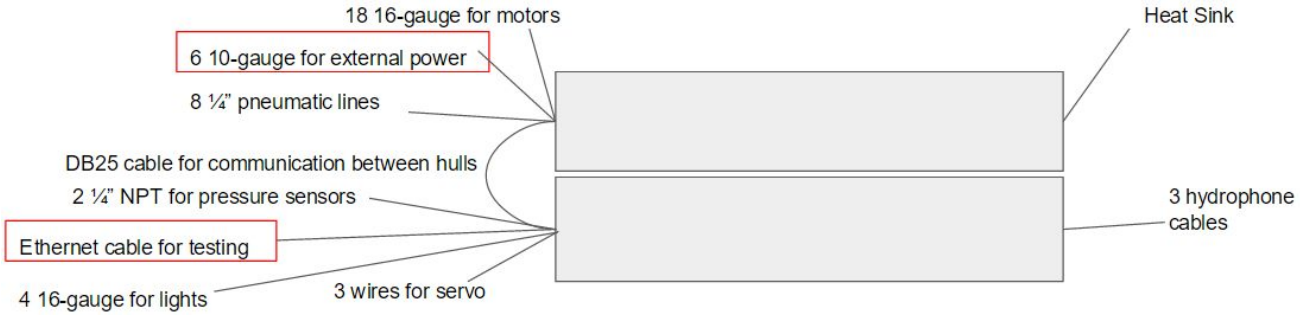
Operations Manual Trident Autonomous Submarine

**Mechanical overview**

- *Bracket mounting*
    1. Identify mounting location
    2. Position top bracket with hardware attached (bolt-plate-bracket-plate-nut)
    3. Slip on lower bracket, secure with same order as above
    4. *Optional: secure with ABS slurry (4-5 parts acetone:1 part ABS)*
        a. The ABS *must* dissolve fully before being applied
        b. Make sure to use metal tools to stir and/or apply , if possible
- *Bracket unmounting*
    1. Identify attached components
    2. Partially or fully unscrew components from selected brackets
        a. If there are multiple attachment points, or if the component is sufficiently rigid, it may not be necessary to fully unscrew it
    3. Verify that no additional hardware is attached to bracket to be removed
    4. Unbolt plates, remove hardware
    5. Remove bracket
- *Changing tubes*
    1. Follow "*bracket unmounting"* procedure
    2. Disconnect any cables leading to attached components or other tubes
    3. Verify that no cables or attachment hardware is interfering with the tube
    4. Remove/replace tube
- *Mounting additional hardware*
    1. Ensure that distance between mounting holes is 4.5inches (114.3mm, **±1mm maximum)**
        a. The brackets were designed for #6-32 bolts/nuts, but larger *may* fit
    2. Attach hardware
        a. It may be easier to first attach the hardware before mounting the brackets
    3. Ensure that adequate mounting points were used, e.g. part is not "floppy" or weak
- *Thruster mounting/unmounting*
    1. Follow "*Mounting additional hardware"* procedure
    2. The brackets are designed *for* the thrusters, so there should be zero issue with mounting them in any orientation
    3. Ensure that *all four small screws* hold the thruster onto the thruster plate before mounting the thruster plate to the brackets
    4. If needed, additional thruster plates can be 3D printed from a Solidworks model from the BlueRobotics website

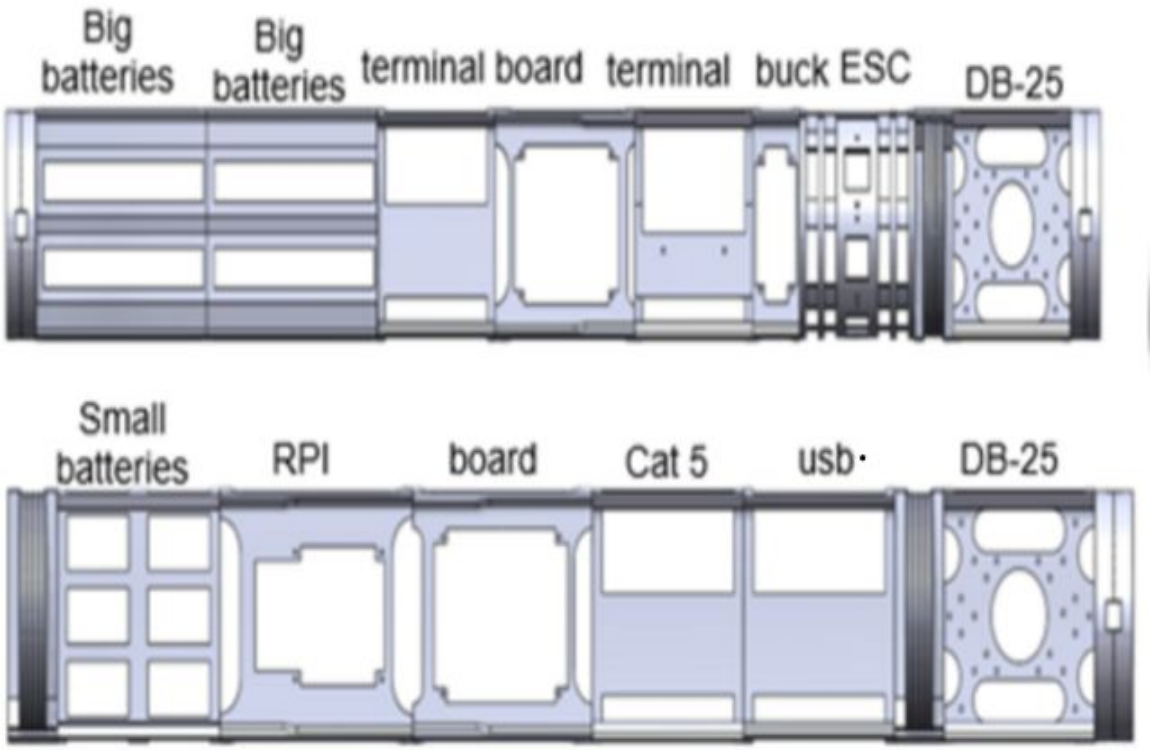
- Through ports
    - How to put cameras in and hold inside tube
        - First, the cameras must be bolted to the camera mount

            - Do not screw the bolts too tight, rather ensure that the camera
        - Insert mount with camera into hull
            - Mount will have to be twisted in
            - Don't force it
                - If it doesn't fit, adjust the cameras on the mount
    - How to put the end caps on
        - Ensure that the inside of the tube and all 3 rubber seals on each cap are well greased
        - Insert and bolt the through port bolts on each aluminium cap
            - Follow the diagram below for cable connection locations along the tube



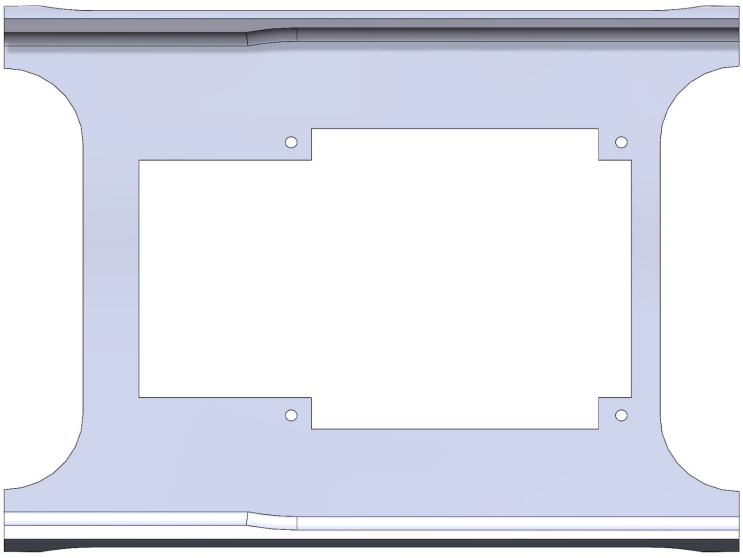

        - Attach end cap to aluminium flange, screw in bolts
            - Tighten bolts on alternating sides to ensure an even seal
        - Force cap assembly into tube
            - It helps to have one person hold the other end of the tube as you push the cap in
            - If the cap is very hard to force in, make sure the cables are not getting caught in the connection and apply more grease
    - How to disconnect/connect the ethernet teather
        - To attach, first wrap male pipe thread with pipe tape (teflon tape)
        - Screw on to highest allowable torque (don't use pneumatic tools)
        - This will work for the bolts and adaptors
    -
- Interior(Feras and mansour)
    - How to assemble interior portions

The interior design has multiple sections. As shown in the next figure ()



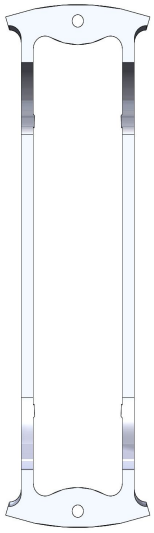

- How to attach boards

Each section was made to hold a specific part. There are some holes in order to use screws to attach the parts.



- How to slide whole sections in and out

The sections will be attracted to each other by using 2-56 screws.



After putting all the needed sections you can slide it in the tube.

- How to sand down parts if necessary

If the tube are not perfectly rounded. These sections can be sanded down. So can the outside part of the holding sections

---

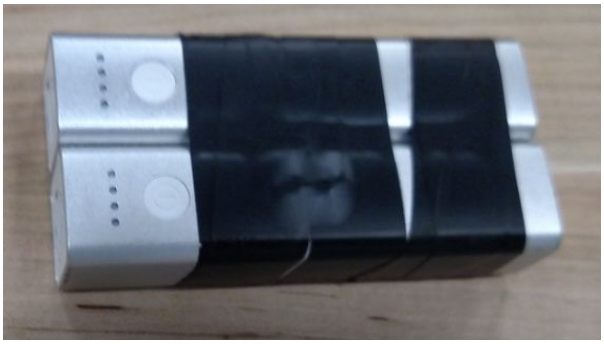
# Electrical overview

## Control section

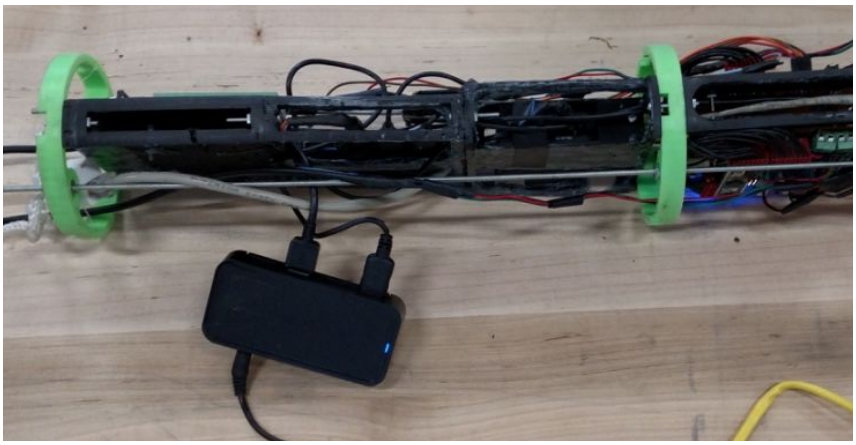
### How to power up

How to charge control batteries:

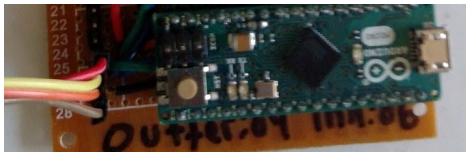
Connect micro usb cables to charge up, look at the manufacturer's pamphlet. Pretty self explanatory, it’s a pair of the extra phone power batteries.



How to plug in with usb hub:
Plug in the usb hub, as pictured below.



## How to wire up I2C line, for debugging:



Red Wire = 5v

Yellow Wire(changes elsewhere in sub) = SDA

Orange Wire(changes elsewhere in sub) = SCL

Brown Wire = ground

**Don't have motors on while doing this, with the current glitch, or without the sub in the water.**

## Why is the DB25 cable exposed?

There was an EM problem with the com cable between the sub hulls. There might be some extra inductance that wouldn't make the cable work, when we take the metal coupling off the end of the DB cable, everything works, so that's what we did.

# Ethernet wiring

Plug the ethernet going through the inside of the end cap and into the cat5 coupler. Do not cut the inside ethernet wire on the end cap unless you really have to, because once it gets

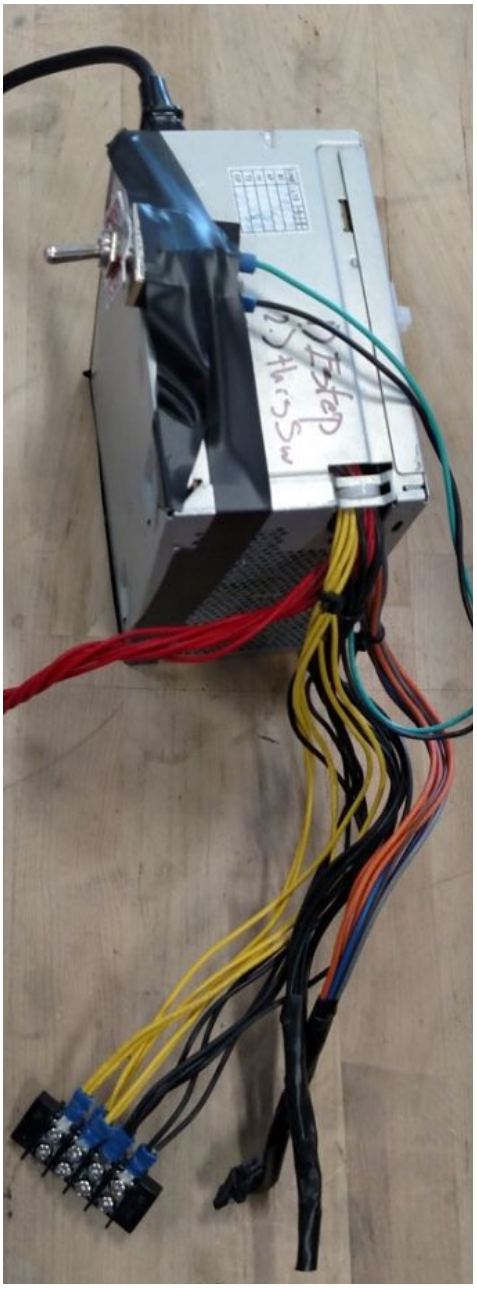
too short, it will be almost impossible to plug in when adding extra wiring, look at how to on the internet, it would be better than anything I can say.

# Power section

## How to power up (main batteries)

**Charging batteries**

The four main batteries have never been charged before but still have a charge from the factory. With the computer power supply attach the battery turnigy charging unit to the 150W power supply, pictured below. For each battery repeat, the charger balances the cells so that all the cells won't dump into each other. The switch that is taped onto the power box is for turning on the power supply.



**Balancing batteries externally:**

The batteries might not have the same voltage between the batteries ends. This means that the voltages have to balanced between the 2 hot ends of 2 different batteries. This can be done with a power resistor that has some 5mm bullet connectors soldered on. The ground of all the batteries can be connected and then attach a power resistor 1 at a time between all the hots of the batteries. Think about it, for we never got around to the actual battery implementation

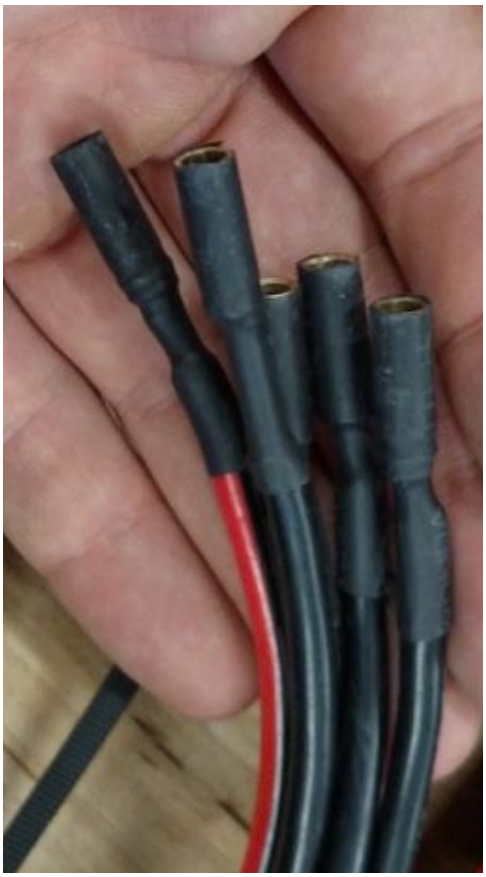
**Plugging in batteries :**

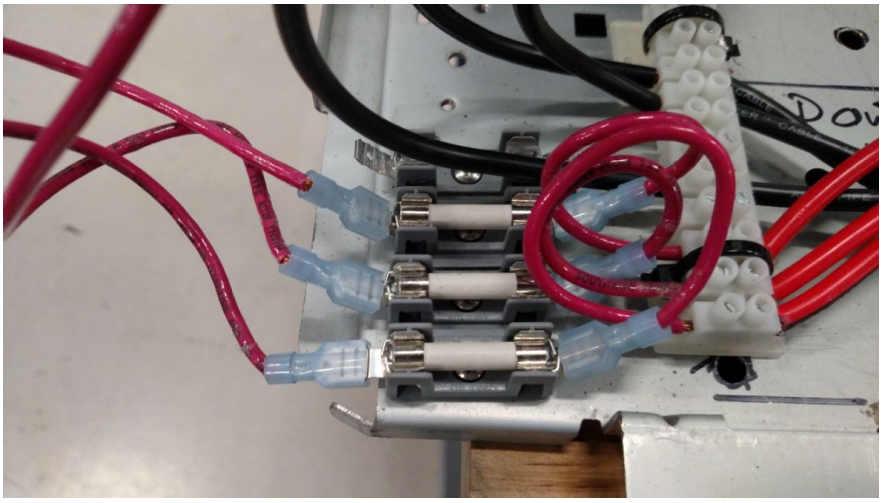
Connect the main batteries to the respective bullet connectors that connect to the terminal on the bottom side of the battery holder.

## How to power up (tether)

**How to wire up tether:**

When the submarine is not in main battery mode, the power tether needs to be used. There are six 10 gauge wires on the outside of the sub: three grounds and three 13.3V. These come off the server power supply that has 20A fuses for each branch of the power.

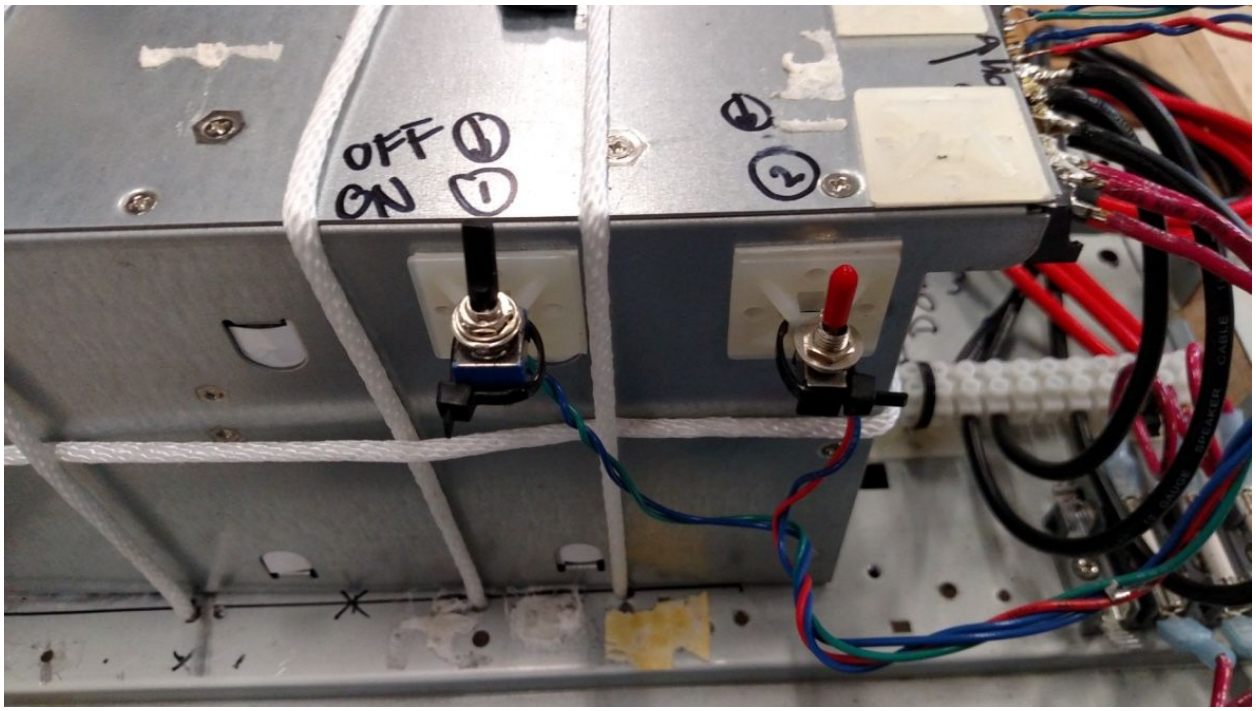




## How to flip buttons on server power supply:

The server power supply has to be turned on in 3 steps:

1.) Plug into wall outlet with nothing else on that breaker(maybe), the server takes 1kw of power which almost translates to 10 amps at 120vac which is a lot of power for a single breaker.
2.) Turn on 3.3v power, this is done with the blue switch
3.) Turn on the 13.3v power, this is done with the red switch
4.) When turning off, go in the exact opposite order. Also make sure that the E-stop on the sub is in the correct position to allow power to the ESC's.



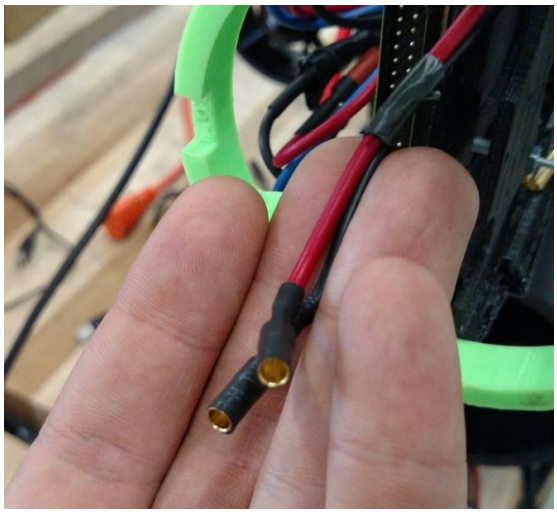
# How to wire up 3 phase motors

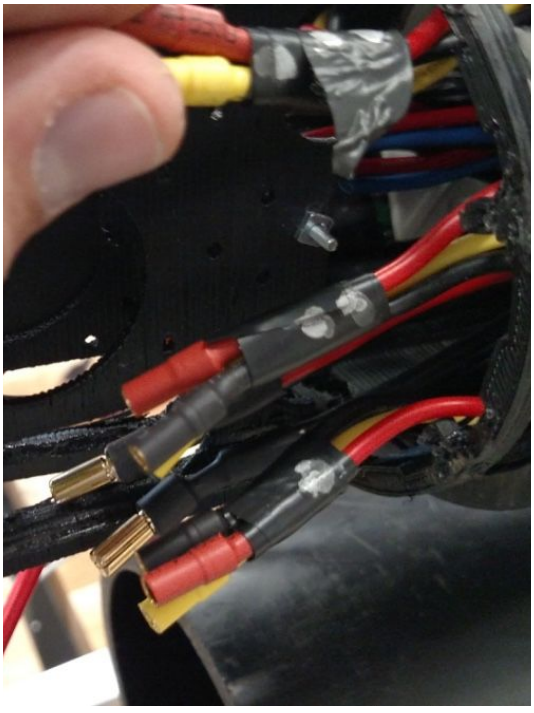
## Match wire codes:

The wires for the 3 phase motors have to be connected and disconnected every time that the power section needs to be take out. On the back of the power section there are 3 different connections:

- Power tether
- 3p motors
- E-stop connection
- DB25 connector

The power tether is comprised of simple 5mm bullet connectors . The 3 phase motors on the back are coded with the marks on the red wires of the 3p motor bundles. The estop wires come off the relay block which are the 2 wires that aren't bundled with a blue, therefore aren't a 3p motor connect. It is good practice to electrical tape all the connections so that there aren't any plug pull outs that could damage the sub





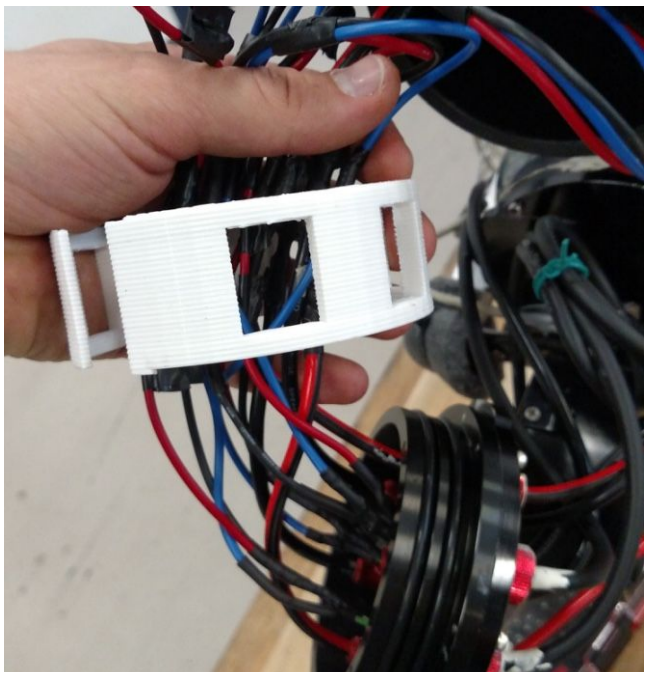
## Front power connection:

There are 2 types of cables coming out the front of the power section:

- Power tether
- 3p motors

The 3 phase motor connects on the front are labelled with tape on the red wires, attach them accordingly.

**NOTE** if wires get pinched, the waterproof seal will not work rendering your electronics wet. The white 3d printed part that was scraped works perfectly for making sure the bundle of wires on the front of the power section don't get pinched in the front power section.



# Software Overview

## Control section

### How to start ethernet tether communication:

#### Ping the corresponding IP:

This can be done with a computer that is on the same network and has access to the RPI. The RPI's are on a network that is 100.100.100.10# which means the RPI's are numbered for more detailed instructions on how to do this, look it up on Instructables.com

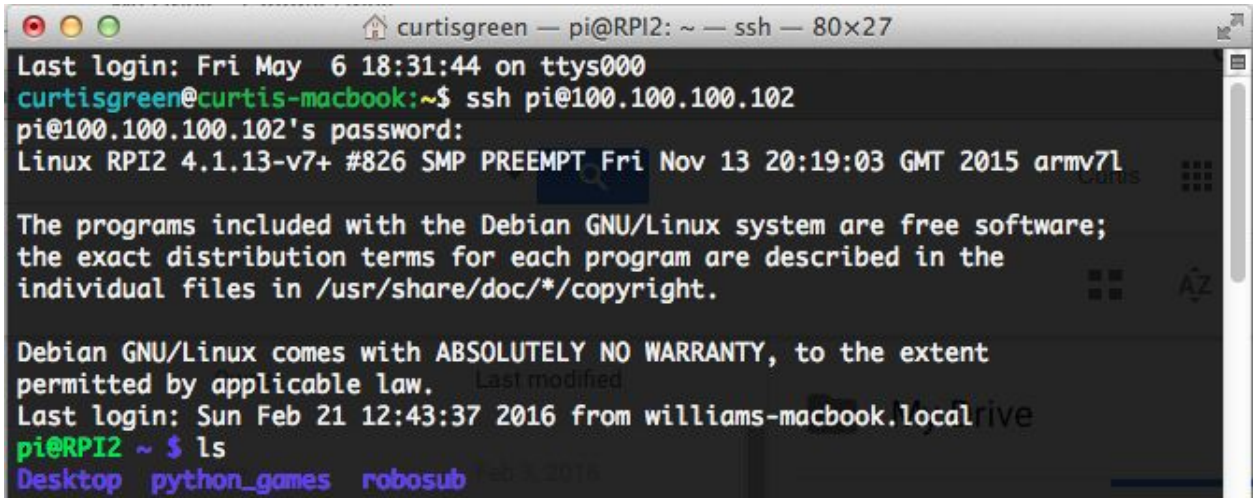
```
williams-MacBook:~ williamritchie$ ping 100.100.100.102
PING 100.100.100.102 (100.100.100.102): 56 data bytes
64 bytes from 100.100.100.102: icmp_seq=0 ttl=64 time=1.541 ms
64 bytes from 100.100.100.102: icmp_seq=1 ttl=64 time=1.502 ms
64 bytes from 100.100.100.102: icmp_seq=2 ttl=64 time=1.625 ms
64 bytes from 100.100.100.102: icmp_seq=3 ttl=64 time=1.655 ms
^C
```

#### SSH:

With a MAC this is very easy, with a windows computer, the user will have to download a putty program in order to access the linux terminal of the RPI.

#### Start up program:

Through the SSH terminal type in the comand "python FILE_NAME.py" to do things. This is to do hard command that don't start up automatically. The password for all raspberry pis is 'naurobosub', without the quotes. The user names are 'pi' , without the quotes.




# I2C Errors that may occur:

## I2C good detect

This is an example of what a good I2C detect should show.

```
pi@RPI2 ~/robosub $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- 04 -- 06 -- -- -- -- -- -- -- -- --
10: -- -- 12 -- -- -- -- -- -- -- -- -- -- 1d -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- 6b -- -- -- --
70: -- -- -- -- -- -- -- --
```

## I2C detects full addresses

This means that the SCL AND/OR SDA line are high. Checks for shorts on the spliter board. Also check that the SCA and the SCA wire haven't been switched. It doesn't hurt to switch them so try switch the wires.

## I2C detects no addresses

This means that the SCL AND/OR SDA line are low or not connecte, also look at the previous section's notes.

## I2C detect no 04, 06

There is a problem communicating the the other hull, check for wiring or that the DB cable isn't connected or is too noisy for various reasons.

# IMU

## Change of IMU address:

For some reason there was a address change on the IMU from the specified address from the python code from github. There are 2 possible pairs of IMU addresses which the IMU can use. These address are selected by setting one of the pins on the breakout board high or low. This may unintentionally happen when there is high capacitance on the line. Or maybe there was a revision number error on the file, either way this is a problem we would sometimes run into.

## Other operations:

There are 2 main ways of thinking about the IMU code that was developed: one was a timed gyroscope data return while the other was a non-timed orientation return.

In all honesty there is about 10% reusable code from the IMU, it's some very ugly time travelling spaghetti with some trig problems sprinkled on top, with a dash of initialization confusion. Before you know it the spaghetti is on the roof, however it was meant to be there, it's ugly.

But anyway the 1st algorithm was to have a queue send data back from the IMU which the gyroscope is time sensitive. There's 2 main threads running in this one, one that checks the BD queue com that will kill the other thread when a 'False' is sent down it. The other thread automatically requests data from the IMU, does some trig on the data, then sends data into a queue which can be done with in whatever fashion on the motor control side.

The second algorithm isn't time dependant and only has one thread running. This thread will run a main cycle that would have been ran automatically in the other algorithm. This set up is good for sending back data that is non-time dependent such as the acceleration and the magnetometer.

The initialization process might be hard to make so that the data is normalized.
Also there' the _returnData method which can be changed in order to get the wanted data back to the requesting  core. This could be updated into the future so that the return method takes in parameters so that the data that is sent back would be dynamic.

# Visual

Camera: Line detection:

There are several parameters that can be adjusted throughout the line detection program for very fine adjustment. As is, the program loops through the threshold parameters right after capturing the image, which is what makes the program slow. For each parameter a chain of function calls is done:

```
#take picture
cap = cv2.VideoCapture(0)
_,frame = cap.read()
#set lines to None to enter while loop
lines = None
while(lines == None):
    for i in range(256,0,-2):
        #threshold to take inverse of image
        _,threshold = cv2.threshold(frame,i,255,cv2.THRESH_BINARY_INV)
```

The underscore in front of the threshold function is to unpack both return values from threshold, the first isn't used. The way threshold function works is by taking the image (frame) applying a binary inverse threshold, which can also be changed, looking at every pixel in the image classifying it a particular way if it is greater than given pixel value and different way if it less than given pixel value. This threshold type (THRESH_BINARY_INV) seemed to produce good results, although in order to find the color that was being searched for the image had to be shown to the screen right after the threshold was applied which is now a different color. For example when searching for orange, orange colors came back light blue after the threshold was applied. So the color array had to look for light blue not orange. The way the for loop operates can also be changed in an attempt to optimize the threshold step. Notice the increment by 2, it likely that is enough and the program will find lines. Increasing the increment is desired but the program may miss lines. More information on threshold types can be found by searching the web for openCV documentation, it is readily available. The color array can be seen later on in the code.

This next piece of code is a continuation of the inner for loop. The color conversion to HSV from BGR is suggested since HSV is a more workable color scheme. Also openCV sees images in reverse so an RGB image shows up initially as BGR. The color sets for lower_blue and upper_blue are in HSV. They surround the exact HSV color set for light blue to give leeway when searching for the desired color. There is a program called hsv_finder.py that will find HSV color sets. The inRange function creates a binary mask using the color sets. This is where the color is detected.

```
#convert image from BGR to HSV
hsv = cv2.cvtColor(threshold,cv2.COLOR_BGR2HSV)
#color arrays currently detecting light blue when on inverse threshold
lower_blue = np.array([90,90,150])
upper_blue = np.array([110,255,255])

#mask to filter filter color arrays
mask = cv2.inRange(hsv,lower_blue,upper_blue)
```

This last piece is the last part of the inner for loop and determines when to break from outer while loop. The medianBlur option is commented out and currently not used but it can be implemented to blur out excess unwanted noise. The canny function detects any edges created

by the mask. The details of the canny input parameters can be found in openCV documentation. The HoughLines function decides if an edge is long enough to be considered a line.

```
#blur to filter noise
#blur = cv2.medianBlur(mask,15)

#detects edges of blurred image
edges =  cv2.Canny(mask,5,20,mask,3,True)

#decides if any eggdes are worthy of being considered a line
lines = cv2.HoughLines(edges,2,np.pi/180,120)

#if lines are created break from for loop
if lines != None:
    break
```

At any point in the program the current image can be displayed with the follow code. This will pause the code and show whatever image is chosen until the escape key is pressed. This is very useful for debugging and adjusting the code. For example, this will show the mask if inserted in the inner for loop right after the mask line and will pause the program until the escape key is pressed. The name ‘mask’ is just the name of the window that pops up, this can be anything.

```
cv2.imshow('mask',mask)
k = cv2.waitKey(0)
if k == 27:
    cap.release()
    cv2.destroyAllWindows()
```

The rest of the code found in this program is to draw the found lines on the original image.

Camera: Buoy detection:

The buoy detection program works very similar to the line detection algorithm except utilizes the HoughCircles function to find colored circles. Refer to the code for the full algorithm. The program has not been as developed as the line detection program

```
circles = cv2.HoughCircles(blur2, cv2.HOUGH_GRADIENT, 6, 100)
```

Camera: U-Shaped PVC detection:

This program is currently under developed but the main function used is findContours. Refer to the code for further details.

```
_,contours,_ = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

## OTHER NOTES

The electricals used the free electrical CAD program called “tiny CAD” download to alter the files.