**IEEE@IIT Quadrocopter**

**Project 3**

**“The Drones”**

# Project sub-teams:

* **Structural** — This team deals with assembling the quadcopter frame and mounting components on top of it.
  + Members: *Jackson Gu, Guhao Fan, Xing Zhong, Khor Hui Mink, Haimin Lin*
* **Flight** — This team works with the flight control system, ESC's, motors, and other related components to make sure everything is wired up properly to be flyable.
  + Members: *Peyton Zeller-Av, Jackson Gu, Karol Gluc, Eric Tendian, Manoj Varma Penmathsa, Boyang Dong*
* **Communications** — This team connects the flight control system to the receiver and ensures that the transmitter and receiver are talking to each other. This team prevents the quadcopter from going out of range, as well as ensuring that other teams' wiring does not interfere with any commands being sent.
  + Members: *Bobby Unverzagt, Yevgeniy Rezhichenkv, Dhiraj Zalse, Manoj Varma Penmathsa, Raed Tawil*
* **Programming** — This team programs the autopilot system as well as calibrates the controls. They deal with all aspects of software, including making sure the flight control system is calculating the correct power values. They also ensure that the software is compatible with the other hardware.
  + Members: *Peyton Zeller-Av, Guhao Fan, Yevgeniy Rezhichenkv, Eric Tendian, Boyang Dong, Odikekachukwu Eze-Echesi, Sijia Wu, Shoaib Khan, Raed Tawil*

**Resources:**

* <http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/>
* [**http://blog.oscarliang.net/build-a-quadcopter-beginners-tutorial-1/**](http://blog.oscarliang.net/build-a-quadcopter-beginners-tutorial-1/)
* <http://diydrones.com/>
* <http://copter.ardupilot.com/>
* <http://www.diydrones.com/notes/ArduPilot>
* <http://www.goodluckbuy.com/free-flight-ff-auto-balance-controller-with-3d-acceleration-sensor.html>
* <http://oddcopter.com/flight-controllers/>
* [**http://www.scoutuav.com/build-scout/parts/**](http://www.scoutuav.com/build-scout/parts/)
* <http://www.ecalc.ch/xcoptercalc.htm?ecalc&lang=en>
* <http://www.instructables.com/id/Cheap-Sturdy-1-Hour-Quadcopter/>
* <http://www.ebay.com/itm/5-pairs-315Mhz-RF-transmitter-and-receiver-link-kit-/400498100396?pt=LH_DefaultDomain_0&hash=item5d3f8c08ac> - transmitter/receiver. Can make an array for extra accuracy? - Jeff R
* <https://collaboratory.airforce.com/quadrotor/>
* <http://copter.ardupilot.com/wiki/common-six-modes/>

***Journal entry format:***

**MM/DD/YYYY**

***Attendance:* names here**

**What we did today:**

blah

**Things to do the next meeting:**

* blah

# **Week 1**

**12/09/2013**

***Attendance: Eric Tendian, Boyang Dong, Guhao Fan, Sijia Wu, Manoj Varma Penmathsa***

**What we did today:**

First meeting of winter break! We introduced ourselves, then looked at the different components of a quadcopter. Eric explained the purpose of each component we ordered, and identified some issues with the current set of parts. We then worked on cleaning up the room and unpacking any orders. The projector works!

Some sample code we can modify for our own quadcopter: [**https://code.google.com/p/quaduino-ng/**](https://code.google.com/p/quaduino-ng/)

It is Arduino-based so it should be compatible.

**Things to do the next meeting:**

* Decide what parts we are missing that need to be ordered
* Put together a single motor/propeller and make it spin

**12/10/2013**

***Attendance: Eric Tendian, Boyang Dong, Guhao Fan, Manoj Varma Penmathsa,***

**What we did today:**

We started out by watching a video called [“Dance of the Flying Machines”](http://youtu.be/Glvla0nFWHo) and then connecting that to the purpose of our project. After we did intros again (since we had a new member) we talked about various “landing gear” designs so that the impact of landing would be absorbed without going directly into the quadcopter.

Some ideas we came up with:

* memory foam
* springs
* springs in X-shape w/ supporting beams
* sponge
* [foam] pool noodle
* steel straps
* cage
* tennis balls
* plastic straps
* <http://www.pyramidmodels.com/shop/shopimages/AC285-c.jpg>
* <http://www.gregcovey.com/RCU5/March9a.jpg>

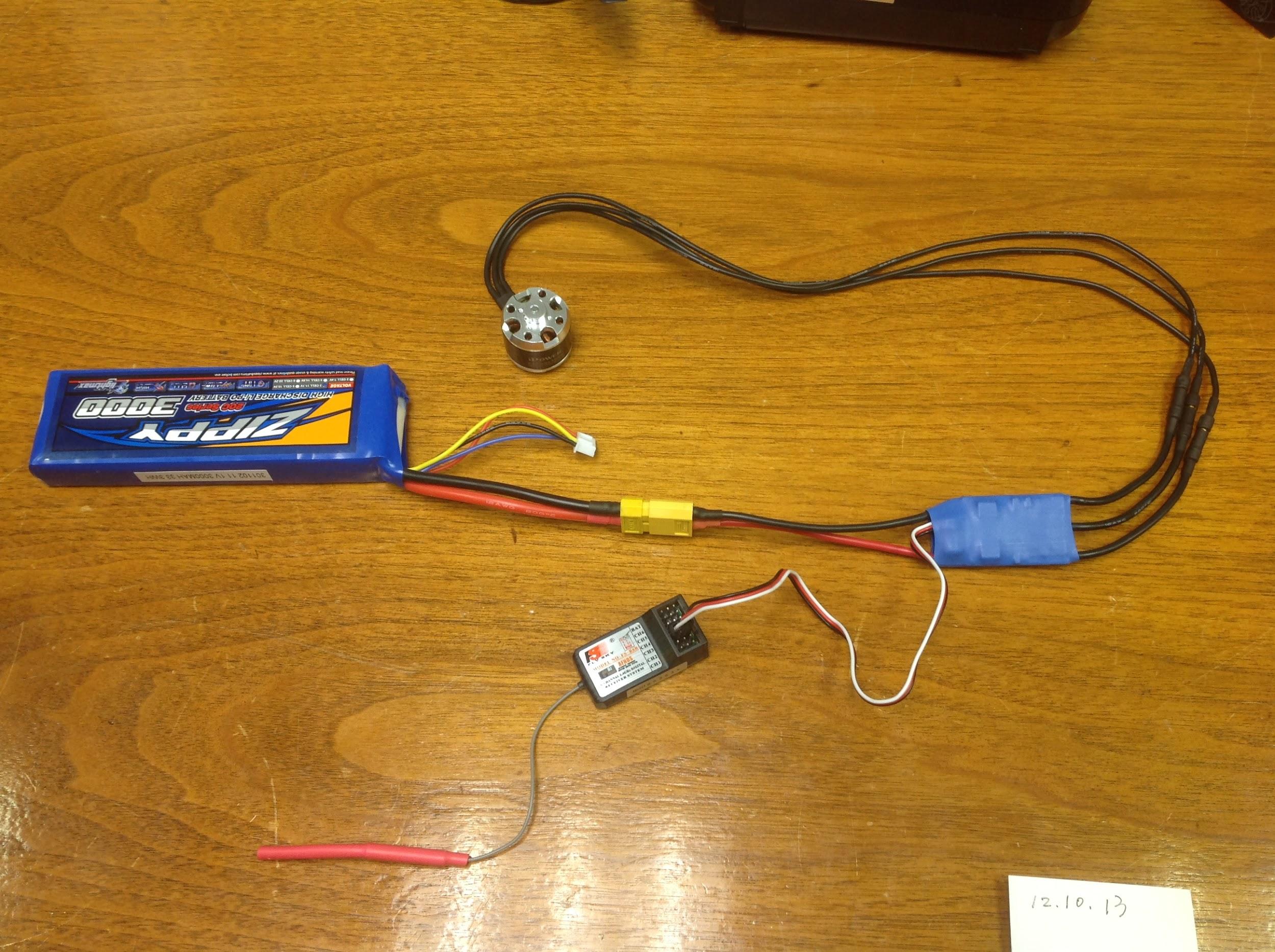
We need to find the most cost-efficient solution.

We also made a list of some parts we need to get:

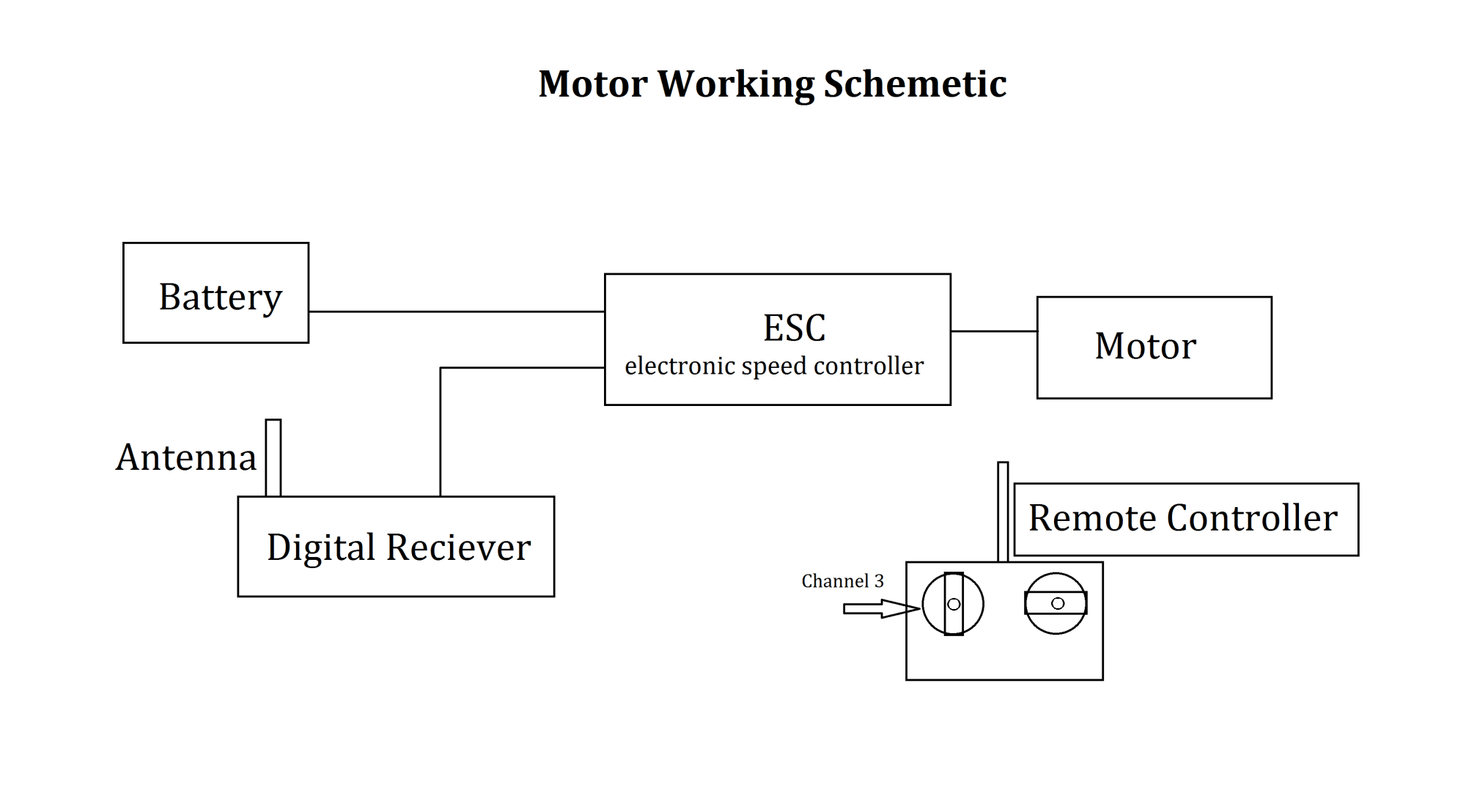
|  |  |  |  |
| --- | --- | --- | --- |
| **Quantity** | **Part** | **Why** | **Link** |
| 1 | Telemetry Kit | Sensor data to calculate position/rotation | <http://store.scoutuav.com/product/add-ons/3dr-radio-telemetry-kit-915-mhz/> |
| 1 | Carbon fiber tubes | For our frame! | No specific product found yet |
| 1 | Power Distro board | To split power among motors | <http://www.hobbyking.com/hobbyking/store/__23140__hobby_king_quadcopter_power_distribution_board.html> |
| 1 | USB A to Mini-B cable | For connecting to Uno32 | <http://www.amazon.com/C2G-Cables-27005-Mini-B-Cable/dp/B000067RVL> |
| 1 | Fasteners (nuts, bolts, screws, etc) | To mount motors, other components | No specific product found yet |

Finally, we worked on assembling a propeller/motor combo. We attached a propeller to a motor successfully, and then tried to power the motor. We had some issues powering the motor because there were three black wires, and we weren’t sure which ones go where. Powering the motor with the power supply didn’t work, after the multiple combinations we tried. Perhaps it wasn’t getting enough current? We then decided to connect the motor to the ESC, and the ESC to the battery with the power connector, and the ESC’s ribbon cable to the receiver. Then we added batteries to the transmitter (RC remote) and powered it up. The ESC beeped, but nothing moved. After some more troubleshooting, Manoj moved the trim setting for the joystick to a different position, and tried the joystick again. This time, it worked! We still aren’t sure if that was the issue or not, but now we have one motor spinning. More work tomorrow.

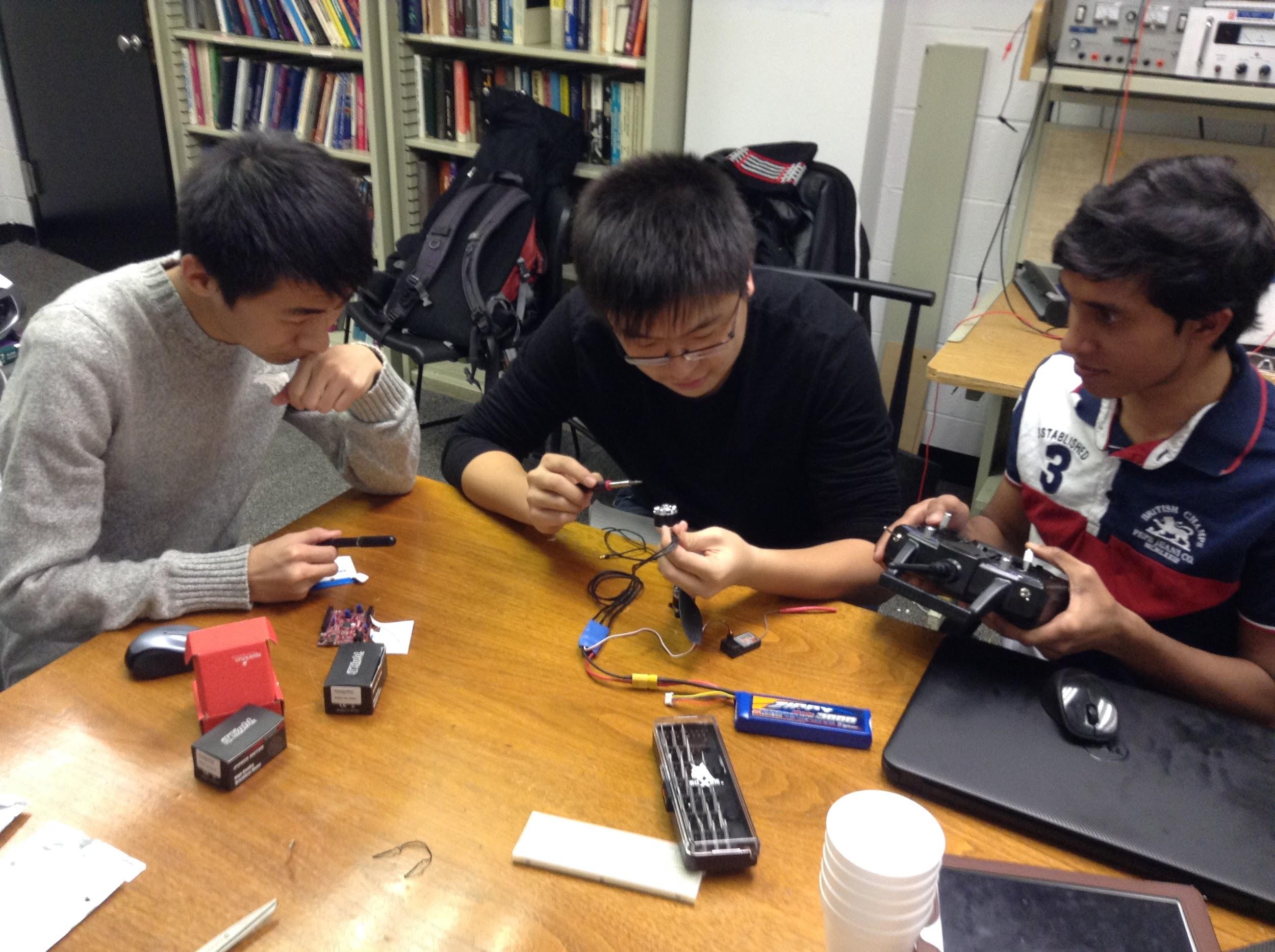
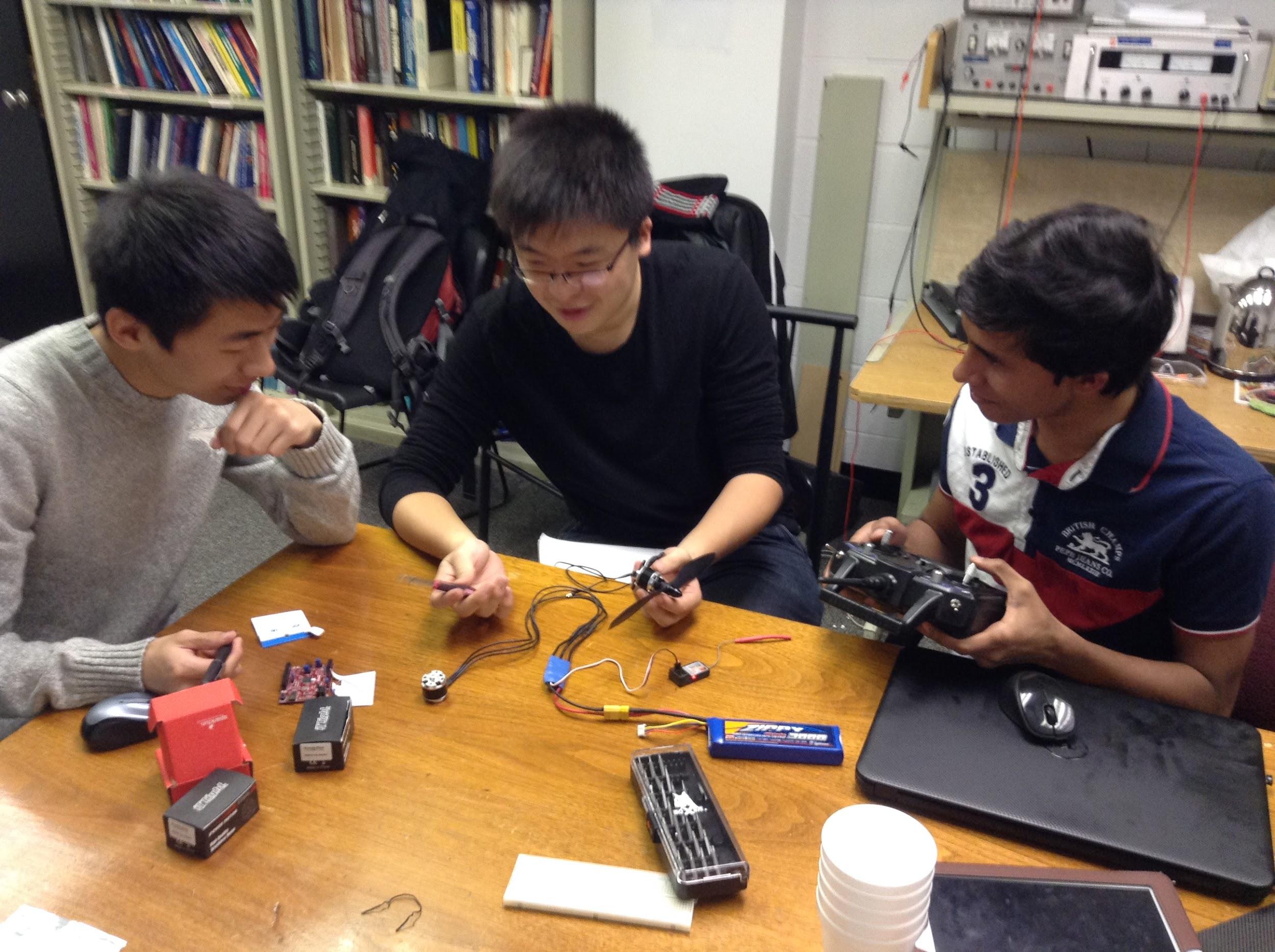
**Pictures:**



The completed setup.



Doing work (photo taken by Eric):



**Things to do the next meeting:**

* Put together and test the rest of the motor/propellers
* Plan out what to do with carbon fiber

**12/11/2013**

***Attendance:* Eric Tendian, Boyang Dong, Manoj Varma Penmathsa**

**What we did today:**

For today’s meeting, we started out by attaching the propellers to the rest of the motors. Then, we thought about testing the rest of the ESC’s and motors. However, we wanted to mount the motor onto some wood before we tested it, so that nothing would fly around and hurt the wires or us. After searching unsuccessfully for screws and other tools to mount the motor, we headed over to Alumni Memorial Hall and asked Stan Johnson (the shop manager) to help us out. He helped us find screws and nuts which would work, give us a tutorial on how to use the drill press, and allowed us to drill holes in our compressed wood board to mount the motor. After we had drilled the holes and mounted the motor w/ a propeller, we tested out each of the four ESC’s to make sure they were working properly.

For an ESC to be connected properly, the following needs to occur:

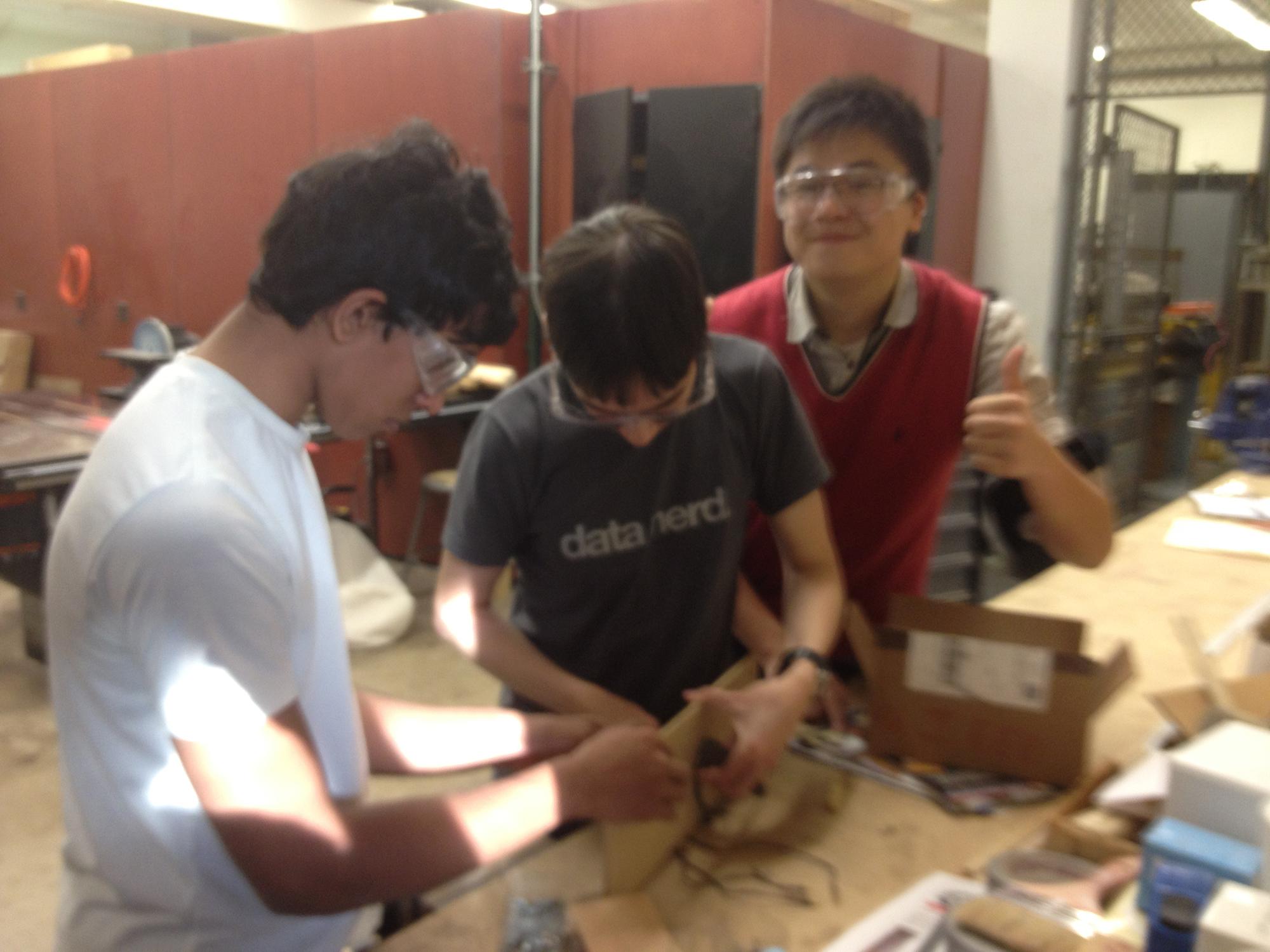
1. Connect the 3 wires from the motor to the ESC - order doesn’t matter
2. Connect the ESC ribbon cable to the receiver, with the black wire pointing to the right (see illustration on the receiver).
3. Connect the power connector to the battery (or power distro board).
4. You should hear a series of 3 beeps. If you do not hear that, the battery is not connected properly.
5. When you turn on the transmitter (the remote) you should hear another 2 beeps (or some other number). This means the connection is good.
6. Test the motor by moving the joystick. There is a switch on the transmitter at the top left (SW. B) which turns the left joystick up/down motion on and off. The top right switch should control the right joystick.

After some troubleshooting, we got all of the ESC’s to work successfully. Then we disassembled our testing station and went back to the lab. *Now all our electronics are working properly!*

*Here’s some pictures of us in the shop:*



Using the drill press to make holes in the wood panel.



Mounting the motor to the wood panel.

After coming back to the lab, we did some research on battery charging and found two ways to charge it. We looked at [this video](http://www.youtube.com/watch?v=C9KZkvW9jyw) for inspiration. There is a [single-purpose charger](http://www.hobbyking.com/hobbyking/store/__41635__iMAX_B6_Compact_50W_5A_Automatic_Balance_Charger_2_6S_Lipoly.html) which only works for our LiPo battery, but costs less. There is also a [universal charger](http://www.amazon.com/Tenergy-TB6-B-Balancing-Charger-Battery/dp/B00466L0BW/), which may be harder to configure but will be more cost-efficient in the future if we need to buy additional types of batteries.

**Things to do the next meeting:**

* Go over programming plan (do we need to buy an ArduPilot)
* Figure out what size screws we need, add to order list
* Figure out where our carbon fiber is (hasn’t arrived yet?)
* Plan out what to do with carbon fiber

# **Week 2**

**12/16/2013**

***Attendance:* Eric Tendian, Boyang Dong, Manoj Varma Penmathsa**

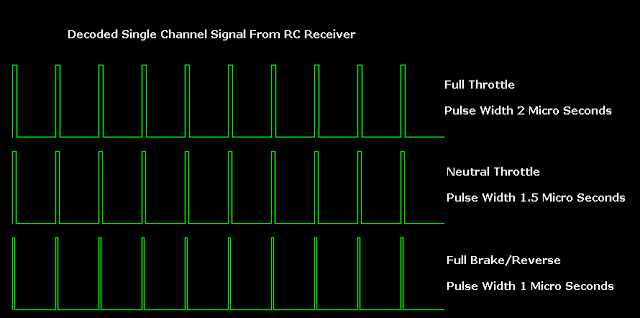
**What we did today:**

We did some research on “[the difference between analog and digital signals](http://www.diffen.com/difference/Analog_vs_Digital)”. Eric briefly introduced the process on how the motors rotate controlling by the signal transmitted from the remote. He also explained what a PID controller is, and how it integrates with flying the quadcopter.

More info on PID controllers: <http://blog.oscarliang.net/quadcopter-pid-explained-tuning/>

Analog signals are electric pulses of varying amplitudes, while digital signals have a single amplitude. In analog signals, the amplitude is used to measure the value, while in digital signals the duration of the pulse is used to measure the value.

For example:



In the quadcopter receiver, there are six of these channels, each pulsing with activity. Usually, the pulses are of a very short timespan (e.g. 1ms to represent 0) but as the values increase the width (duration) of the pulse can increase. The receiver's job is to convert the radio waves back into digital signals, and transmit those signals through the pins on the receiver board. We use a jumper cable to pick up that signal, then send it over to the Arduino. The Arduino program processes that signal value, manipulating it if necessary, and then sends out the processed digital signal on another pin, which is connected to the ESC’s cable via a jumper cable or breakaway header. The ESC takes that digital signal, reads it, and then adjusts the amount of power given to the motor.

Relevant links:

[rcarduino.blogspot.co.uk/2012/04/how-to-read-multiple-rc-channels-draft.html](http://rcarduino.blogspot.co.uk/2012/04/how-to-read-multiple-rc-channels-draft.html)

[rcarduino.blogspot.com/2012/01/how-to-read-rc-receiver-with.html](http://rcarduino.blogspot.com/2012/01/how-to-read-rc-receiver-with.html)

We decided that for our first edition of the quadcopter will make use of existing code, probably from the [quadrino-ng](https://code.google.com/p/quaduino-ng/) codebase. We can adapt the code for our own use, and learn how it works. Eventually we want to write our own quadcopter control program. The quadrino-ng project page has a list of parts and wiring diagrams, which will be very useful as we start assembling our own quadcopter.

As for the connection between ESCs and the Arduino, [here](http://dronesandrovs.wordpress.com/2012/11/24/how-to-control-a-brushless-motor-esc-with-arduino/) we can get some inspiration.

Additionally, Eric sent an email to Peyton asking for info on the arrival date of the carbon fiber tubes, a key structural component of the quadcopter. We ordered these back in early November and they still have not arrived.

**Things to do the next meeting:**

* Hook up receiver channel to Uno32, and Uno32 to motor
* Make a sample program to process signal

**12/17/2013**

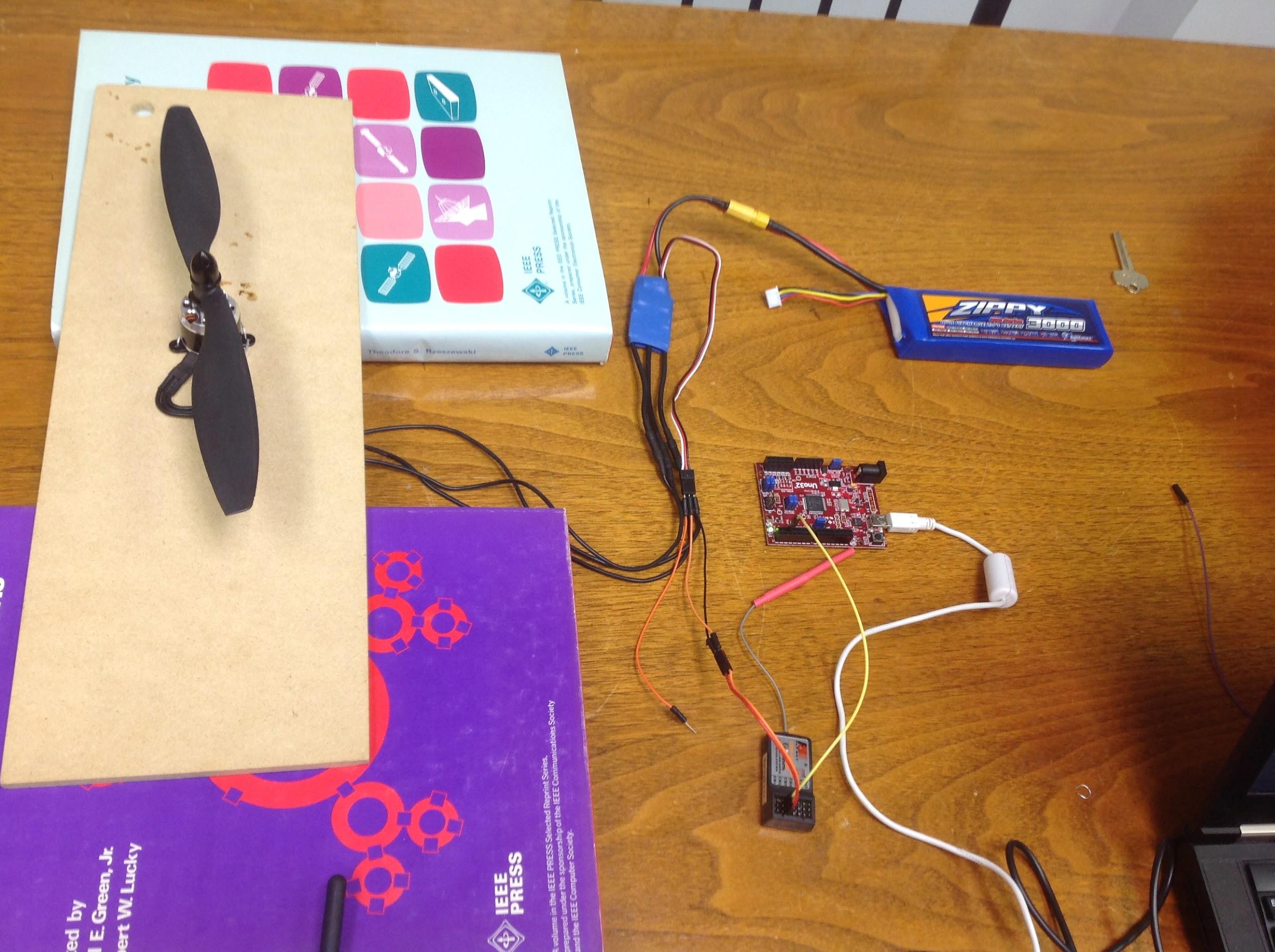
***Attendance:* Eric Tendian, Boyang Dong**

**What we did today:**

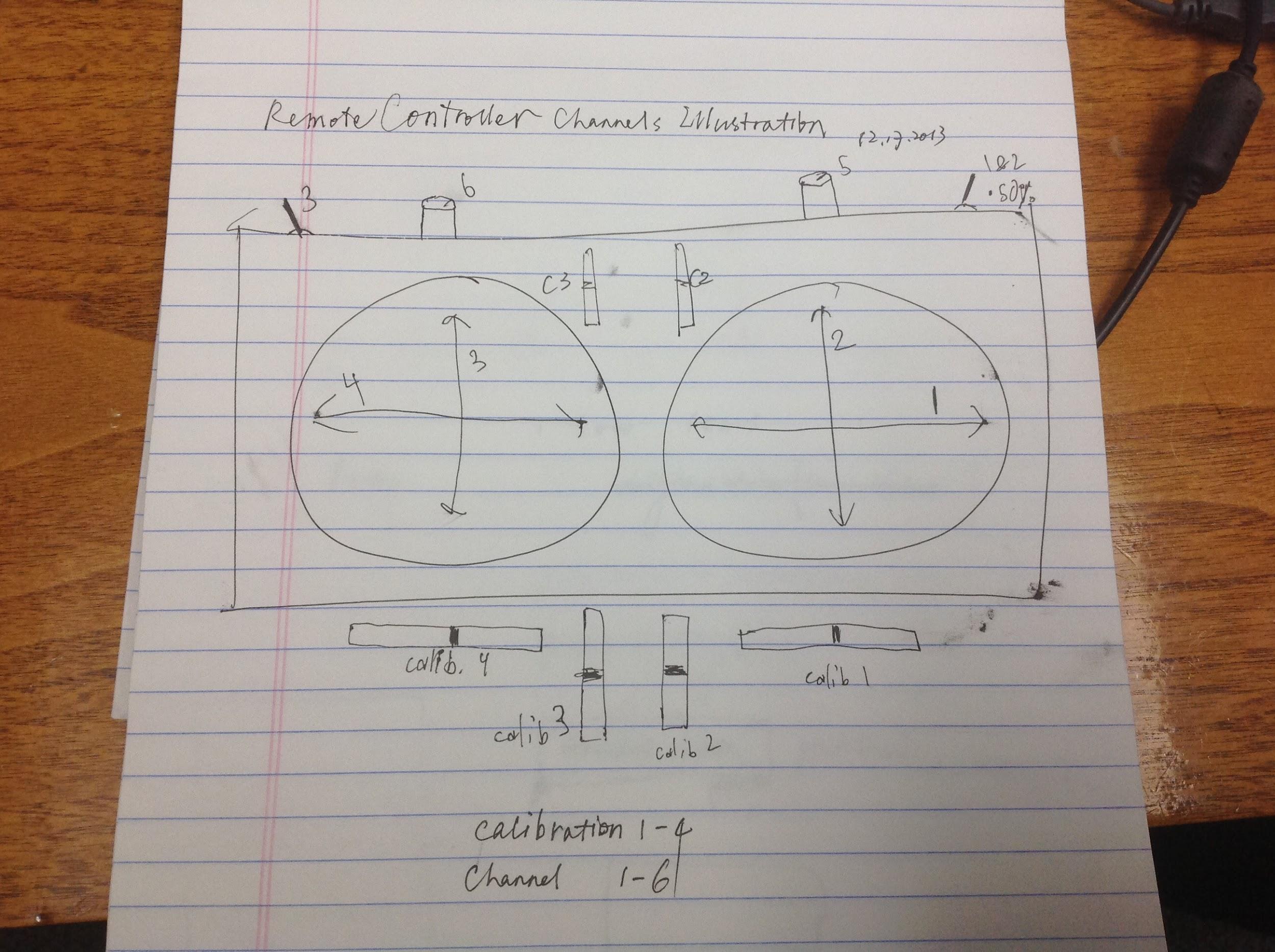
Today was an interesting and enlightening meeting. First, we mounted the motor onto the wood plank with paperclips (it is more sturdy than you think). Next, we connected the components in the following fashion:

* 3 motor wires to 3 ESC wires (all black)
* Black and red wires on the ESC jumper cable to the black and red (+ and -) pins on the receiver\*, by means of jumper wires; black-black and red-red obviously
* Signal pin on the receiver\* to pin 0 on the Uno32 (RX pin, digital)
* Signal pin on the ESC jumper cable to pin 1 on the Uno32 (TX pin, digital)
* Connected battery.
* \* - for the receiver, we used channel 3

It looks something like this:



After we connected these components, we plugged in the Uno32 to the computer and waited for the software to install. While we were waiting, we created a diagram of the different channels and their corresponding movements on the transmitter:



After the software install was complete, we came up with a program like this:

int receiverPin = 0;

int motorPin = 1;

int power = 0;

void setup()

{

Serial.begin(9600);

pinMode(receiverPin, INPUT);

pinMode(motorPin, OUTPUT);

}

void loop()

{

power = digitalRead(receiverPin);

Serial.println(power);

digitalWrite(motorPin, power);

}

With this program, the digital input is read, the power value (or so we thought was the power value) was printed, and the power value was written to the motor. It seems like this approach was working as we were able to adjust the power and the motor RPM would change accordingly. However, the problem was that when reading the power value, it did not read a specific integer, but rather displayed HIGH or LOW (which meant that it was not getting a correct power reading).

After some research, we discovered this, and Eric gave another tutorial on digital signals. We then started to doubt that the signal wire actually contained the signal, but after more research we realized we were in fact using the correct signal wire.

We found this very helpful in our research: <http://scprojects.wordpress.com/2013/10/13/hobby-radio-receiver-arduino-class-fs-r6b-fritzing-files/>

This site also links to a Github repository which contains a [receiver library](https://github.com/SCamarena/Arduino/blob/master/Libraries/HobbyRadioReceiver/HobbyRadioReceiver.h) that makes use of pulseIn and some other functions to change the signal into a value between -255 and 255. On line 171-172, there is the following code:

**return** constrain( map( checkRaw(ch), 1000, 2000, **-**255, 255 ), **-**255, 255 );

The function checkRaw(ch) is essentially the same as pulseIn(ch, HIGH). The map() function re-maps 1000 to a value of -255 and 2000 to a value of 255, and the values in-between get mapped to values in-between -255 and 255, making use of ratios. The constrain() function is sort of unnecessary, as it merely maps values <-255 to -255 and values >255 to 255, which should not happen if the map() function is working properly.

Using the attached schematic from the article, we realized that we should be reading the power value on an analog pin, and not as a digital pin. Our new code looked more like this (now the goal was to simply display the value):

int receiverPin = A0;

int power = 0;

void setup()

{

Serial.begin(9600);

}

void loop()

{

power = pulseIn(receiverPin, HIGH); //get the duration of the HIGH pulse

Serial.println(power); //print out the duration, which was in microseconds

delay(50); //allow the Arduino to reset

}

With this new code, we were getting values between 1000 and 2000, which made more sense because as we increased the power on the transmitter, the values went up accordingly in the serial window.

Now that we had the duration of the HIGH pulses and the corresponding power values, we wanted to replicate those pulses back out to the ESC/motor, so we could adjust the speed of the motor without making use of the transmitter. This is where things got tricky.

Because the ESC has a specific delay amount between pulses (incorrect, but based on current knowledge we assumed it to be correct; [see Eric’s note](#h.og0ff7mj420h)) we realized that we also needed to record the delay amount. We modified our code above to change HIGH to LOW in the pulseIn() function so it would measure the duration of the LOW pulses instead. When printing the values out, we got strange values such as 0, 14624, 17852, and other values in the 15000-18000 range (approximately). We tried to go off of one value, 17895, and use that as our delay for the LOW pulses. That meant each LOW pulse should be 17.895 milliseconds. For our sample program, we had a loop function similar to below:

void loop()

{

power = pulseIn(receiverPin, HIGH);

digitalWrite(motorPin, HIGH);

delayMicroseconds(power); //the duration of the pulse determines the power

digitalWrite(motorPin, LOW);

delay(17.895); //17895 microseconds, or 17.895 milliseconds

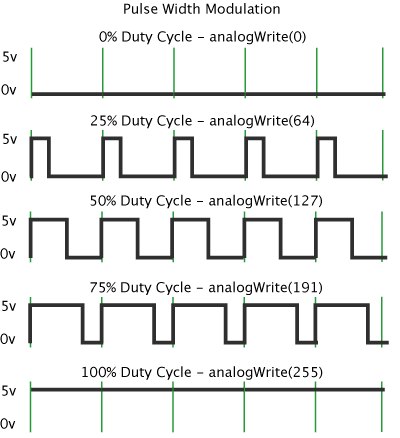
}

When this program runs, it should allow us to change the power from the transmitter, or via the code (through manipulating the power variable). However, upon running it, nothing happened. We then tried an arbitrary constant value for power, e.g. 1500, but still no success.

After trying out different unsuccessful programs, some of which used pulseIn(pin, LOW) rather than pulseIn(pin, HIGH) we decided to leave our current progress and continue tomorrow.

## Eric’s note:

I just read <http://arduino.cc/en/Tutorial/PWM> and looked at the different durations for different analogWrite values. See below:



Picture retrieved from <http://arduino.cc/en/Tutorial/PWM>

As you can see, the “duty cycle” remains the same each time (green dividing lines) while the duration of the HIGH pulse changes. In our previous understanding of digital signals, we had assumed that there was a constant LOW pulse duration between each HIGH pulse. E.g. there would be a HIGH pulse of 100ms (for 100% power) then a LOW pulse of 10ms, then a HIGH pulse of 50ms (for 50% power) and another LOW pulse of 10ms. While some digital systems have this behavior, the ESC is not one of them. It goes off of the PWM model, where each duty cycle (one loop) is of constant duration, and the sum of the duration of the HIGH pulse and the LOW pulse (think HIGH+LOW) equals one duty cycle. Therefore, when looking at a graph of the pulses, with a duty cycle of 200ms, and a max power value of 50% of the duty cycle, you would see: [50ms HIGH (50% power), then 150ms LOW], then [100ms HIGH (100% power), then 100ms LOW], then [10ms HIGH (for 10% power), then 190ms LOW] and so on. The two durations added together equal the duration of a single duty cycle.

In the context of our ESC, the duration of one duty cycle may be something like 20 milliseconds, going off of [the product page](http://store.scoutuav.com/product/parts/ipeaka-iq-22a-speed-controller/) where 50Hz = a period of 20ms (see input rate range). The upper end is 400Hz = a period of 2.5ms. Based on earlier data where we observed LOW durations around 17000, the hypothesis of a 20ms duty cycle period may be accurate.

We shall test this hypothesis tomorrow as we learn more about Pulse Width Modulation.

See this article for more info: <http://arduino.cc/en/Tutorial/SecretsOfArduinoPWM>

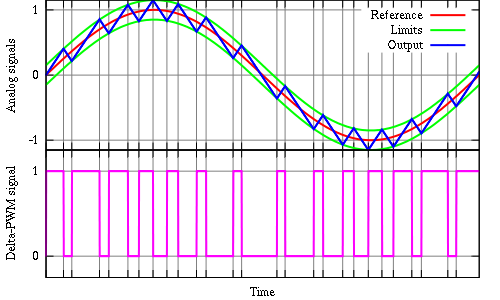
**Supplement from Boyang**

Here are some common Q&A:

* *What is PWM? What does it do?*

PMW means “Pulse-width Modulation”.

It is basically a technique that conforms the width of the pulse, formally transfer the analog signal to pulse duration.



Delta Pulse-Width Modulation

Picture retrieved from <http://commons.wikimedia.org/wiki/File:Delta_PWM.svg>

* *What are the characteristics of PWM?*

Pulse-width modulation usually uses rectangular pulse wave. The pulse width of wave is modulated resulting in a variation of the average value of the waveform.

Typically, the period of a modulation is a constant. From the picture in Eric’s notes, in each single cycle duty, the percentages of HIGH and LOW will be re-distributed with different power input, which we can control by remote controller.

* *What is the an analog circuit?*

As we know, in an analog circuit, the value of the analog signals can be given continuously. There is not any restrictions on the time or numerical range. It can be any real number. Input and output changes linearly, as a result, some household appliances, such as speakers, air conditioners can be easily remote controlled by the continuous current and voltage.

* *If the analog circuit has such a bunch of advantages, why we try to use PWM?*

Good question. There are numerous unavoidable issues of an analog circuit. For example, the control signal can be drifted easily by current and time, which is difficult to control. Besides that, an analog circuit is high power consuming, and also easily to be interfered with ambient conditions and noises.

* *Can a digital circuit solve all the problems listed above?*

Absolutely yes.

Quite different from analog circuits, the digital circuit is given values in a predetermined range. The output can only be two states, ON or OFF at any time. Therefore, the values of current or voltage can be fed to the load controlled by turning the switch between supply and load on and off at a fast pace. Trying to supply a higher power to the load? Just longer the time switch is on.

**Things to do the next meeting:**

* Test Eric’s PWM hypothesis
* Learn more about what PWM is
* Look back at quadrino-ng code for guidance

**12/18/2013**

***Attendance:* Eric Tendian, Boyang Dong**

**What we did today:**

Today was another big accomplishment. We started out by continuing on yesterday’s work, trying to implement a new PWM method. After connecting the Uno32 to the receiver, we tried to get a pulse reading, but only came up with seemingly random numbers. We tried changing the inputs to get better pulse readings, but no luck. We then tried using an Arduino Uno instead of the chipKIT board, but got similar results.

After doing some more research, we found these articles:

<http://www.altitudehobbies.com/ipeaka-22a-brushless-multicopter-esc-simonk> - this was on our ESC specifically, and gave some information on how to callibrate it. Because there are many different Tx/Rx combinations with the ESC’s, the ESC must be able to interpret the different signals that may be sent by the transmitter. We tried calibrating the ESC using the steps listed, and it worked! However, we still did not get accurate data.

The next article we founds was this: <http://robots.dacloughb.com/project-2/esc-calibration-programming/>

This article was key to our understanding of the ESC. As it turns out, you can tune the ESC to accept different PWM signals, which you can then base your power values off of. We copy-pasted the code shown in the document and tried it out. The ESC went through the proper beeping sequence, so next we added a line in the loop() function that set the PWM signal to be 1500microseconds (or around 60% power). This worked well and we had a spinning motor.

We then made some more changes to the program, adding a function to set the power via the serial monitor. The user would input a value between 0 and 100, then using the map() function, we converted that to a duration between 700 and 2000, which we then used the writeMicroseconds() function to change the PWM signals being sent. This worked out beautifully the very first time.

After that, Bobo had a great idea of using a small joystick to control the motor. As we already had prior code to control a different motor, a step motor, we were able to adopt it to fit our new brushless motor. We made some changes so that when the joystick was moved either forward or backward, the motor had positive power (using the absolute value function abs()). Using the map() function we mapped the 0-512 range of the joystick (actually 0-1023 without absolute value) to the 700-2000 range of the motor. This worked great as well, and our final code is shown below.

See this as a Gist: <https://gist.github.com/EricTendian/0b22b5bc447865a1be6f>

**#include <Servo.h>**

**#define MAX\_SIGNAL 2000**

**#define MIN\_SIGNAL 700**

**#define MOTOR\_PIN 9**

Servo motor;

**int** joyPin1 **=** 0; *// slider variable connected to analog pin 0*

**int** joyPin2 **=** 1; *// slider variable connected to analog pin 1*

**void** **setup**() {

Serial.begin(9600);

motor.attach(MOTOR\_PIN);

*// the following code should be uncommented for calibrating the ESC*

*/\*Serial.println("Now writing maximum output.");*

*Serial.println("Turn on power source, then wait 2 seconds and press any key.");*

*motor.writeMicroseconds(MAX\_SIGNAL);*

*// Wait for input*

*while (!Serial.available());*

*Serial.read();*

*// Send min output*

*Serial.println("Sending minimum output");\*/*

motor.writeMicroseconds(MIN\_SIGNAL); *//so the motor doesn’t start moving automatically*

}

**void** **loop**() {

*// reads the value of the variable resistor*

**int** p **=** analogRead(joyPin1);

*// this small pause is needed between reading*

*// analog pins, otherwise we get the same value twice*

delay(100);

*//while (!Serial.available()); //wait for user input*

*//float p = Serial.parseFloat(); //get a number the user has entered*

**int** power **=** map(abs(p**-**512), 0, 512, MIN\_SIGNAL, MAX\_SIGNAL); *//map the difference from the center to the min/max signal of the ESC*

motor.writeMicroseconds(power); *//write the power value*

Serial.println(map(abs(p**-**512), 0, 512, 0, 100)); *//print out the power as a number between 0 and 100 (like % power)*

}

Watch a video of us explaining what we accomplished today, and get a demonstration of the final program: <http://youtu.be/j-23svFmWm4>

Another important discovery we made was that by wiring up the GND and +5V pins of the joystick and ESC to the Arduino, we did not need to keep the Arduino plugged in for power. This was because as long as we had the battery connected to the ESC, the ESC distributed power from the battery down the jumper wires attached to the signal, and so it powered the Arduino. This is also why the receiver does not need any extra power while connected to a powered ESC. We hope to make use of this when we begin wiring everything together.

As this was the last meeting of the year, we ended it quite successfully.

**Things to do the next meeting:**

* TBA
* **Next meeting will be week before semester starts, check mailing list for more info later**

10/31/2014

Attendance: Manoj, Shoaib, Yi, David.