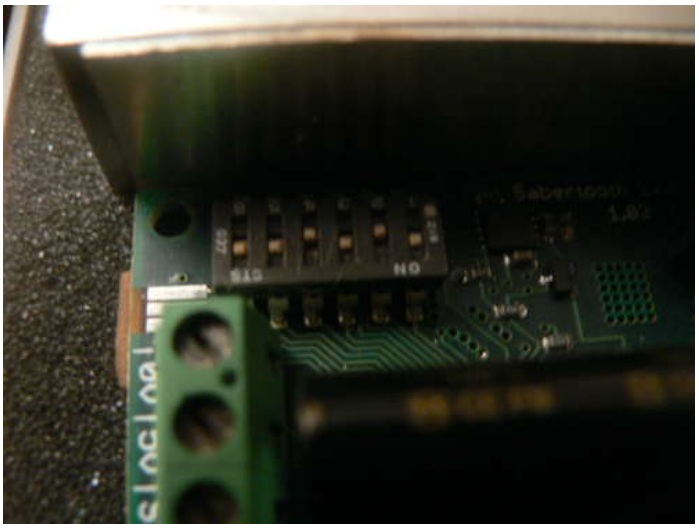


Image Notes
1. Optional 24 VDC charging plug

Step 7: MOTOR TEST

Now that the wiring has been completed, it is time to test the communication between the Arduino and the Sabertooth.
IMPORTANT!!! Maker sure the Sabertooth DIP switches are set for Simplified Serial operation.
Raise the machine up on blocs high enough for the wheels to clear the floor and can spin freely.
Open the Motor Test notebook sketch. Copy and paste this sketch to a new Arduino sketch. Upload the new sketch to your Arduino.
This sketch only uses digital pins 9 and 13. (Dead man and serial input to the Sabertooth)
Turn on the main power switch and depress the Dead man switch. Both motors should start turning in the same counter clockwise direction. If they don't, reverse the wires going to the M1 or M2 connections on the Sabertooth. Both motors should slowly increase in speed in 10% increments from stop to 50% and then from 50% to stop.





File Downloads

new_motot_test.txt (5 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'new_motot_test.txt']

Step 8: THE FIRST TEST RIDE

Now that you know that the motors are turning in the right direction, and communicating with the Sabertooth, it is time to upload the full rocker switch code to the Arduino.. Copy the full test sketch into a new Arduino sketch and upload it to your Arduino.
This is a good time to exercise extreme caution as these motors are very powerful and can run across the room and do bad things.
This code makes use of a TIP START routine that lets the machine start gently at first and then after about 5 seconds comes up to full power.. With the machine tipped to its parked position turn on the master switch. Count to five slowly. This allows the IMU to calibrate itself.
Next depress the dead man switch. (Note) I installed an LED to light up when pin 9 the dead man circuit is activated. Slowly bring the machine to an approximately level position. If all goes well the machine should balance it's self. Any time you sense a problem let go of the dead man and the machine will stop. If you De-activate the dead man you will have to start from the beginning and go through the TIP START routine again. Make these first tests with out getting on the machine. Tip the machine forward and backward a little and see if the machine tries to move accordingly Try a gentle turn in both directions. If all goes well you are ready for the first test ride.

Do not take the first ride when you are alone. Have some one there to help you if something goes wrong. At my age my bones are getting pretty brittle so I let my best friends teen age son do the honors. We made him wear a helmet and we brought him up to level after the TIP START by hand. He did a great job and mastered the machine in just a couple minutes.
For this first test we used the hand held controller that Xenon John uses in his skate board. It became obvious that for the SITWAY we would need a different control method. I built the new joy stick out of 1" steel tubing. Like the original Segway you need something to hang onto. The joy stick houses the two steering switches and the dead man circuit. The stick does not move in the fore and aft plane, the only movement is from side to side. This system really works well and has not given me any trouble. I moved the trim switched to a little aluminum plate mounted whitin reach of my left hand.

File Downloads

full_balance_rocker_test.txt (25 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'full_balance_rocker_test.txt']

Recent_mild_setup_code.txt (29 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Recent_mild_setup_code.txt']

Step 9: ADDING 3D PRINTED OBJECTS

To spruce up the project I decided to add some caps and plugs to the frame to make it look a little better' The deadman switch is round and did not fit very well into the square hole of the joystick.. I designed a cap to attach it using "123D". I also printed out four plugs to cover the holes from the 1: tubing forming the frame. Not really necessary, but it looks good.. I am always looking for good uses for the 3D printer.

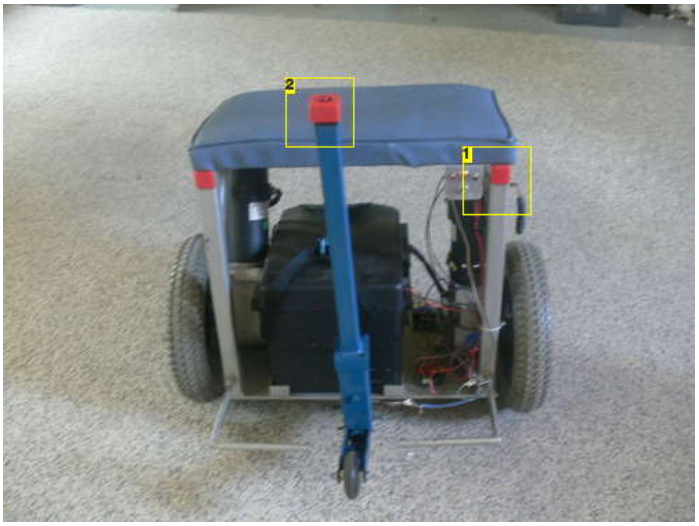


Image Notes
1. 1" tubing
2. Deadman switch

Step 10: CONCLUSION

I built the SITWAY entirely out of steel, mainly because I had the steel available and I love to weld. It could be built out of wood, but I don't think it would be quite as substantial.. This is an on going project. I am still tweaking the code to get the smoothest performance. I will publish improvements as the are ready. The present code will at least assure that the machine will Balance, Move, and Steer.

This version is a little slow at this time and that is alright by me for now. I plan to raise the front wheel a little for more forward speed.
Eventually I might eliminate the front training wheel all together when my confidence factor builds up.

The most important to learn is that the machine should balance you, not the other way around. Try to remain in a stiff position and let the machine do the work instead of fighting it.

SAFETY SAFETY SAFETY I take baby steps with any changes I make If you try to go too fast, this thing could hurt you. There is a lot of power with these motors. I have not tried to navigate over any large objects as yet. I plan to test it by running it over a 2X4 and see what happens.

It is a fun project. The neighborhood kids are crazy about it and zip around the yard with ease. Thar's the main reason I have limited the speed for now. The wheel chair motors are very frugal with battery use. I have actually run on 12 VDC. It was a little sluggish but operated just fine.

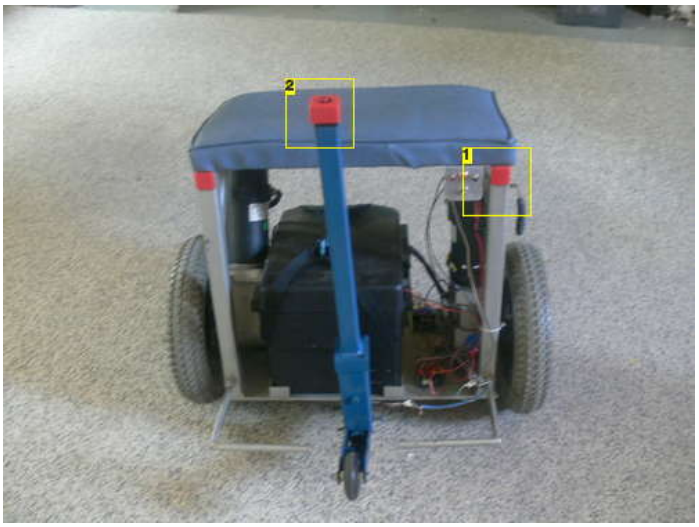


Image Notes
1. 1" tubing
2. Deadman switch

Related Instructables

Easy build self balancing electric skateboard by XenonJohn

Self Balancing Scooter Ver 1.0 by ScitechWA

Balance-BOT (Photos) by daniel2008

Steampunk Segway (Legway) by bdring

Domo Kun WobblyBot, Simple Self Balancing Robot by Chein

LEGO iPod touch dock. by Gun Fun

Seg...stick. by scolton

Polystyrene throw airplane by njacksx

A Makers Wedding - Photo booth

by **letMeBeFranks** on November 5, 2011

Author:letMeBeFranks
Im a interaction designer. I work at frog design :)

Intro: A Makers Wedding - Photo booth

This Instructable is about:
building an automated photo booth. The total build cost was around $150 as I re-used a lot of the components and materials I already had in my garage - in addition to what I could salvage from scrap yards.

Why? - I decided to build my own photo booth after trying to rent one from local photography studios. The going rate for a rented photo booth is around $600 in addition to the hourly rate of the attendent to watch over the equipment. As this was not in my wedding budget, and I did not want to deal with an additional vendor, I decided to build my own for under $200.

My Goal - Build an automated photo booth for under $200 - that could be easily operated by anyone at a party - and is durable and compact enough to fit into a compact car.

(Note* this photo booth does not print pictures. I have been working on a script that automatically uploaded the photo to flickr, but I did not finish it in time for the wedding. I'll try and include that in a future post)

Step 1: How it works

The photo booth operation is simple. Users walk up to the back side of the camera - See themselves on the screen - Press a button and strike a pose.

The mechanics of the photo booth are a little bit more complicated, but ideally, the user never has to know what is going on under the hood.

The guts behind this photo booth are based on OSX lion. With Lion, the photobooth application can be extended to full screen and it can be set to use an external camera. So I connect a logitech web cam and an external monitor to a laptop running OSX Lion. The only thing i needed to build in addition to this hardware setup was an array of lights and a button (mapped to the enter key) to trigger the photobooth application to take a picture.

The "business end" of the photo booth can be seen in this step.

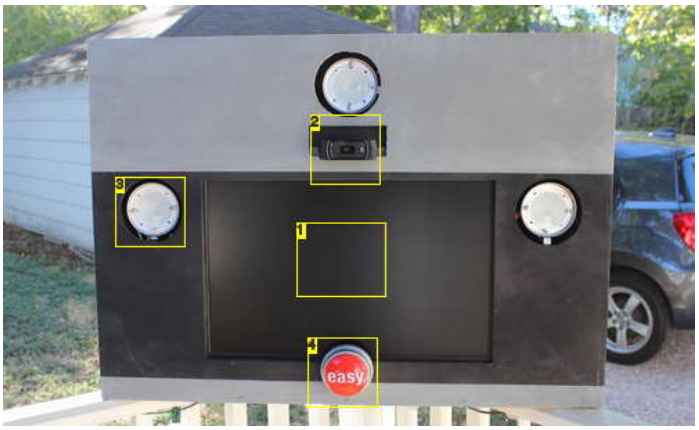


Image Notes
1. LCD Monitor
2. Logitech Webcam
3. Halogen Lights from Ikea
4. Modified Staples Easy Button

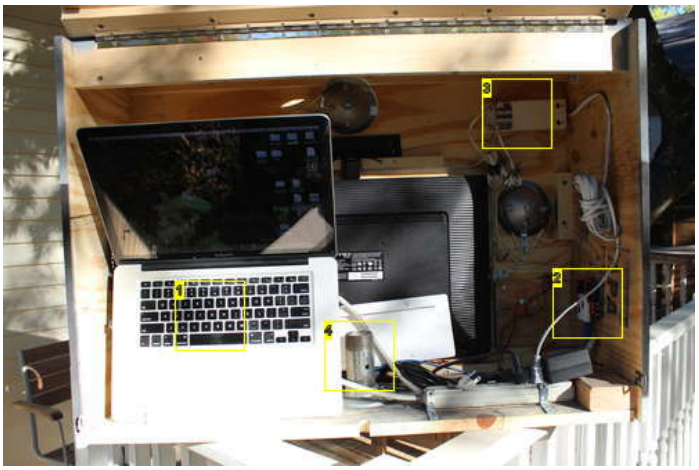


Image Notes
1. MacBook Pro - OSX Lion
2. Arduino
3. Light Controller

4. Tripod Mount

Step 2: Software and Trigger Button

A Brief Overview

As previously mentioned, this photobooth uses the OSX Photobooth application. The OSX Photobooth application was chosen because it was the most stable software i could find - and it comes with every MAC computer. Like most applications, users can trigger features and functions with mouse clicks and keyboard commands.

Triggering the Photobooth Application

With OSX Photobooth, pressing the Enter Key triggers the program to take a photo. I didn't want to expose my computer to people hammering on the keyboard (espeically if they had been drinking). This is why i decided to use an external button, connected to an arduino microcontroller, to trigger the photobooth application.

This is how it works:

The button is pressed - A Staples Easy button was modified to act as a regular button. It's really durable, so people can beat on it without breaking it.

An Arduino registers the button press - When it registers a button press, it sends a serial command to the computer. In this case, it sends the [enter] serial command.

AAC Keys listens to the serial port for serial commands - AAC keys is a free application which litens for serial commands and emulates mouse and keboard events. You can download it here. In this case, when AAC keys receives the [enter] serial command, it tells the computer (and the photobooth application) that someone has just pressed the enter key on the keyboard.

When the photobooth application registers the enter key being pressed, it takes a photo.

Wiring the circuit - If you do not know how to make a button circuit for an arduino, read this tutorial - http://www.arduino.cc/en/Tutorial/button

Be sure to connect the button to pin 10 on your arduino. If you choose to wire your button to a different pin, be sure to change [int buttonPin = 10] in the arduino code to match the pin number you selected.

Writing the code - Here is the code i wrote to send an [enter] serial command to the AAC Keys. If you are not familiar with writing arduino code, use this tutorial here. It's pretty easy once you get the hang of it. http://arduino.cc/en/Guide/HomePage

```
const int buttonPin = 10; // the number of the pushbutton pin

int buttonState = 0; // variable for reading the pushbutton status

void setup() {

pinMode(buttonPin, INPUT);
Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop(){

buttonState = digitalRead(buttonPin);

if (buttonState == HIGH) {
Serial.println();
}
else {
// nothing
}

}
```

Installing AAC keys - As previously mentioned, AAC keys is a free program. "That receives commands through your computer's serial port and translates them into keystrokes and mouse movements, giving you full control of your computer from another device such as an [arduino]". You can download the program here: http://www.oatsoft.org/Software/aac-keys

Using AAC Keys is quite simple. Make sure you have an arduino plugged in via usb, running the code seen above. Open AAC keys application and access the applications preferences. When the dialogue appears, check to see that you have selected the serial port associated with the connected arduino (generally it's selected by default, but it is good practice to check), and that it is running at 9600 bps.

If you've done this, AAC keys should be interpreting the button press from the arduino as an [enter] command on the keyboard. open a text editor and give it a shot. Type a few lines of text and press the button attached to your arduino instead of using the enter key. You can also open photobooth at this time and see that pressing the button triggers the program to take a picture.

Image Notes
1. Testing the connection between the easy button and the computer.

Image Notes
1. Disassembled Staples Easy Button

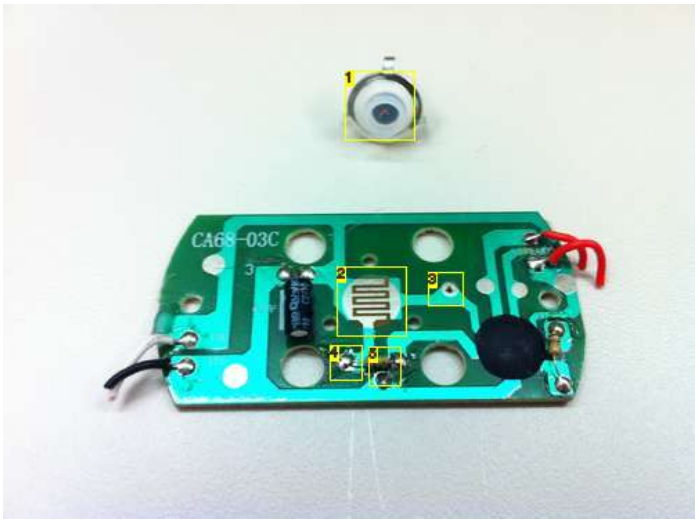


Image Notes
1. Rubber button which makes contact with the circuit board

2. Circuit board from the staples easy button. Inorder to use this button to trigger an arduino, Solder wires to the following locations that are labeled in this image.
3. Solder one wire here
4. Solder Another Wire here
5. Cut this resistor off the board. This prevents any errors (false trigger events) as this pathway on the circuit board is still connected to the original microprocessor (seen as the black dot in this picture)

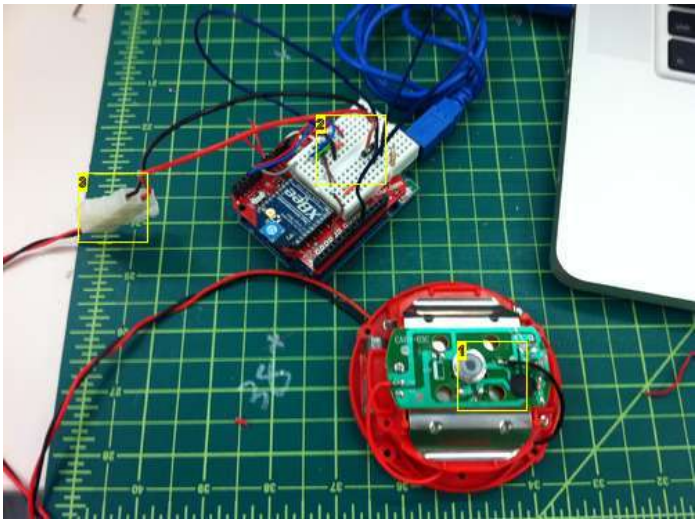


Image Notes
1. wires soldered to the circuit board from the staples easy button. These wires lead back to the breadboard on an arduino
2. Arduino registers when the button has been pressed. It then sends a serial command to the computer.

The button is connected to Pin 10 on the arduino
3. I added a quick disconnect so that i could disconnect the button from the arduino when it came time for installation.

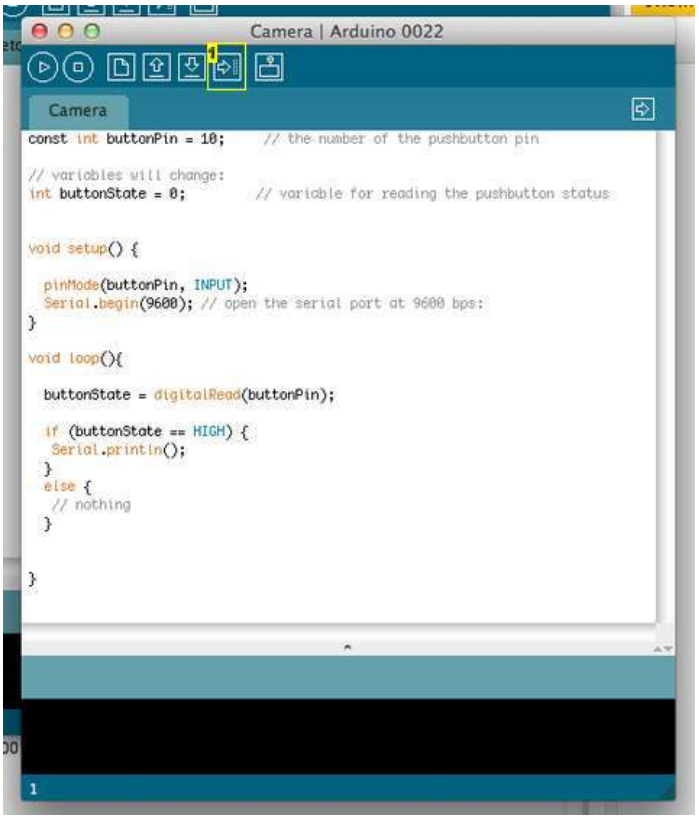


Image Notes
1. This is the arduino code in the editor. Click this button to send the code to the attached arduino

Step 3: Booth Design

The photo booth was modeled after an old school LOMO camera, instead of the traditional box with a curtain, for three reasons:

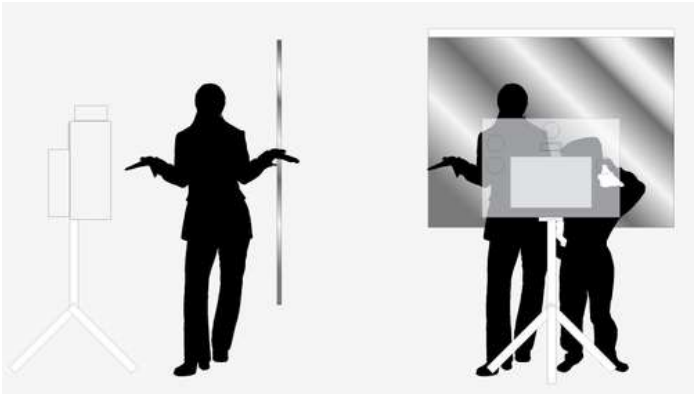
- Ease of construction - Its basically a box with a fake lens on it
- Recognition - People will be able to see a large camera from far away, and they might easily deduce that it must take photos in some form or fashion.
- Novelty - The accentuated size of the camera will create a conversation piece in addition to eliminating the fear of social contract (the users fear of approaching and using it without permission or instruction)

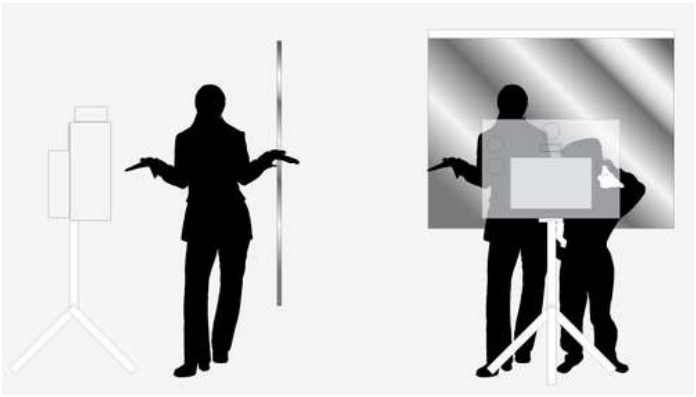
After settling on the design, I sketched it out in adobe illustrator (the .AI file is included on this page). Illustrator is a good tool to make quick "blueprints" which can be easily scaled and printed. if you don't already own adobe illustrator, you can get the demo right here http://www.adobe.com/cfusion/tdrc/index.cfm?product=illustrator

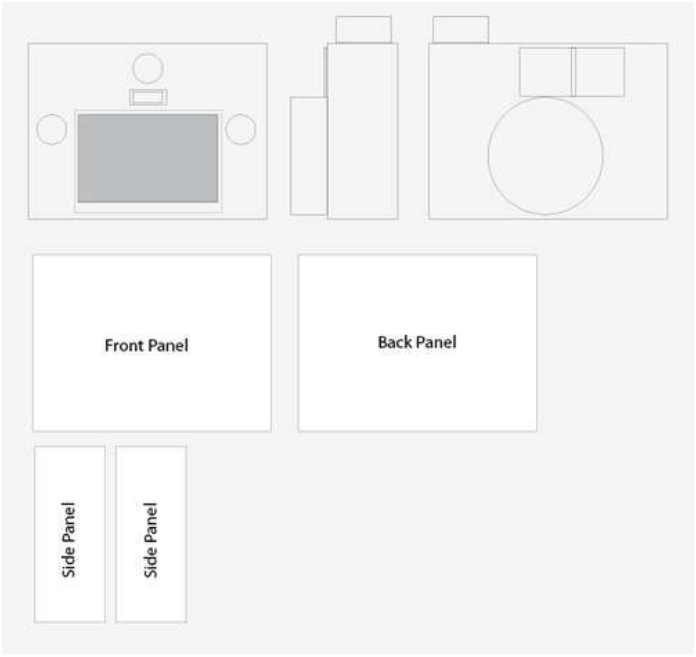
With this quick blueprint, I created a front and side view of the camera. The size of the camera is dictated by the size of the LCD monitor and a a varying height range of users - something that would be easily accessible for people who are 5'2" - 6'4".

After completing the design, I measured the size of the panels and started building.

The Adobe Illustrator file (.AI) is attached to this page.







File Downloads

Camera.ai (1 MB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Camera.ai']

Step 4: Cut The Panels

The panels were cut out of 1/2" Plywood. They are thin enough to be light weight, but thick enough to screw / nail into without worrying about additional reinforcements inside the case.

Image Notes
1. Each panel was cut out and labeled so I didn't lose it or accidentally chop it up for another project.

Image Notes
1. The holes to be cut in the back panel. Their size and placement were checked against the illustrator blueprint.
2. Originally the button was placed to the side of the screen. Unfortunately this placed an odd amount of torque on the mount when it was pressed, causing the camera to spin. It was later moved to the center, just under the screen.

Step 5: Bottom Panel - Tripod Mount

The camera body was designed to mount onto a heavyweight tripod. Inorder to make it stable, I had to provide an additional support in which the tripod tube could slide into and lock.

I borrowed this tripod from a friend. it has a 1 1/4" Outer Diameter tube which i used to mount onto. Home Depot sells galvanized pipe with an Inner Diameter that is slightly larger than 1 1/4" in addition to the necessary surface mounting hardware.

The galvanized pipe has a hole drilled into it, allowing me to insert a pin (which intersects with holes in the tripod tube), creating a stable mount that can only be released from inside the camera box.

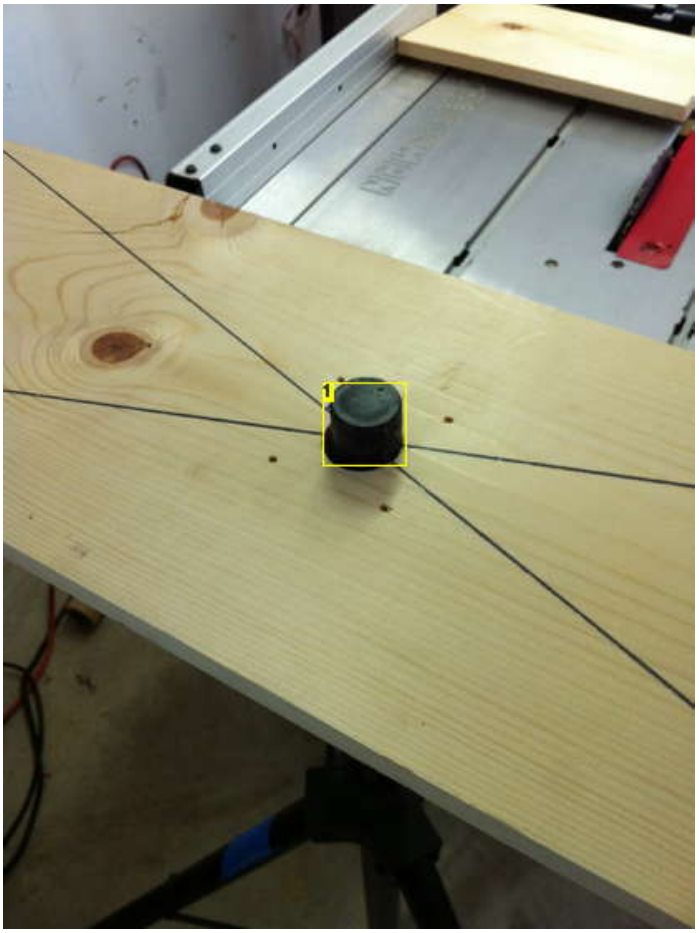


Image Notes

1. A 1 1/2" hole is drilled in the center of the bottom panel

Image Notes
1. Surface mounting hardware is added so that the galvanized pipe has something to secure to

Image Notes
1. A hole is drilled into the galvanized pipe so that a pin can be inserted into it

Image Notes
1. The pin is inserted into the pipe, and through the tripod tube. The entire assemble is now safe and secure

Step 6: Box Construction

The camera box was constructed from 1/2" plywood, with a 3/4" Pine base (for added strength).

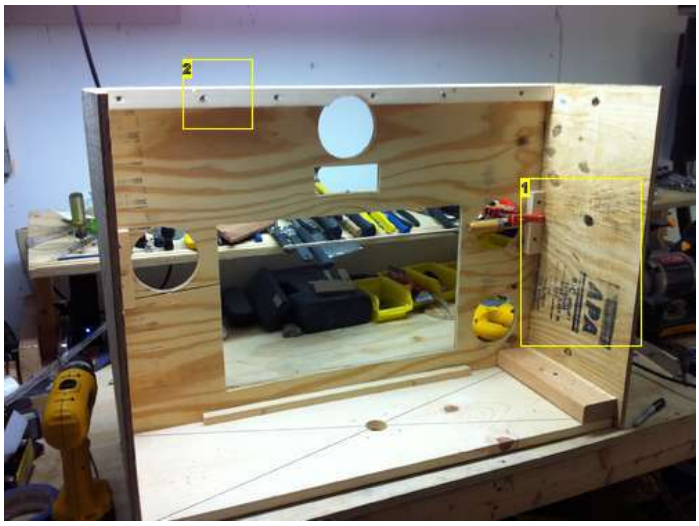
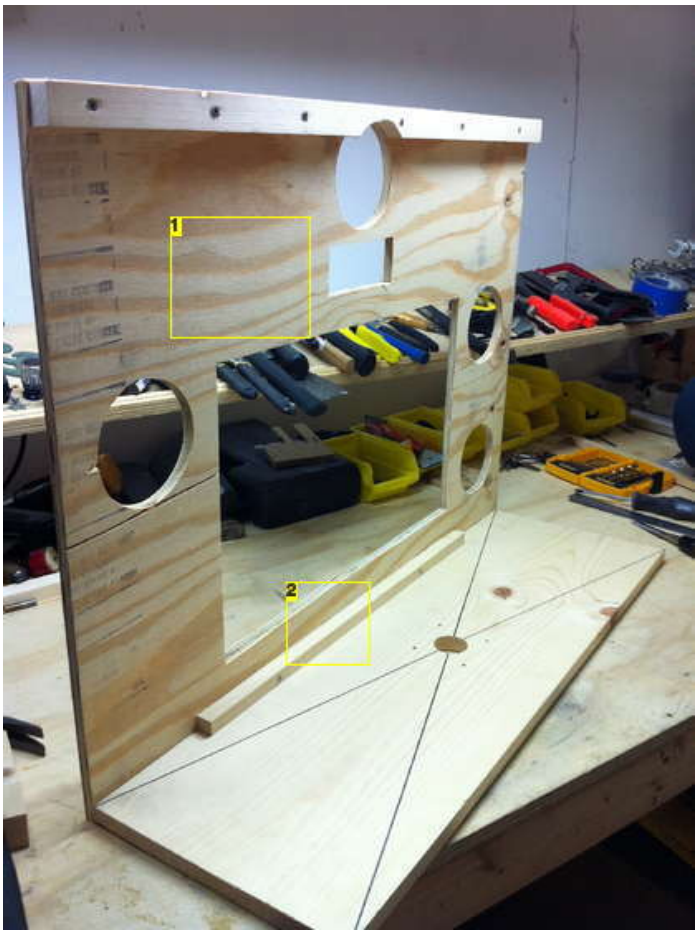


Image Notes
1. The side panels are added
2. This beam is used to give the top panel additional points to attach to



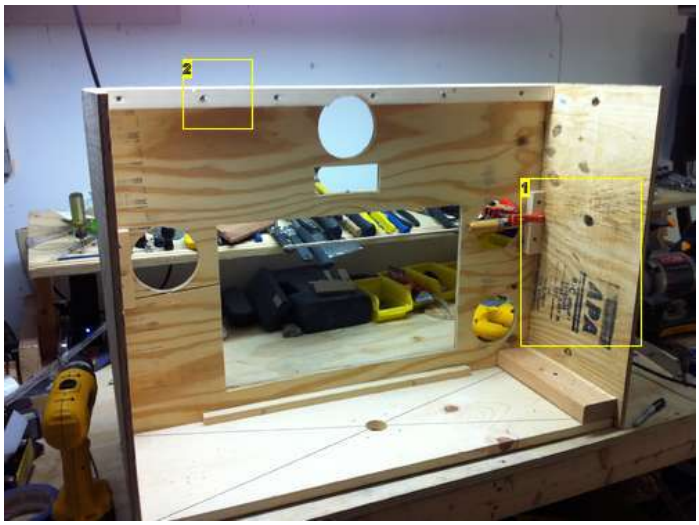


Image Notes
1. The Back of the camera, attached to the base
2. This support raises the LCD monitor to the right height off of the base, keeping it centered in the window

Image Notes
1. This is the door for the front of the box. I added a beam to the top of this panel so that I can secure it to the camera box with a piano hinge.

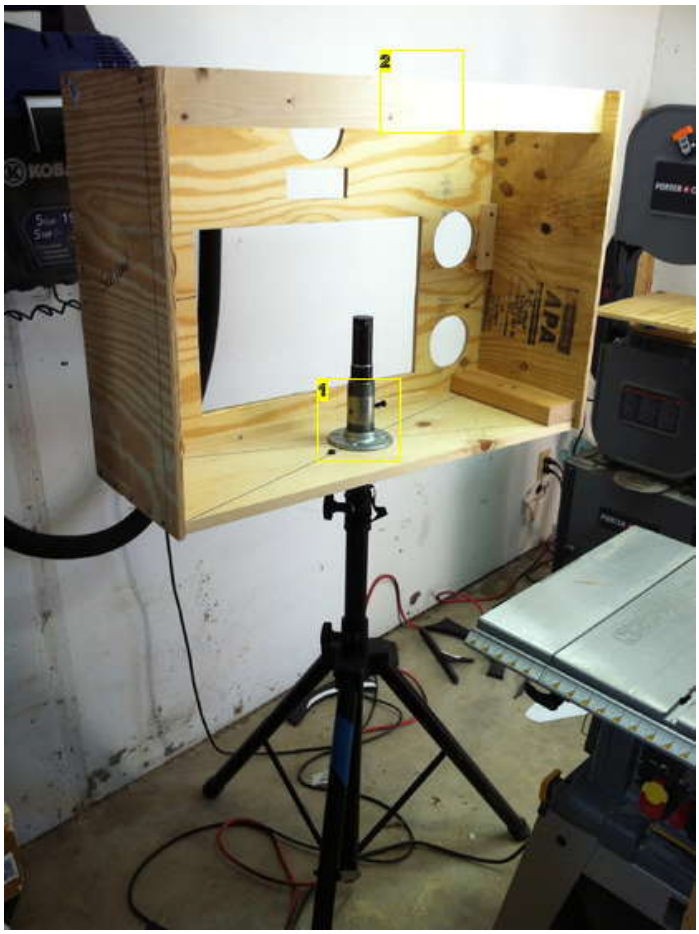


Image Notes
1. The camera box is secured to the tripod
2. This beam provides support between the two side panels, and will also be used to secure the front panel onto via a piano hinge

Step 7: Adding Components

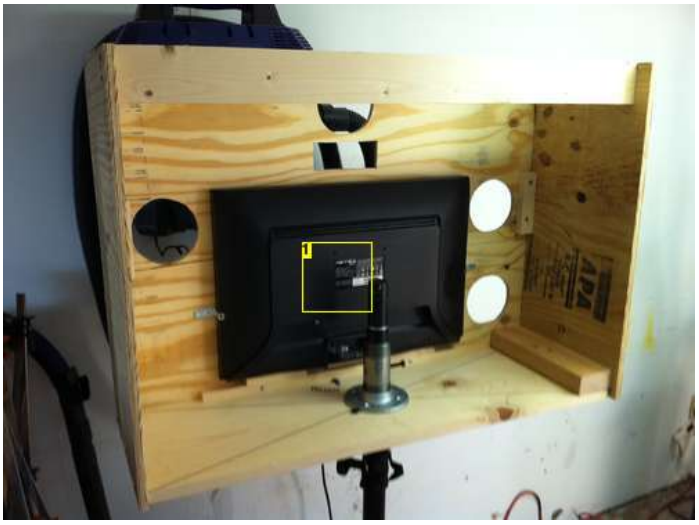


Image Notes
1. The LCD is added to the Camera Box. It is secured by a few L Brackets

Image Notes
1. The L Bracket is bent at the tip to hold the LCD monitor still.

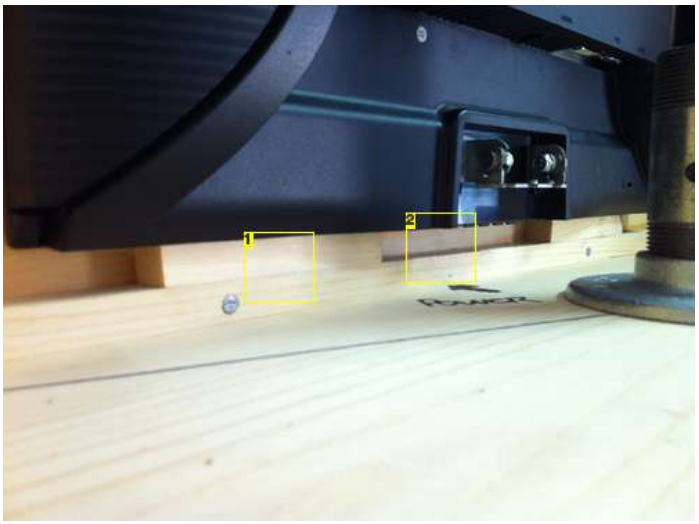


Image Notes
1. Additional shims were needed to center the screen in the cut out window.
2. Be mindeful of where the buttons on your monitor are. you will still need to be able to fit your fingers in here to turn on the screen when its ready to be used

Image Notes
1. A little note so i knew which button to press once the camera was all put together and the LCD monitor is locked in place.

Image Notes
1. Halogen lights from IKEA were modified to attach to the camera box.
2. A bent L bracket is used to secure the light to the box

Image Notes
1. The light control box is placed out of the way, and the extra cable is wound up.
2. All of the lights and the LCD monitor are mounted and working.

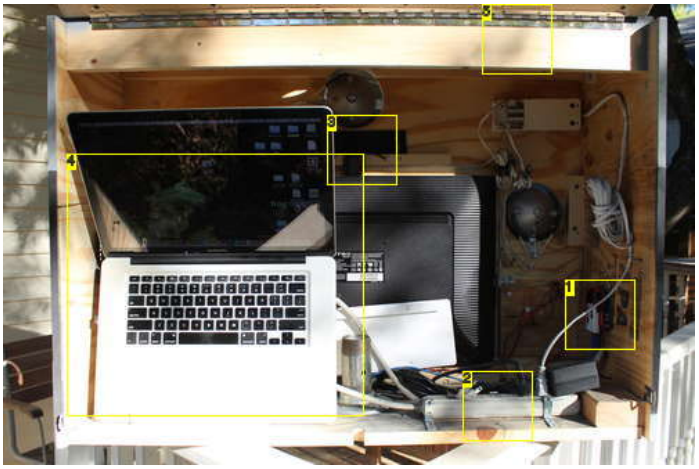


Image Notes
1. The arduino Micro Controller is mounted to the side panel here
2. I needed to add a power strip
3. The Logitech Camera is mounted just above the LCD monitor
4. The MacBook Pro is positioned here. After some initial Testing, i found that it overheats when it was mounted with the lid closed. Keeping the lid open prevented the computer from overheating and shutting down.
5. In this picture, you can also see the piano hinge which secures the door to the camera box.

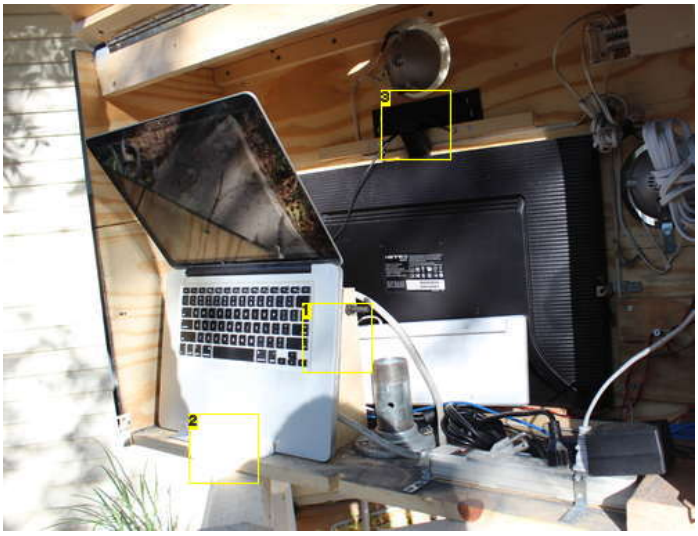


Image Notes
1. Here you can see the supports that hold the computer. A simple bungie cord is used to secure the computer to the supports
2. A piece of angle iron stops the computer from slipping forward
3. The webcam

Step 8: Testing

Once all of the components were added, I decided to stress test the photo booth. I wanted to see how long the device would continually run without any intervention, in conditions that it would likely encounter (sitting outside in the sun with +90 degree temperatures).

Note to you - TEST OFTEN AND EARLY!!

Here is a video of my early test - this is what i learned

http://anotherfrog.tumblr.com/post/12445371555/photobooth

Heat is an issue - After watching the computer overheat with about 15 minutes of use, I realized that heat was a big issue. I installed some old PC fans to help get rid of the heat.

Mac computers heat up with the Lid closed - Even with installing the fans, the compute was getting too hot. My original mouting system secured the computer in a vibration proof rig - with the lid closed. It was only after opening the computer so that the processor fans could run unobstructed that the device no longer overheated. This discovery led to 1 more fan (in the base of the camera box) and a mount which held the computer open.

The button was in the wrong place - with the button on the edge of the camera box, pressing it too hard caused the box to rock back and forth. Potentially an issue with inebriated guests, I moved the button to the center of the camera box, so that the force is directly perpendicular to the tripod.

Image Notes
1. With the button mounted on the side of the box, pressing too hard would cause the camera box to rotate back and forth

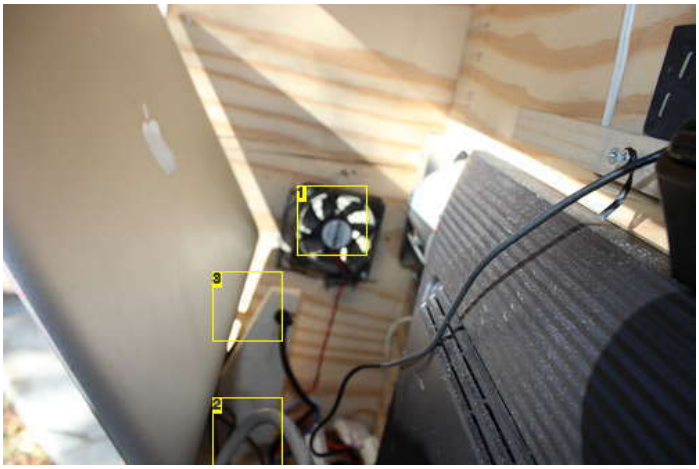


Image Notes
1. The first PC fan directs air directly across the back of the computer
2. The second PC fan directs air up from the bottom of the computer
3. This mount holds the computer open during operation - allowing the internal cooling fan to operate without obstruction

Step 9: Details and Finishing - Part 1

In this step, the components are removed, the box is primed and sanded. A coat of spray adhesive is applied to the surface to give the camera box a rough texture. The box is then primed and painted with Duplicolor paints - available at most auto supply stores.

Image Notes
1. A little bondo and a lot of sanding to smooth out all the imperfections of plywood
2. The old button hole has been refilled and sanded smooth

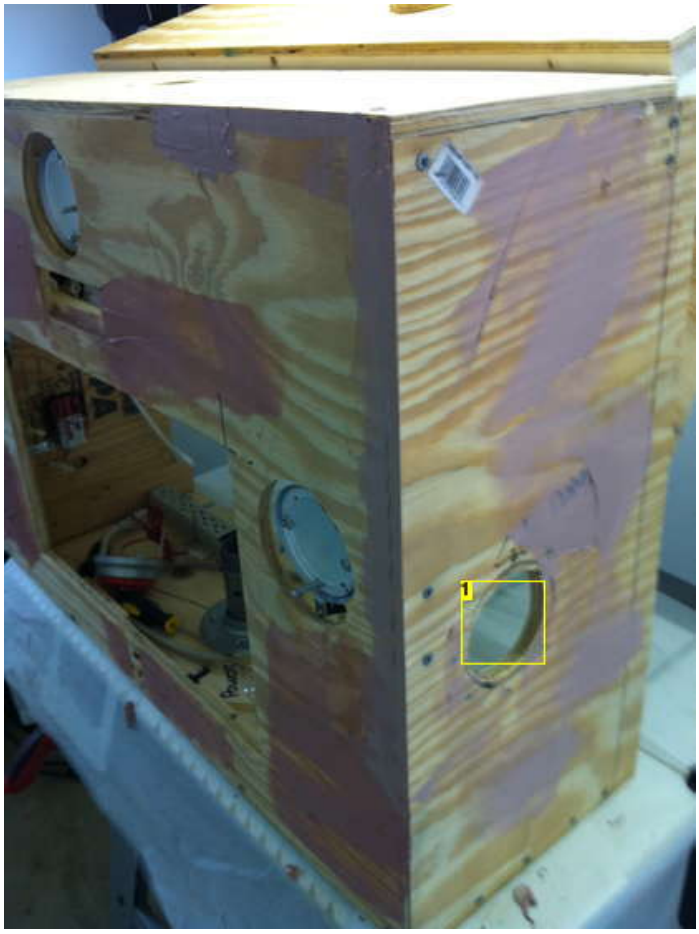


Image Notes
1. This is the hole for the side mounted PC fan. It will need some type of cover. I found one at Frys Electronics for $2.

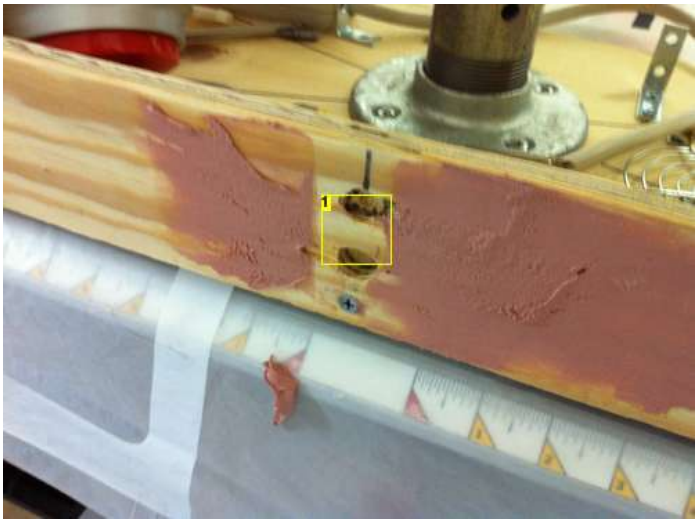
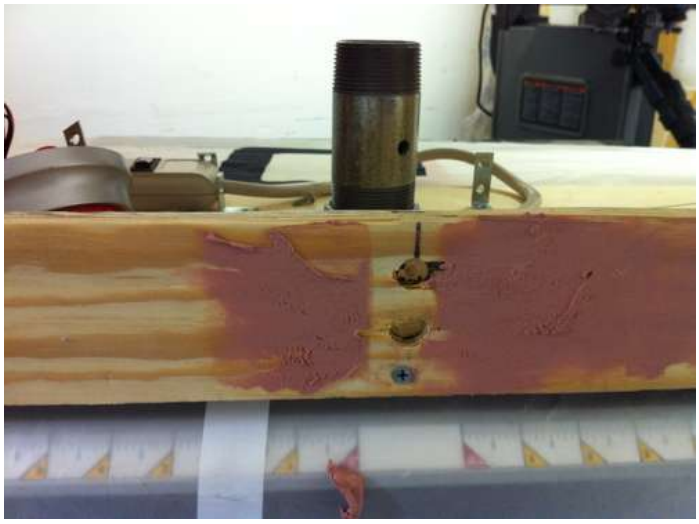


Image Notes
1. New attachment points for the button



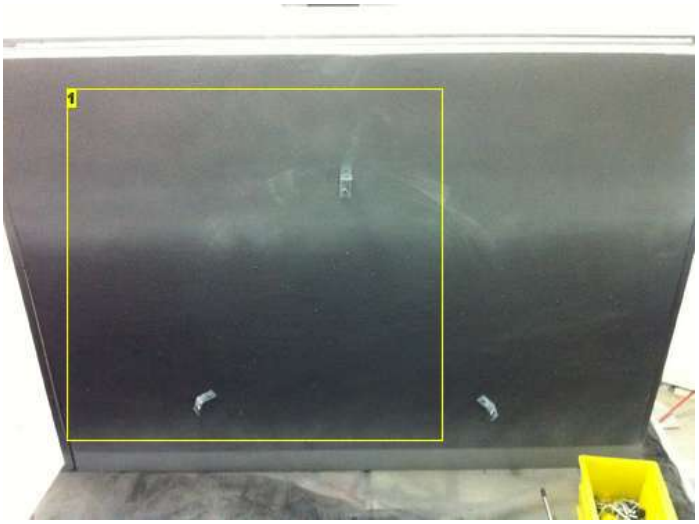


Image Notes
1. I used a trick to texturize the paint - and make it look like plastic. If you put down a layer of spray glue, holding the can far away from the surface, the glue will dry before it hits the panel - allowing for a nice texture to paint over, further obscuring any imperfections in the wood

Image Notes
1. The final paint job with the components re-installed

Step 10: Details and Finishing - Part 2

In this step, additional embellishments are added to make the camera box appear more like a camera.

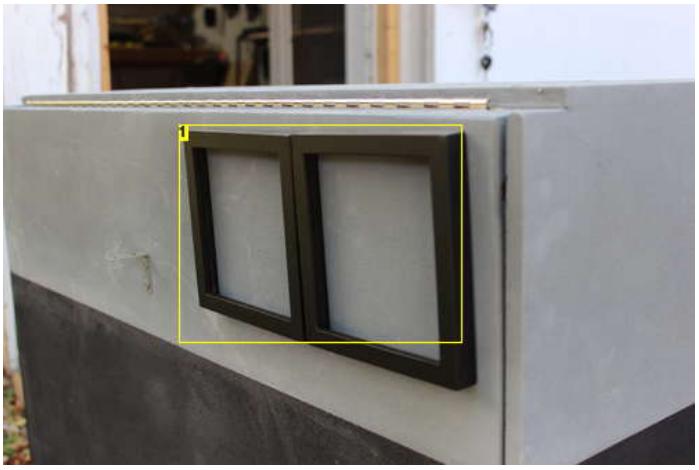


Image Notes
1. Some old photo frames were used to give the appearance of a view finder

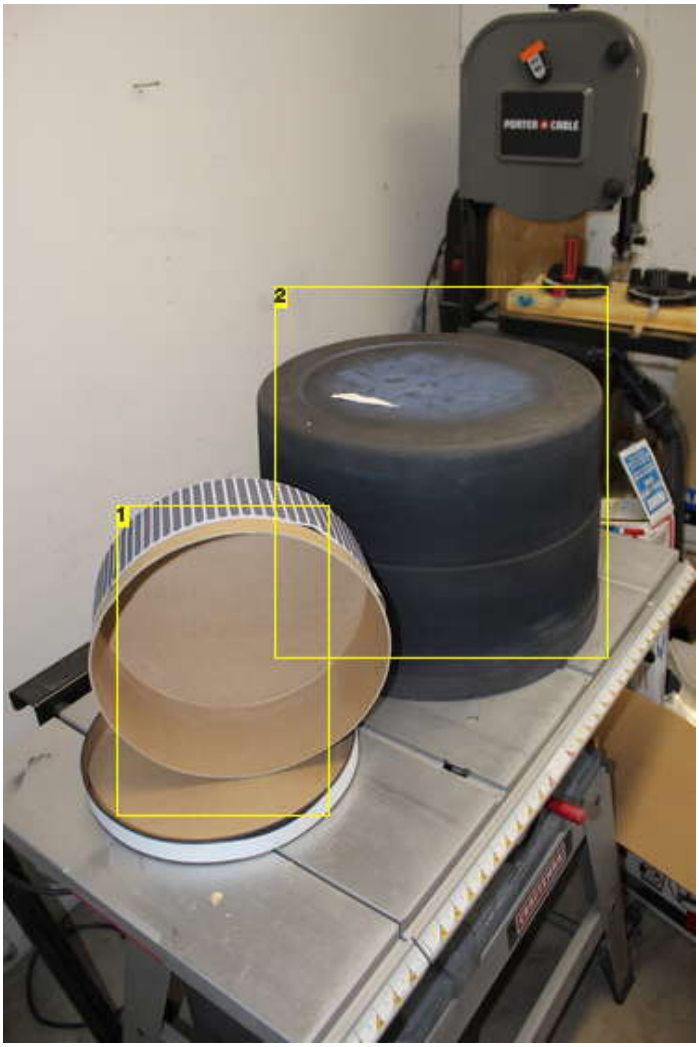


Image Notes
1. A hat box from the craft store is used to make the settings dial
2. This plastic drum from big lots is used to make the lense - a $5 part

Image Notes
1. Lens with decal applied
2. Settings dial with decals applied

Image Notes
1. The drum is mounted onto the front of the camera with l brackets and small screws. This method allows the lens to be removed for travel

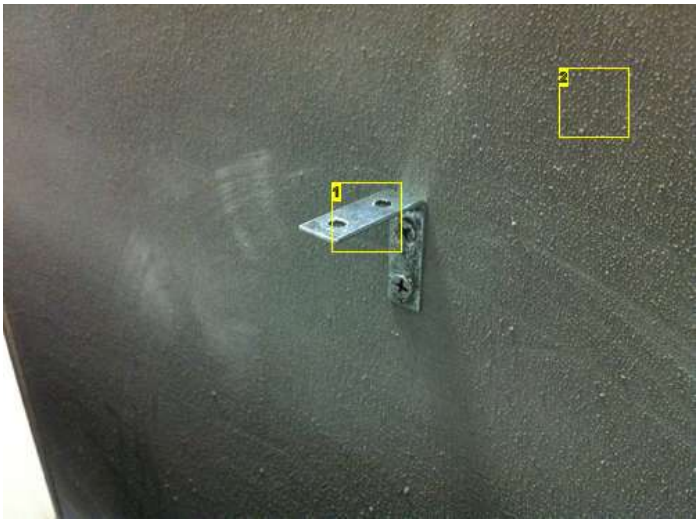


Image Notes
1. Close up of the l bracket
2. Here you can see the texture of the paint - achieved with a light dusting of spray adhesive before the paint and primer are applied

Image Notes
1. A small screw in the bucket fits perfectly into the hole in the L bracket

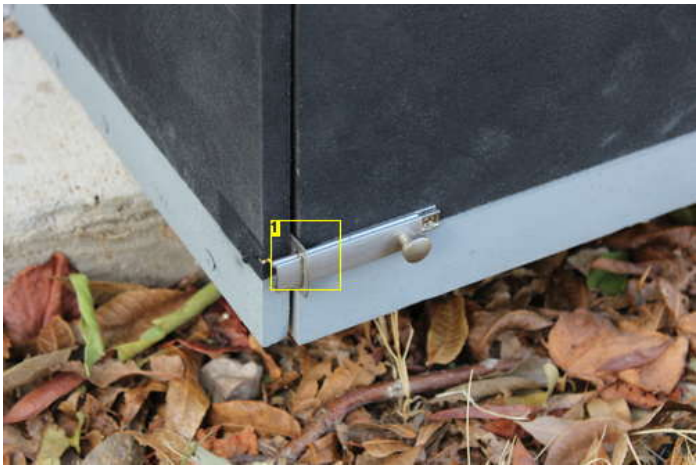


Image Notes
1. Locks are added to the camera door

Image Notes
1. a cover is added to the PC fan hole

Image Notes
1. A few more coats of black primer, and the lens will be finished

Image Notes
1. Everything is finished!

File Downloads

lens vector2.ai (915 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'lens vector2.ai']

Step 11: Usage

Here are some of my favorite photos from the wedding (The guests took over 800 photos in a 4 hour period). My wife and I also rented a bunch of props from a local theater company, in addition to making a few of our own.

I hope you enjoyied this post. Please let me know if you make this yourself and especially if you make any big changes. I'm always excited to see how people make things their own.

Matt Franks

Mfranks at famunited dot com

Image Notes
1. Me and the little lady!

Related Instructables

DIY Portable Wedding Photo Booth by dohjoe

Automatic Photo Booth Using Arduino Board (Photos) by jordanpacker

How to build your own Photo Booth by MoonRaker

DIY Wedding Photobooth with Easy Disassembly (pics only)

Photo Booth with a Live Slideshow by lightingCat

Make Your Own Tabletop Photobooth by jumpfroggy

Photo Booth by leuff

DIY Photobooth by jchorng

(Photos) by nonoodlez