ME 424 Engineering Design VIII

Final Report

MATE ROV

Underwater Robotics Competition

Group ME-07:

Stephanie Senkevich

Chris Stollen

Kevin Grudzinski

Advisor:

Dr. Frank Fisher

“I pledge my honor that I have abided by the Stevens Honor System”

Stevens Institute of Technology

Castle Point on Hudson

Table of Contents

Glossary of Terms

- ROV - Remotely Operated Vehicle
- MATE - Marine Advanced Technology Education
- Raspberry Pi - Microcontroller board
- Beaglebone Black - Microcontroller board
- Scuttle - The sinking of a ship
- Bow - The front of a vessel
- Stern - The rear of a vessel
- Heave - The upwards/downwards motion of travel
- Surge - The forwards/backwards motion of travel
- Sway - Forward/backward side-to-side movement
- Yaw - Twisting about the vertical axis
- Roll - Right/left side-to-side movement
- Top-side - System components located above the pool surface
- AWG - Standardized American wire gauge
- DC - Direct Current (Power)
- Bulkhead - The push-pull cable termination end
- IMU - Inertial Measurement Unit

Abstract

Project Change

Our team initially began as the Autonomous Surface Vehicle (ASV) Roboboat Competition Team. However, we decided to switch our project to the Marine Advanced Technology Education (MATE) Center’s Remotely Operated Vehicle (ROV) underwater robotics competition. We made this switch because the ASV project had too large a programming aspect for our team of solely mechanical engineers. Previous teams had been interdisciplinary, with electrical and computer engineers on the team. Another factor initiating the change was that the pre-existing boat would require very minimal mechanical engineering input. Lastly, the updated competition rules would not be released until mid to late December so our initial research on the competition was based off of the previous year’s rules.

MATE ROV Competition:

The Marine Advanced Technology Education (MATE) Center hosts an annual remotely operated vehicle (ROV) underwater robotics competition. This competition is held at the beginning of May and this year will focus on shipwreck remediation and exploration where remains have fallen to the bottom of the ocean. Over the past two semesters, we have successfully designed, fabricated, and tested an ROV that is capable of exploring and documenting a simulated shipwreck at the bottom of a 20 feet deep pool for the competition. Controlled live by a pilot from ashore, the ROV can navigate through the makeshift shipwreck and also collect microbial samples, test the conductivity of the water, and remove trash and debris from the shipwreck’s surrounding area. The ROV has been designed to be modular, so components can be changed and relocated from year to year. The ROV has also been constructed with a budget substantially less than most other teams. We intend to prove our design at the upcoming competition.

Introduction

Our remotely operated vehicle (ROV) will be designed to execute several specific tasks specified by the Marine Advanced Technology Education (MATE) Center for its 2014 MATE ROV competition. The theme of this year's competition is exploring shipwrecks, investigating sinkholes, and performing conservation tasks in the Thunder Bay National Marine Sanctuary in Lake Huron. Shipwrecks can be very dangerous for humans to explore so robots are necessary to complete the task. These robots can take video footage and survey the area and obtain data for humans to analyze. This design project challenges the team to develop a functioning ROV robot that meets specific requirements set forth by the competition.

The MATE ROV Regional competition will take place on May 10th at Rowan University. If the ROV successfully passes the demonstration and safety check, we will move on to the International Competition, which takes takes place in Alpena, Michigan on June 26-28. There, the team will compete against over 20 other teams.

Throughout the course of the past two semesters our group has remained on task to successfully design, fabricate, and test an operational ROV that fulfills mission requirements. The team has done so with a modular design that can be easily modified for future years and at a budget that is substantially less than the the budgets of most of the other competing teams.We have had the opportunity to test the ROV's functionality in the water and it has successfully performed. Our current focus is to continue to test and fine tune our ROV to fully prepare it for the Regional Demonstration Day, and then based on feedback there prepare it for the International Competition.

Competition Background

The MATE ROV Competition is unique. Focusing on more than just engineering skills, the MATE ROV Competition challenges teams to think as an entrepreneur, and to develop the ability to understand the breadth of business operations. The MATE ROV Competition is broken into 4 different levels called classes, which are Scout Class, Navigator Class, Ranger Class and Explorer Class. Our team will be competing in the Explorer Class competition, which is designed for university level and experienced high school level teams.

Figure 1: Picture of a Shipwreck on the ocean floor

Competition Overview

The competition involves three mission tasks that are split into three themes:

1. Explore, document, and identify an unknown shipwreck recently discovered in sanctuary waters.
2. Collect microbial samples and measure the conductivity of the groundwater emerging from a sinkhole.
3. Remove trash and debris from the shipwreck and surrounding area.

Teams are given a time limit of 15 minutes to complete all tasks. The competition takes place in a 20 ft. deep indoor pool. No water currents will be intentionally created. The robot must conform to an extensive list of requirements and constraints that are outlined in the competition manual. These specifications involve documentation, safety, mechanical properties, and electrical requirements.

Beyond the main underwater missions, the teams will be scored on a technical report, an engineering presentation, a poster display, and a safety inspection with the following scoring breakdown, for a total maximum score of 580 points:

- Mission - 300 points
- Written Technical Report - 100 points
- Oral Engineering Evaluation - 100 points
- Poster Display - 50 points
- Safety - 30 points

Mission Tasks

The mission consists of 3 tasks that our ROV will need to perform, with a time limit of 15 minutes. Tasks may be done in any order and teams may switch between tasks freely. Teams may remove the vehicle from the water for troubleshooting, buoyancy adjustments, and payload changes, but the timer will not stop. Teams will receive bonus points for completing all tasks before the mission time ends and penalties for exceeding the given time or leaving debris on the pool bottom. In addition to the mission time, teams are allotted 5 minutes to setup and test the vehicle at the mission station and 5 minutes to break down and exit the mission station. A maximum of two mission attempts will be allowed per team, with the higher scoring attempt used toward the overall score.

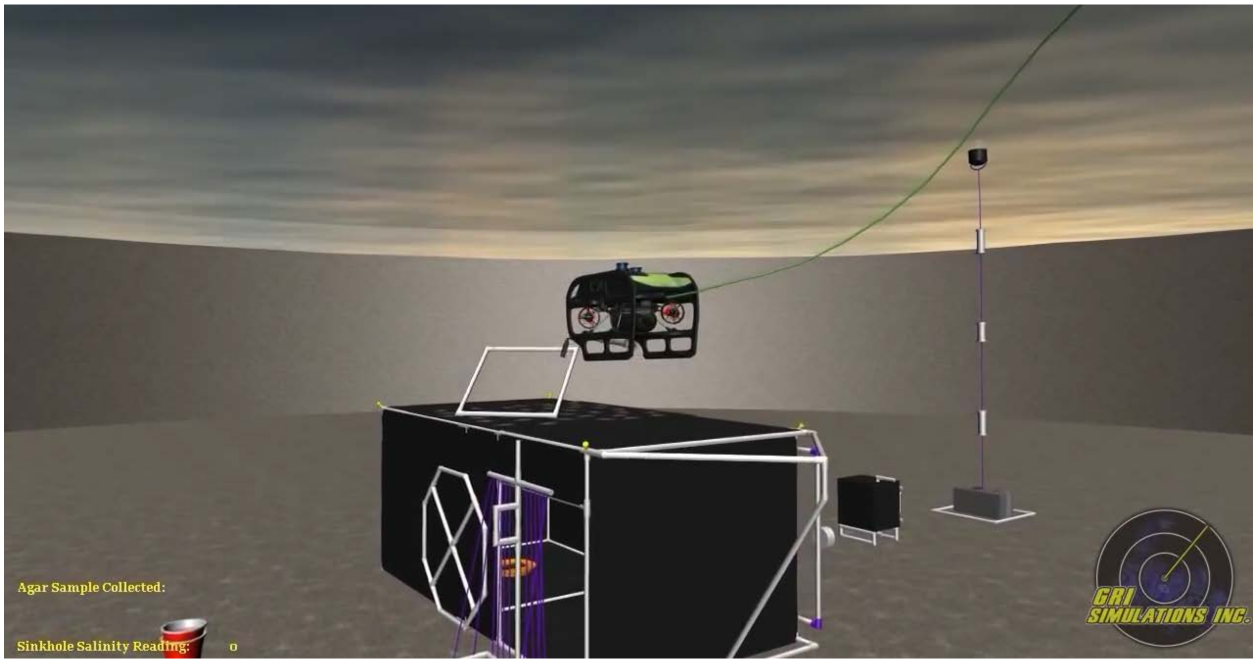


Figure 2: Simulated shipwreck environment

Task 1 - Shipwreck

The first task is to explore, document, and identify a newly discovered shipwreck. The ROV will complete several different tasks to piece together the identify of the wreck. Using all of the features, the team will identify the ship from a list of 24 possible wrecks. This first task will involve the following steps:

- The length, width, and height of the shipwreck will be measured.
- The shipwreck will be “scanned” with sonar. This is simulated by taking pictures of the wreck at three different target locations. The ROV must maintain alignment for five seconds.
- A photomosaic of images taken at five distinct locations will be stitched together.

- Visual evidence of a propeller, paddlewheel, or mast head will be found to identify the type of ship. This will allow the team to determine whether the ship was a bulk freighter, a paddle-wheel ship, or a sailing schooner respectively.
- A cargo hatch will be unlocked by turning a PVC handle to open the cargo container. Then, the door of the cargo container must be opened to allow for identification of the cargo. No cargo must be returned to the surface. After identification, the cargo door will be closed and the hatch relocked. The handle and cargo door will take less than 2 newtons to turn and open.
- The ROV will enter a 75 cm X 75 cm hole near the bow of the ship. Debris will be located in front of this hole that the ROV must remove before entering the wreck.
- The date the ship was built will be etched on a 5 cm x 15 cm piece of black plastic. There will be limited visibility inside the shipwreck so a source of light may be required. The date on the video feed must be shown to the judge.
- There will be a plate at the bottom of the ship that has its home port on it. This plate will need to be found and brought back to the surface. There, the crew will remove debris from the plate and identify the home port of the ship.

Task 2 - Science

The second task involves collection of data, collection of a sample of a microbial mat, replacement of a sensor string, and a calculation of the number of mussels on the exterior of the shipwreck. The individual steps of this task are:

- Investigating a sinkhole in the site area and using a conductivity sensor to measure and record the conductivity level of the groundwater that will be overflowing out of the sinkhole. The sensor will have to be inserted into the sinkhole at least 7 cm to get an accurate reading. The sensor may be incorporated into the ROV or be independent of the vehicle.
- Retrieving 150 mL of microbial mat (simulated by plastic cups full of agar) from a cup near the sinkhole and returning it to the surface.
- Returning a sensor string to the surface and replacing it with a new one to the same specific area on the pool floor. Sensor strings will have a 2 lb dive weight attached to its bottom. The strings will weight less than 25 newtons underwater.
- Using a 50 cm x 50 cm quadrat to measure the amount of zebra mussels on the top of the ship. The dimensions of the wreck calculated in task 1 will be used to calculate the overall amount of mussels covering the shipwreck. The bottom surface of the ship does not need to be calculated. Subtraction for the area of the hole does not need to be made.

Task 3 - Conservation

This task involves removing debris from the site area. The steps for this task are:

- Removing a capless 1 liter plastic bottle from the pool floor and returning it to the surface
- Removing a capless glass bottle from the pool floor and returning it to the surface
- Removing the anchor line rope debris that blocks the hole in the shipwreck and returning it to the surface. The debris will weight less than 10 newtons.
- Removing an 8 lb danforth anchor with 1.5 meter long chain attached from the bottom of the pool and returning it to the surface. The anchor and chain combined will weigh less than 100 newtons underwater.

Specifications/Constraints

The competition manual and rulebook lays out many requirements and constraints that our ROV must satisfy. Combined with the mission requirements, the team identified the core requirements that our ROV must satisfy in order to succeed in the competition. These are the technical requirements that we designed our ROV to meet.

Technical Specification	**Reasoning/Rule Reference**
ROV must grip items ranging in size from ¼” to 2”.	Must move dinner plate and PVC pipe to surface
ROV must be able to lift and grip items weighing up to 100N (22.5 lbs)	Max weight of anchor and chain
ROV must be able to hold position for 5 seconds	Sonar mission task
ROV must have color camera	Required for mission tasks
Electronics housings must be waterproof to 6 meters (20 feet)	MECH-001
ROV must have a frontal area less than 75 cm x 75 cm	MECH-002: ROV must fit into shipwreck
ROV must weigh less than 75 lbs. out of water	MECH-002: Vehicle must be hand launched
Tether length must be at least 20 meters	MECH-003

All power must be obtained from MATE supply with 40 amp fuse (no onboard batteries)	ELEC-001
ROV system must operate on power supply of up to 56 Volts, with expected nominal voltage of 48 VDC	ELEC-002
Any Supply voltage modification must take place on ROV	ELEC-003
No voltage above 48V is permitted anywhere in the ROV system	ELEC-005
Must have 40A fuse or circuit breaker in positive power supply line within 30 cm of attachment point	ELEC-008
Power supply connections must connect via ¼” bolt with wing nut	ELEC-010
All electrical connections must be sealed and not exposed to water	ELEC-016
“Disposable motors” (exposed motors without waterproofing) are not permitted	ELEC-017
ROV must shutdown within 5 seconds of loss of surface power supply	ELEC-019

Design

The ROV chassis is comprised solely of standard 1 inch diameter PVC pipe and fittings, creating the framework for the fully modular system. All PVC fittings and pipe are primed and glued to prevent any water infiltration. Ballast tanks are comprised of twin 4 inch diameter PVC pipe with dual internal bicycle inner-tubes; the tanks are affixed with acetal polymer U-bolts, easy to remove, corrosion proof and lightweight. All circuitry including the DC-DC step-downs, microprocessors, etc. are encased in a 6 inch diameter acrylic tube. The tether inlet and outlet connections are made through a remote pipe-manifold so as to reduce the chance of flooding. Multiple cameras are individually housed and sealed in 1.5 inch PVC pipe to allow modularity to aid in adjustments. Thrusters are mounted with adjustable mounts capable of incremental angles and heights on any section of the PVC frame. The custom fabricated claw is powered with an actuator remotely through a mechanical control cable. All the ROV electrical wire entrances will be waterproofed using wire-gland seals.

Our ROV is controlled using software on two computers. The Beaglebone Black, a low cost single board computer, is located inside the waterproof enclosure on the ROV. It interfaces with all the sensors and motors and performs all of the control operations necessary to keep the ROV operational. A laptop on the surface provides a graphical user interface (GUI) which the operator uses to control the ROV. This interface shows the camera feeds, the input commands being sent to the ROV, and the sensor feedback from the ROV.

The latest version of all of the code is publicly available on GitHub, at https://github.com/kgrudzin/Stevens-MATE-ROV. A copy of the code at this moment in time can be found in the Appendix as well.

ROV Subsystems:

1. Chassis

The chassis was constructed with one (1) 10 foot length of 1 inch PVC pipe, in spec to the attached assembly drawing. To ensure buoyancy force reliability the chassis was glued and tested at depth for an extended period of time. A modular top support bar is used to fixture the top thruster, this is constructed with 'snap' connectors and is not permanently affixed or waterproofed. The picture below is representative of chassis shape, however no mounting holes penetrate the real model..

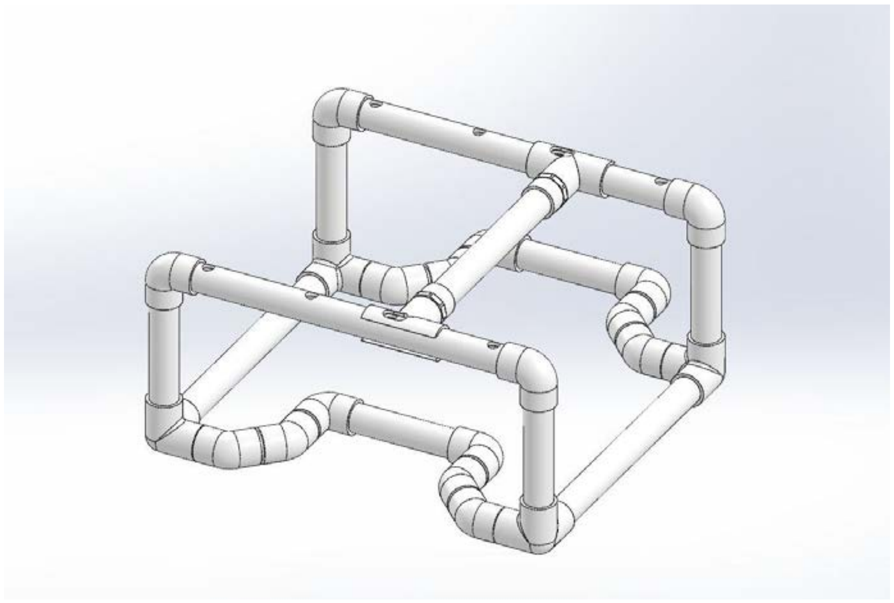
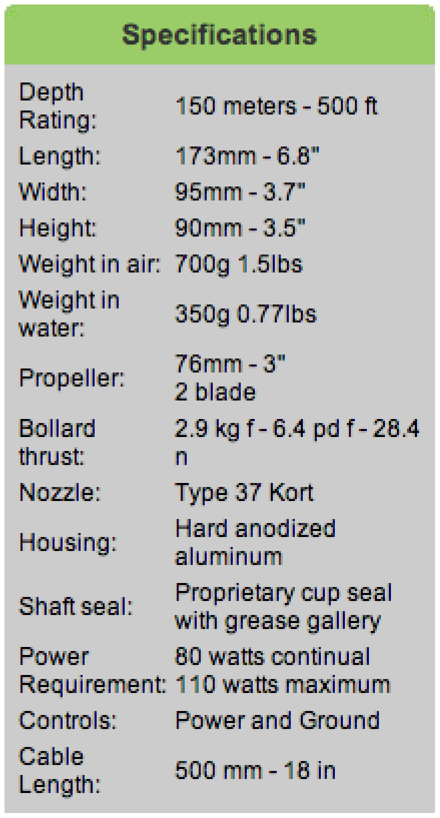


Figure 3: Design of our PVC Pipe Frame for our ROV

2. Thrusters

Thrusters for the MATE ROV project were carried over from the Autonomous Surface Vehicle Project and the naval project Bluedart. Attached in the appendix is the thruster documentation and functionality sheet, which was created to address disfunctional thrusters supplied to us initially. The thrusters are manufactured by Seabotix, the model specifications are listed below:



Specifications	
Depth Rating:	150 meters - 500 ft
Length:	173mm - 6.8"
Width:	95mm - 3.7"
Height:	90mm - 3.5"
Weight in air:	700g 1.5lbs
Weight in water:	350g 0.77lbs
Propeller:	76mm - 3" 2 blade
Bollard thrust:	2.9 kg f - 6.4 pd f - 28.4 n
Nozzle:	Type 37 Kort
Housing:	Hard anodized aluminum
Shaft seal:	Proprietary cup seal with grease gallery
Power Requirement:	80 watts continual 110 watts maximum
Controls:	Power and Ground
Cable Length:	500 mm - 18 in

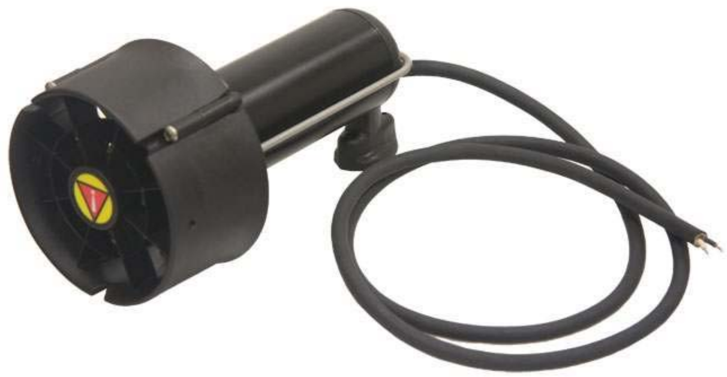


Figure 4: Picture here is the Seabotix thruster and specifications

Modular thruster mounts consisted of a flat-bar adapter to connect to the thruster, and a round clamp intended to mate to the 1 inch PVC chassis framework. These aluminum mounts were affixed with two (2) ¼-20 x 1.5 inch long stainless steel allen bolts. Engineering drawings for these components are attached in the appendix. This mounting design allowed for full adjustability on the ROV framework, allowing balancing to be conducted between the drag force and thruster force. The thruster configuration is in a vectored format to increase yaw maneuverability and reduce the chances of snagging obstacles.

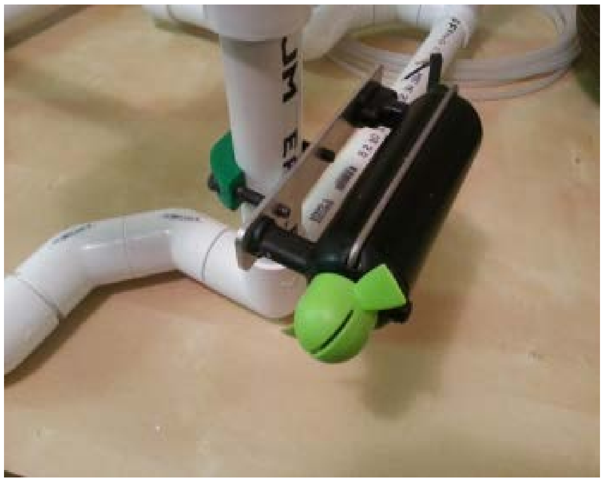


Figure 5: Thruster mount with prototype round clamp (in green)

3. Ballast Control

Neutral buoyancy of the ROV system is achieved with nearly empty ballast inner tubes and permanently affixed foam surrounding the bladders. For heavy lift applications the four (4) 25" inner tubes will be inflated through the tether cable. The inner tube pressure is regulated through a pair of waterproof air actuators, one for venting and another for filling. Variable pressure is kept in the tether supplied by a hand-pump to a maximum pressure of 40 PSI. The standard Schrader valve stems are adapted to the 6mm tubing with modified ⅛ inch NPT to ¼ inch barb adapters. To prevent back-fill of water into the ballast bladders, a miniature 1-way valve is connected inline with the exhaust port. To reduce weight and provide adequate venting the 4 inch PVC ballast tubes are slotted with 0.75 inch wide slots. The entire system is affixed modularly to the frame with 1.25 inch stainless loop clamps and aluminum binding bars. The picture below illustrates this configuration. For engineering drawings of the tube slots and aluminum mounting bar refer to the appendix.

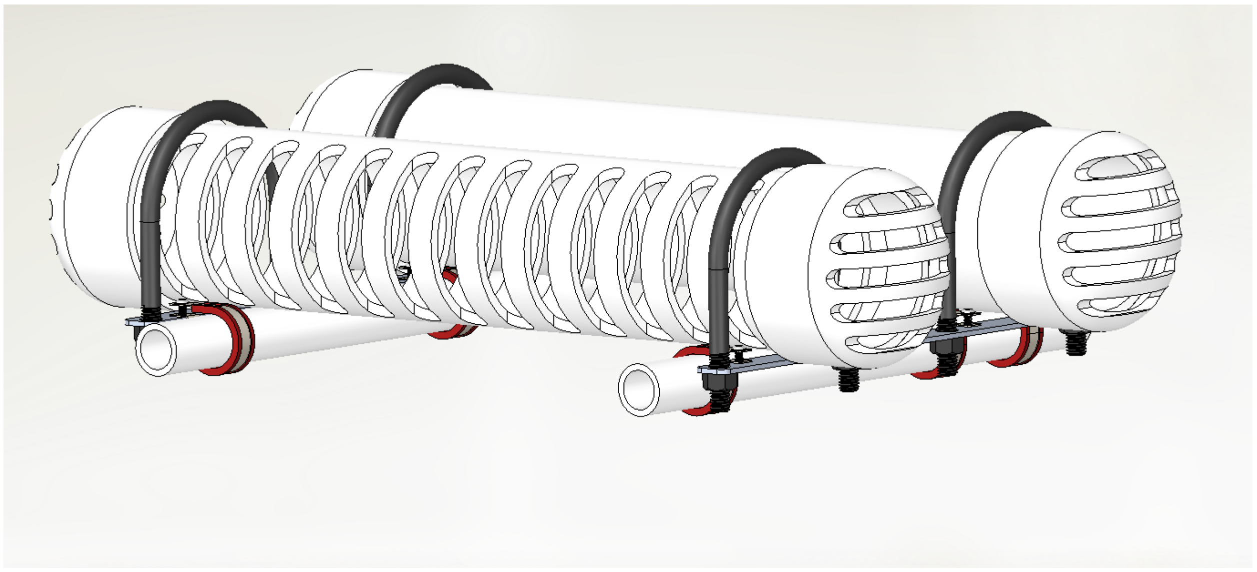


Figure 6: Ballast Tank design

4. Tether

The 50 foot long tether is comprised of 4 individual 0.25 inch lines: shielded ethernet, 8 AWG DC ground, 8 AWG DC positive, and 40 PSI air. In order to prevent kinking and knots, the line is affixed together at 12 inch increments over the whole length. These clips (seen below) are 3D printed and removeable in-line, making repairs very simple and fast. The clips fasten tightly to the air line to prevent bunching, while allowing the data and power lines room to independently shift while bending.

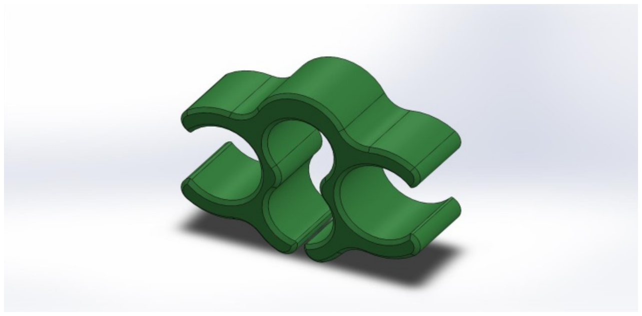


Figure 7: 3D printed tether clip

Tether top-side connections include two (2) ¼ inch lugs to connect to the competition power supply, a 15 amp fuse, and the air hand pump.

5. Actuated Claw

The actuated claw is modularly-oriented, designed to adapt to the 1 inch PVC chassis framework. Control of the claw is supplied by a 200 lb capable IP67 rated DC linear actuator as seen below. In order to transfer this power to the claw a high-strength stainless push-pull rod is interfaced between the two component systems. The cable is attached to the actuator through an adaptable clamping system. This system relies on aluminum clamps for strength, and plastic inserts for getting the right contours of the actuator body. In the event of a design change, new clamp inserts can be printed to change the mounting orientation of the actuator.

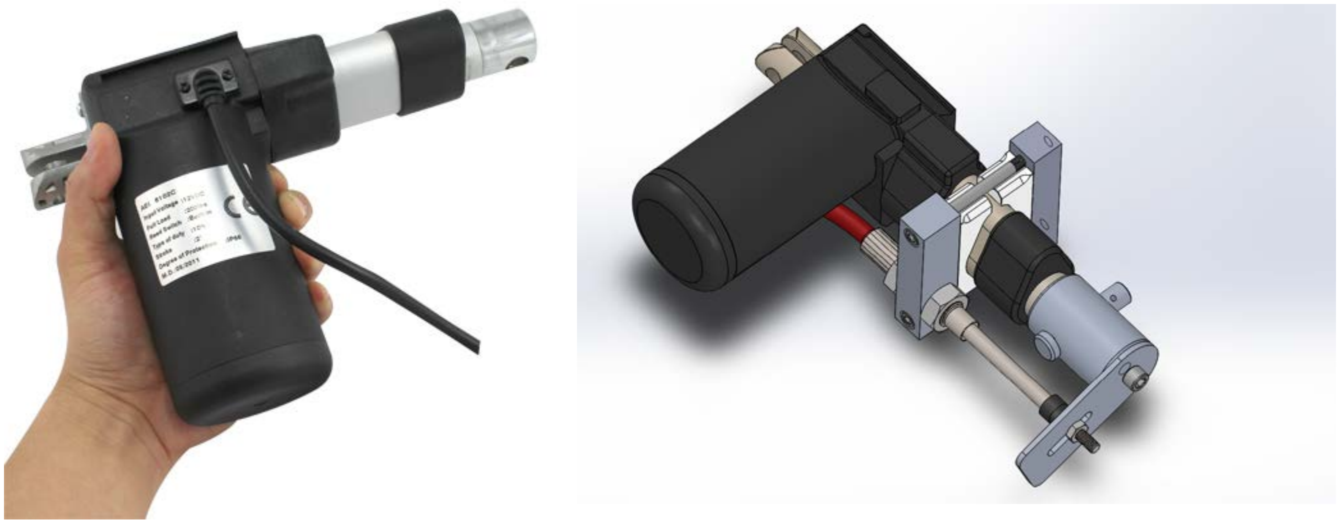


Figure 8: Picture and drawing of the Linear actuator

On the other end of the push-pull cable is the claw clamping and interfacing manifold, which is positioned to the front and center of the ROV. The claw, originally intended to operate on a parallel jaw design, has one fixed lower jaw and a moving upper jaw. The upper jaw operates as a complex 4-bar linkage, providing a motion pattern to assist the collection of debris. Upon closing, the upper jaw lowers and 'pulls-in' towards the lower fixed jaw, making the ROV's location less critical. In the event that a more effective jaw design is created the jaws are replaceable, through 3D printing, laser cutting, or machining. To affix the push-pull rod to the gripper mount the cable's bulkhead is clamped with six (6) set screws. The jaw design is shown below, engineering drawings can be found in the appendix.

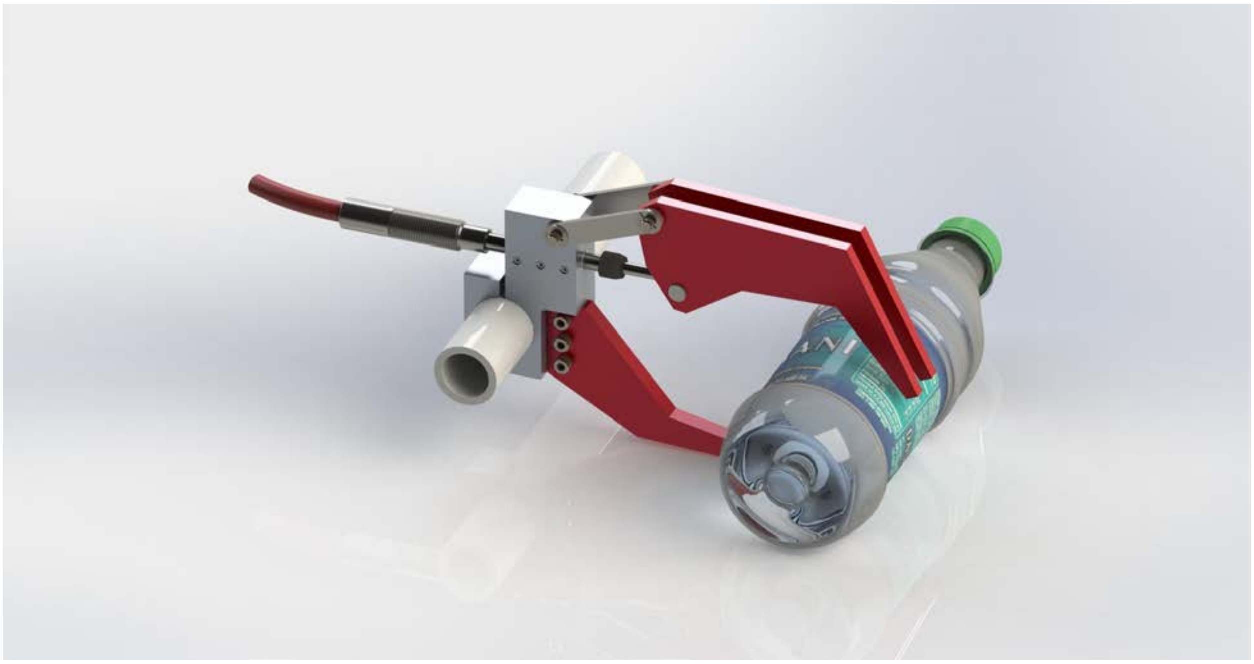


Figure 9: CAD picture of the functioning claw design grabbing a water bottle

Limitations to the jaw in early testing are caused by the stainless high-strength push-pull rod. The minimum bending radius of the thick cable is very large, and prevents easy mounting on board the ROV frame. Both cable bulkhead mounts are very robust however, so severe bending stress is unforeseeable.

6. Electronics Enclosure

Upon attaching to the ROV frame, the tether package is affixed in two (2) locations to avoid excessive strain. Primary wire inputs and outputs are routed through an eight (8) port pipe manifold, consisting of ⅜ and 2¼ NPT fittings. This pipe manifold is routed directly into the electronics enclosure through a 2¼ flexible line. The port usage is listed below:

1. DC ground
2. DC positive
3. Shielded Ethernet

4. Thruster 1
5. Thruster 2
6. Thruster 3
7. DC claw actuator
8. Air actuator cable
9. ⅜ NPT SPARE

Permanent connections are made through a 1 inch NPT potted PVC fitting. These connections include the three Raspberry PI cameras and the potted IMU chip. The potting compound utilized for sealing was two part Urethane. While using the pipe-manifold reduces the chances of leaking water coming into direct contact with critical electronics, the non-permanent IP67 wire glands are not guaranteed seals and leaks are common. Upon further testing the wire glands will be sealed from underneath with silicone for a semi-permanent solution.

The 6 inch acrylic tube was sealed with identical custom delrin end-caps, one modified to accept the wire inputs. To guarantee a waterproof seal two (2) O-rings were used on each end-cap. The electronic enclosure model can be seen below.

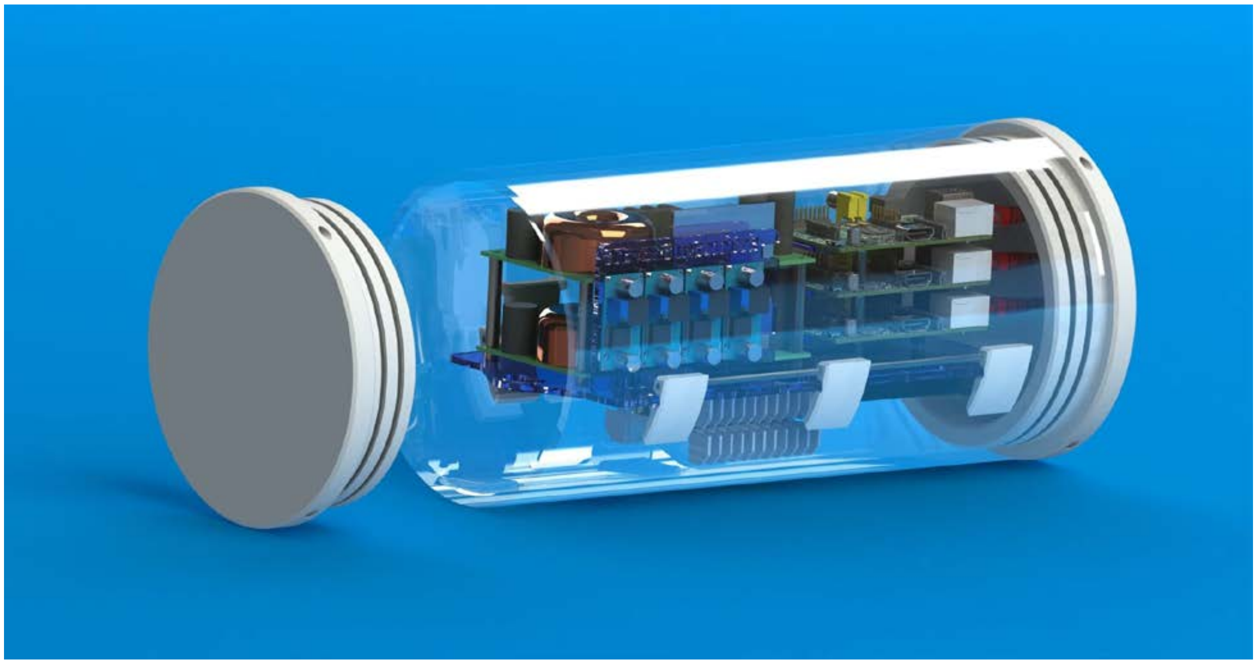
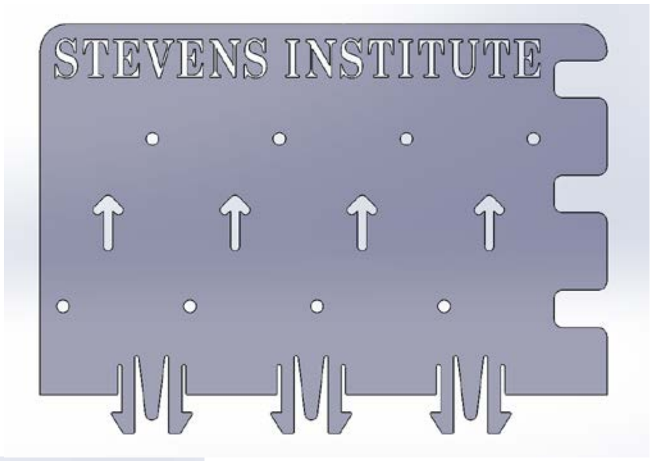


Figure 10: Picture of our electronics enclosure made from 6 inch acrylic tube with all the electronics neatly mounted inside the tube

Internal components were modeled in order to reach a high density pack arrangement. Extra space was allotted for the Beaglebone black for control cables, and the slack cable present upon closing the tube. Small details include custom length stand-offs, a tray-sliding mount, finger-pull hole in front, and the Stevens Institute name on the secondary voltage step-down insert. Additional holes and slots were included to zip-tie loose cables and add snap-fit inserts such as the secondary step-down tray. The electronics tray laser-cut profile with wire-management holes can be seen below.



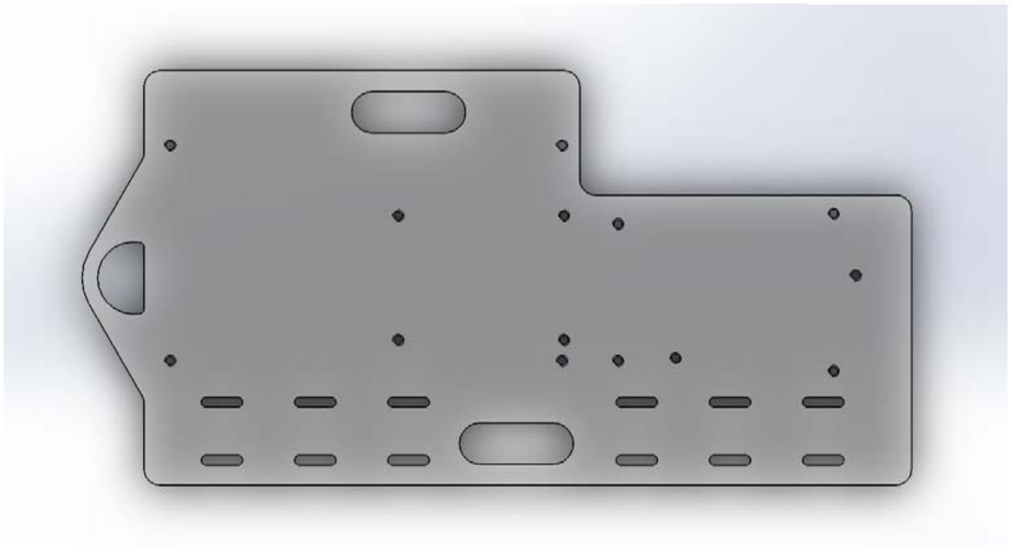


Figure 11: The electronics tray laser-cut profile with wire-management holes

7. Camera Enclosures

There are three (3) onboard color cameras onboard the ROV. These cameras cover the front view, left side view, and claw view. Standard Raspberry Pi cameras are utilized, sealed inside 1.25 inch PVC piping. In order to create a clear viewing window, ¼ inch acrylic sheet was laser cut to size and screwed to a modified PVC mounting flange with eight (8) ¼-20 bolts. To provide a wide range of modular adjustability, a singular silicone loop clamp was used with a custom aluminum camera tube adapter to allow for a wide range of motion. This allows tilting, sliding, pivoting, and turning of the lens on the chassis framework to adjust to nearly any desired viewing angle. The silicone cushioning prevents undesired movement, while adding dampening against shocks.

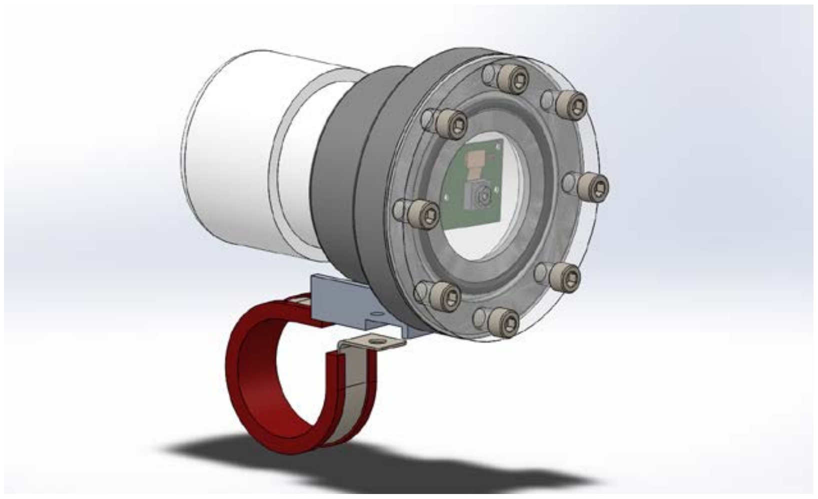


Figure 12: Camera enclosure with mount

To extend the camera data cables a ribbon cable kit was purchased. The camera lengths included two (2) at 30 inches and one (1) at 18 inches. To seal the assemblies, a slit was cut in the endcap of the camera tube to allow the data cable through. This was then sealed with epoxy on the exterior followed by a generous amount of silicone on the inside. With this configuration the cameras should have a considerable working depth, approximately 100 feet. Data control cables are taped down and organized together to prevent snagging and abrasion.

8. Camera Software

The camera system is composed of three Raspberry Pi single board computers with accompanying Raspberry Pi camera modules. Each Pi is running the RPI Cam Web Interface, which provides a web interface for the Raspberry Pi camera. The interface can be viewed in any web browser, and has a live video feed with low latency and high frame rate. Camera settings such as brightness and contrast can be controlled as well. Finally, the interface allows full-HD video or full-res pictures to be recorded on the Pi while the live feed continues. The interface loads automatically when the Pi starts

More information and source code for the RPi Cam Web Interface can be found at http://www.raspberrypi.org/forums/viewtopic.php?t=63276 and https://github.com/silvanmelchior/RPi_Cam_Web_Interface

9. Electronics

The electronics used to power and control the ROV and their connections are shown in the figure below. The main power from the tether is split to two main DC/DC converters to drop the supplied 48V down to 20V for the thrusters. These thrusters are controlled by 2 Sabertooth 2x12 motor controllers. The 4th channel on the motor controllers is used to control the linear actuator that moves the claw.

We used many small adjustable DC/DC converters to step down the 20V to the 5V needed by the Raspberry Pi's. One is also used to give 9V to the network switch, and one more provides 12V for the solenoid air valves. The Beaglebone controls everything over various interfaces. The motor controllers are connected over serial, the pressure sensor and IMU are connected over I^2C, and the air valves are connected via GPIO pins, with a custom made MOSFET to provide the needed power.

During testing, one issue we had was electronic noise from the motor controllers corrupting the sensor readings. To reduce this problem, we attached a ferrite bead around the sensor wire. This drastically reduced the problems we had reading the sensor.

A network switch connects the Beaglebone and the Raspberry Pi's to the surface.

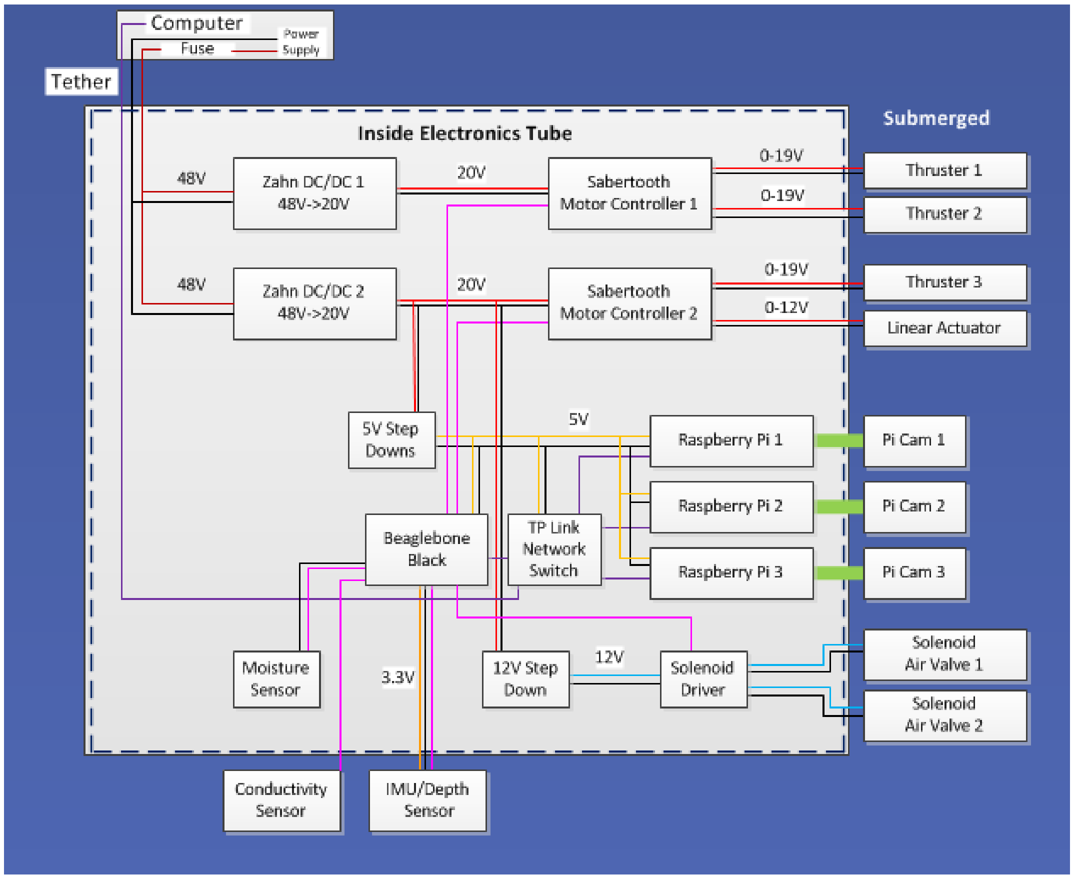
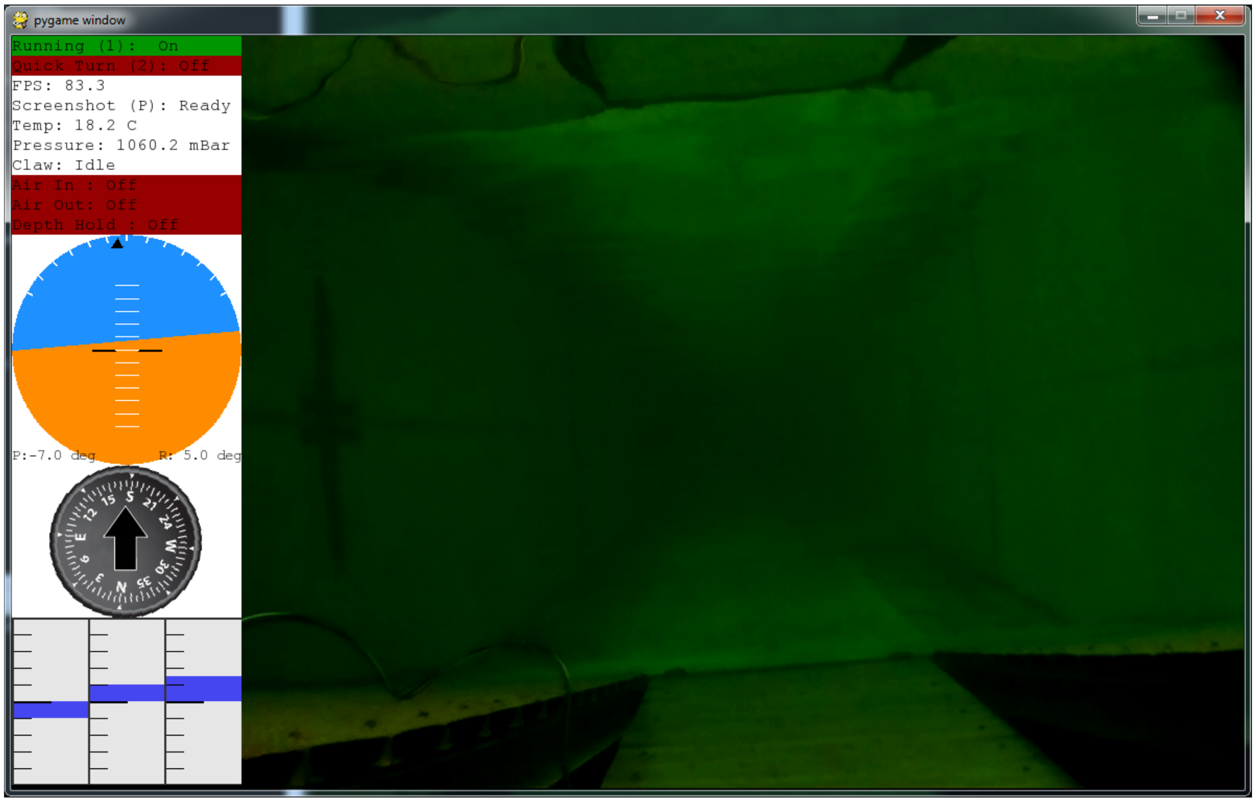


Figure 13: Electronics Layout

10. Top Side Software & Control

The ROV is controlled via a joystick connected to a laptop on the surface. A GUI displays input commands, sensor readings, and a high resolution real time video feed to the operator. The interface is written in Python using the Pygame library. The Pygame library allows access to the joystick as well as drawing to the screen. Python was chosen because it allows rapid development of the code and has a multitude of libraries that make multi threading and networking straightforward.

The GUI is shown in the figure below. The right side shows the high resolution video feed from the main camera to the operator. Additional video feeds showing the claw and side view are displayed separately on a secondary monitor. The bar on the left displays the status of the system and the commands being sent to the ROV. Pressure and temperature readings, the status of the air valves, and the current status of the claw are displayed in text format. Below that, an artificial horizon and compass display the orientation of the ROV based on the IMU sensor readings. On the bottom, the current speed of the thrusters is displayed.



```
Running (1):  On
Quick Turn (2): Off
FPS: 83.3
Screenshot (P): Ready
Temp: 18.2 C
Pressure: 1060.2 mBar
Claw: Idle
Air In : Off
Air Out: Off
Depth Hold : Off
P:-7.0 deg          R: 5.0 deg
```

Figure 14: Pictured above is the GUI. It shows the video feed, status of the systems and the commands being sent to the ROV

The application communicates to the ROV over an Ethernet cable which is part of the tether. Ethernet cable has a maximum recommended run length of 100m, which is well above the length of the tether. With a maximum data speed of 100 Mbps, a single cable has enough data capacity to handle the control data and video feeds.

Data communication is done via the UDP protocol. This was chosen over TCP because we do not need to resend any lost packets as it makes more sense to just send a new packet with updated data. Control packets are sent approximately 100 times per second, making the ROV extremely responsive. The commands are encoded in JSON format, a lightweight data-interchange format that is easy for humans to read and easy for computers to parse. It results in more data being sent, but allows the format of the commands to be changed or new ones to be added with ease. It is also language independent, so if the software on the ROV or surface was rewritten in a different programming language, the commands could easy be parsed and integrated into the new software.

The software has the functionality to save images from the video feed. This is required for the photomosaic stitching task. Since the stitching is required to be done during the timed mission run, an auxiliary laptop will likely be used to process the images into a photomosaic. A free plugin for the free GIMP image editing tool exists that will be used to create the photomosaic.

11. ROV Side Software

The Beaglebone Black (BBB) located on the ROV interfaces directly with all of the electronics via general purpose input output (GPIO) pins, like a microcontroller. The BBB receives control commands sent from the surface. It then uses those command to send appropriate commands to the motor controllers and air valves.

For safety, the system has a watchdog timer. If no control input is received from the surface computer for over a second, the ROV automatically enters a safe state with all of the thrusters off. Normal operation resumes when the control signal is again received.

Reading the IMU data and turning it into a useful representation is rather complex, luckily an open source library is available at https://github.com/mlaurijsse/linux-mpu9150/. Code for the pressure sensor was mostly written in house, with some code used from an Arduino library for the sensor. To access the sensors in Python, custom Python C extension modules were written to bridge the gap between Python and C.

For tasks such as performing a simulated sonar scan and picking up objects, the ROV will need to maintain its depth. To make this as easy as possible for the operator, we would like to use a feedback system that allows us to set a depth and will keep the ROV at that depth. Using feedback from a pressure sensor on the ROV, a PID control algorithm will adjust the vertical thruster speed to hold the ROV at a given depth.

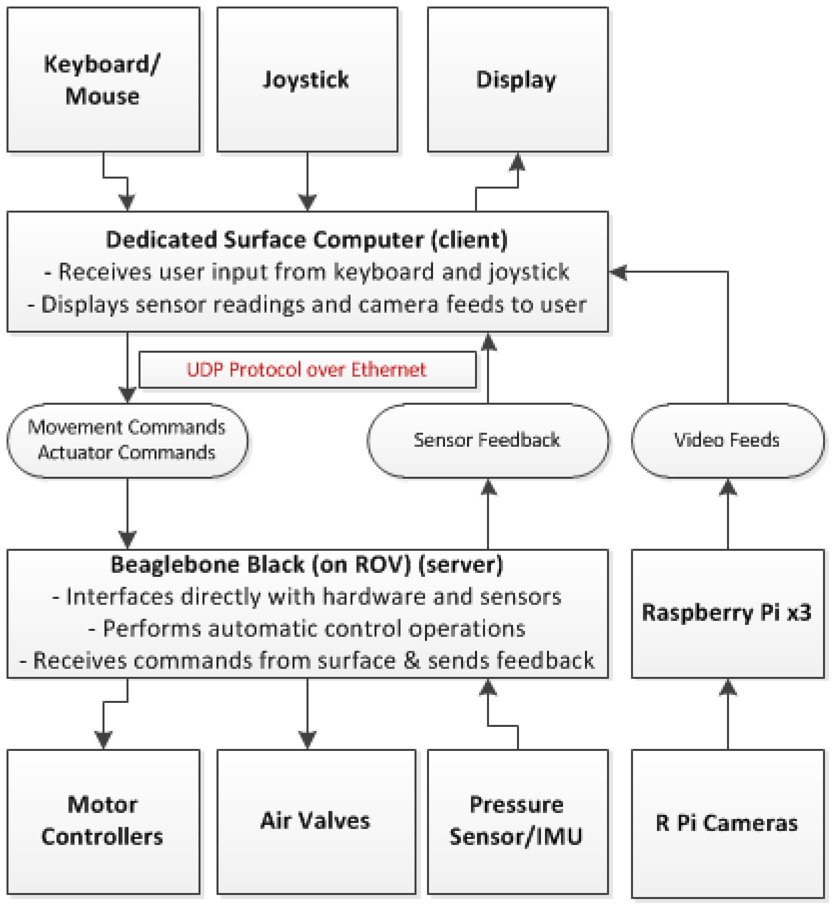

Keyboard/ Mouse
Joystick
Display
Dedicated Surface Computer (client)
- Receives user input from keyboard and joystick
- Displays sensor readings and camera feeds to user
UDP Protocol over Ethernet
Movement Commands
Actuator Commands
Sensor Feedback
Video Feeds
Beaglebone Black (on ROV) (server)
- Interfaces directly with hardware and sensors
- Performs automatic control operations
- Receives commands from surface & sends feedback
Raspberry Pi x3
Motor Controllers
Air Valves
Pressure Sensor/IMU
R Pi Cameras

Figure 15: Shown above is the Hardware and Software Flowchart

Testing

Test #1:

Our first test took place in the Davidson Tank. Our goal was to test the water proofing concept of our ROV. The total depth of the Davidson Tank is 7 feet, which is a big difference from the total depth of our competition. Our testing method was to put several components of the ROV under the water for various increments to constantly check for any water leakage. The first component we tested was our ROV frame. We added weight to the frame to make sure it would sit at the bottom of the tank floor. To properly test it we added weight on the frame to make sure it would sit on the tank floor for the duration of the test. We placed in it for 10 minutes, then 20 minutes and finally 30 minutes. The total length of the competition will be a maximum of 20 minutes. The below figure is a picture of the frame under water. At the end of our test, there was no water in the frame.

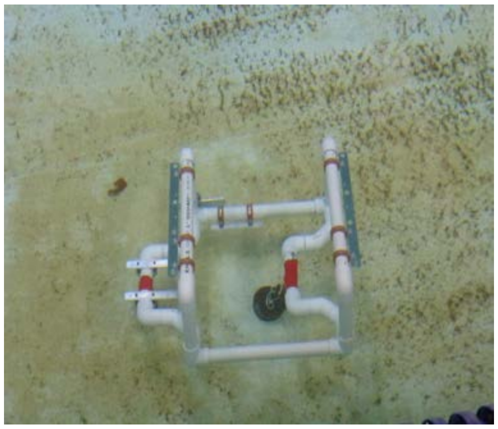


Figure 16: Picture of our ROV frame during water proof testing on the floor of the Davidson Laboratory Tank

We also then tested the water proof capability of our electronic enclosure, pictured below. We placed pieces of paper inside the container to make sure they remained dry during the test. We added weight to this to make sure it sat at on the tank floor. We tested it in various increments of 10 minutes, 20 minutes and 30 minutes. At the end of our test, no water was inside the container and the paper remained dry.

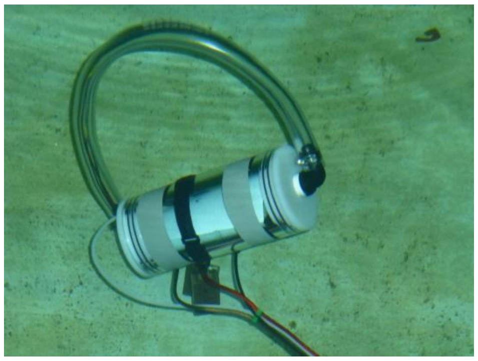


Figure 17: Picture of the electronics enclosure being water proof tested

Lastly we tested our thrusters to make sure they were functioning. In the end we had 3 working thrusters for our ROV and 1 spare working thruster.

Figure 18: Testing the thrusters to make sure they work

Test #2:

Test #2 took place in Davidson Tanks. The objective was to test the functionality of our ROV. We were focusing on buoyancy and center of gravity and how we could manipulate that by adding, distributing or removing weights onto the ROV. To fully stabilize our ROV we needed to add weight to the left back corner of the ROV. To assist the ROV when sinking or rising in the water, the ballast tanks were filled or depleted. This test proved that the ROV was able to function in all needed directions.

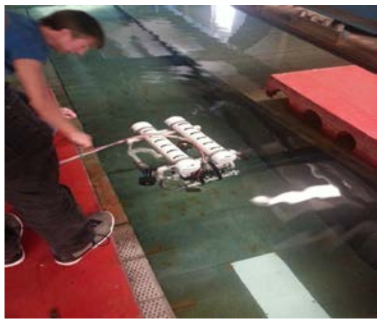


Figure 19: Our ROV functioning in the water

Test #3:

Test #3 took place in Davidson Labs, to test the full functioning capability of the ROV with the claw and gripper mechanism. The ROV was able to perform and successfully picked up a water bottle on the bottom of the tank. During the competition our ROV will be performing tasks and the maximum weight our gripper will have to be able to hold is 100N of force which is about 22 lbs. This test also demonstrated the importance of camera placement. Our ROV has 3 cameras strategically placed on the ROV so that we can see any view we need to. This test demonstrated that our depth perception through our camera were accurate when we were successfully able to pick up the water bottle.

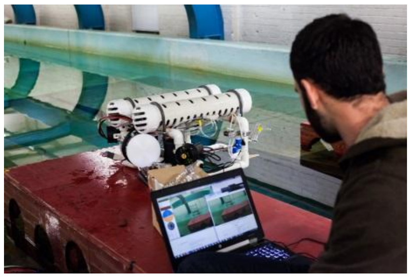


Figure 20: Set up before testing

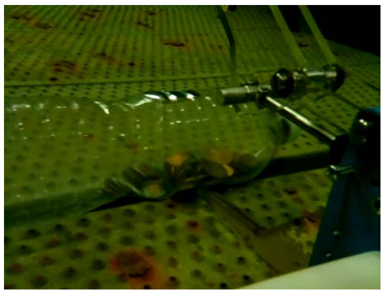


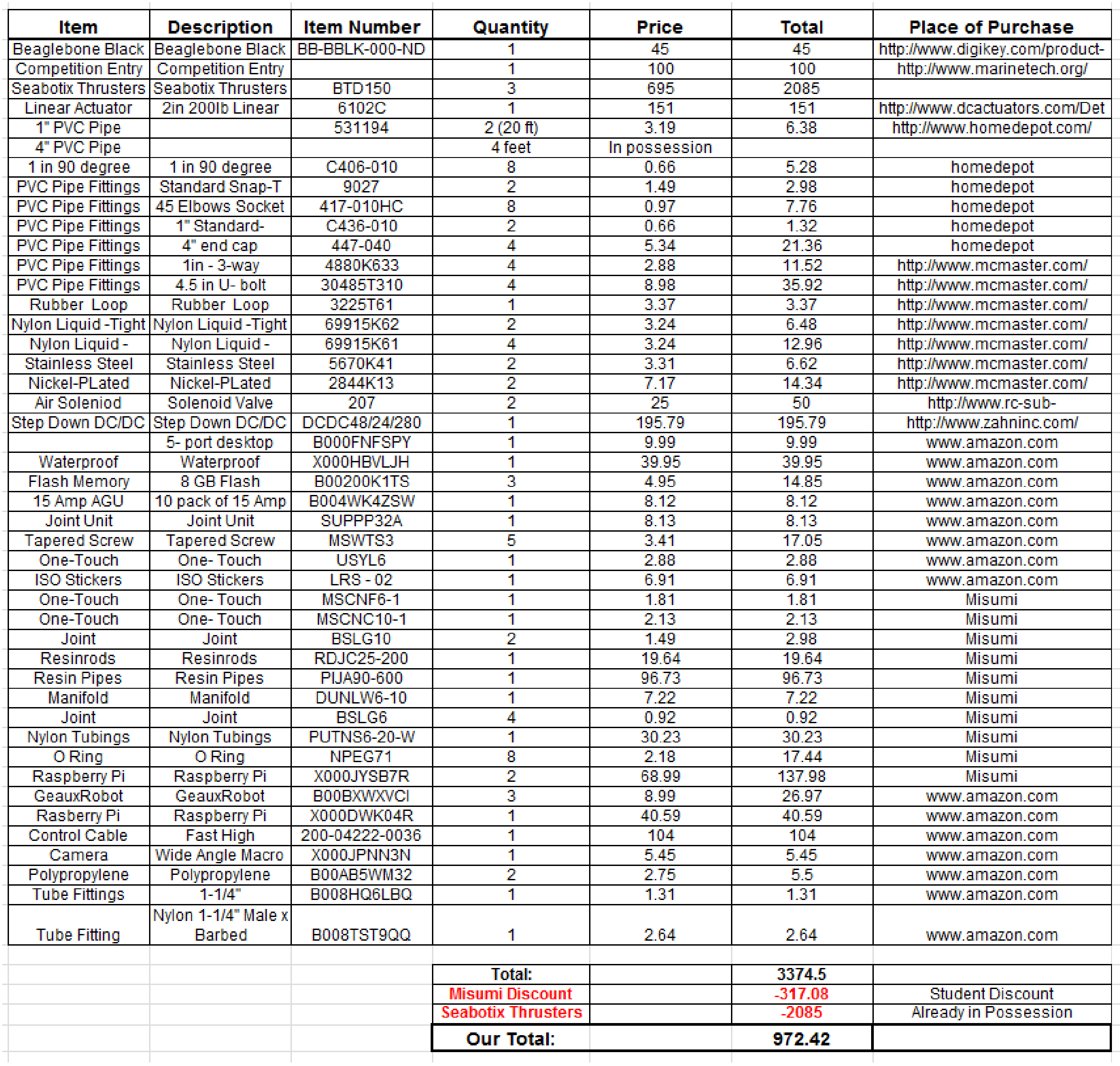
Figure 21: ROV grabbing a water bottle on the tank floor

Learning Experiences:

To be successful on any major project, especially design projects, team work is definitely the most important aspect. Every member of the team plays a special role and it requires hard work and dedication. Communication amongst group members should be done often to make sure everyone is on the same page. To help remain on task throughout the duration of the project, the Gantt Chart is very beneficial and is a valuable asset that is under used in industry. If given this opportunity again, it would be in the student's best interest to seek advice from multiple people, not just from each other and the advisor. Our group started to reach out to various departments and students but would recommend other groups do it before designing the ROV so that their input can have a stronger effect on the outcome of the project. Testing is also very important. Testing is where the group has hands on experience to correct and improve the project. Our group had the opportunity to test different components individually and also as one unit. We were successful and satisfied with our testing procedures but you can never have enough testing. Testing, especially for an ROV gives students the opportunity to see what could go wrong during a competition and how to be able to quickly adjust and fix the ROV so that in the competition environment students will remain calm to fix the problem and finish the competition.

Appendix:

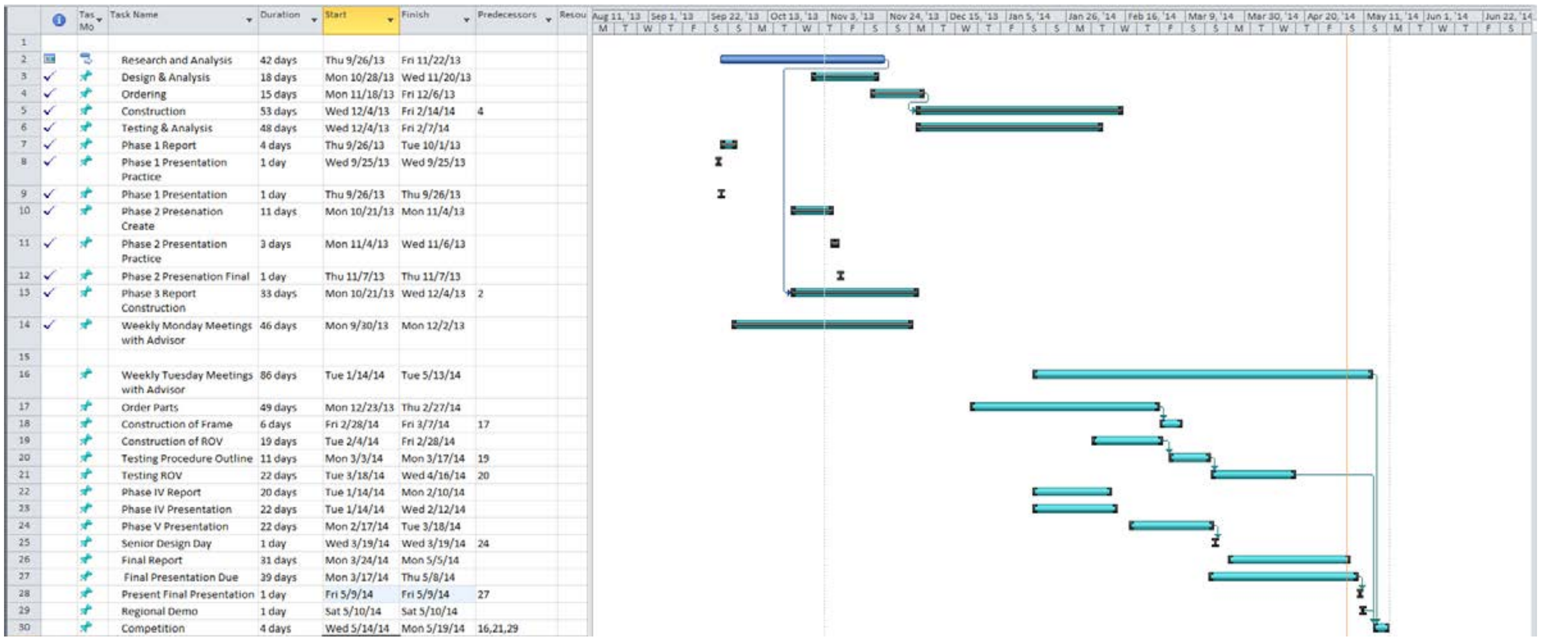
Final Budget for our Project:



Item	Description	Item Number	Quantity	Price	Total	Place of Purchase
Beaglebone Black	Beaglebone Black	BB-BBLK-000-ND	1	45	45	http://www.digikey.com/product-
Competition Entry	Competition Entry		1	100	100	http://www.marinetech.org/
Seabotix Thrusters	Seabotix Thrusters	BTD150	3	695	2085	
Linear Actuator	2in 200lb Linear	6102C	1	151	151	http://www.dcactuators.com/Det
1" PVC Pipe		531194	2 (20 ft)	3.19	6.38	http://www.homedepot.com/
4" PVC Pipe			4 feet	In possession		
1 in 90 degree	1 in 90 degree	C406-010	8	0.66	5.28	homedepot
PVC Pipe Fittings	Standard Snap-T	9027	2	1.49	2.98	homedepot
PVC Pipe Fittings	45 Elbows Socket	417-010HC	8	0.97	7.76	homedepot
PVC Pipe Fittings	1" Standard-	C436-010	2	0.66	1.32	homedepot
PVC Pipe Fittings	4" end cap	447-040	4	5.34	21.36	homedepot
PVC Pipe Fittings	1in - 3-way	4880K633	4	2.88	11.52	http://www.mcmaster.com/
PVC Pipe Fittings	4.5 in U- bolt	30485T310	4	8.98	35.92	http://www.mcmaster.com/
Rubber Loop	Rubber Loop	3225T61	1	3.37	3.37	http://www.mcmaster.com/
Nylon Liquid -Tight	Nylon Liquid -Tight	69915K62	2	3.24	6.48	http://www.mcmaster.com/
Nylon Liquid -	Nylon Liquid -	69915K61	4	3.24	12.96	http://www.mcmaster.com/
Stainless Steel	Stainless Steel	5670K41	2	3.31	6.62	http://www.mcmaster.com/
Nickel-PLated	Nickel-PLated	2844K13	2	7.17	14.34	http://www.mcmaster.com/
Air Soleniod	Solenoid Valve	207	2	25	50	http://www.rc-sub-
Step Down DC/DC	Step Down DC/DC	DCDC48/24/280	1	195.79	195.79	http://www.zahninc.com/
	5- port desktop	B000FNFSPY	1	9.99	9.99	www.amazon.com
Waterproof	Waterproof	X000HBVLJH	1	39.95	39.95	www.amazon.com
Flash Memory	8 GB Flash	B00200K1TS	3	4.95	14.85	www.amazon.com
15 Amp AGU	10 pack of 15 Amp	B004WK4ZSW	1	8.12	8.12	www.amazon.com
Joint Unit	Joint Unit	SUPPP32A	1	8.13	8.13	www.amazon.com
Tapered Screw	Tapered Screw	MSWTS3	5	3.41	17.05	www.amazon.com
One-Touch	One- Touch	USYL6	1	2.88	2.88	www.amazon.com
ISO Stickers	ISO Stickers	LRS - 02	1	6.91	6.91	www.amazon.com
One-Touch	One- Touch	MSCNF6-1	1	1.81	1.81	Misumi
One-Touch	One- Touch	MSCNC10-1	1	2.13	2.13	Misumi
Joint	Joint	BSLG10	2	1.49	2.98	Misumi
Resinrods	Resinrods	RDJC25-200	1	19.64	19.64	Misumi
Resin Pipes	Resin Pipes	PIJA90-600	1	96.73	96.73	Misumi
Manifold	Manifold	DUNLW6-10	1	7.22	7.22	Misumi
Joint	Joint	BSLG6	4	0.92	0.92	Misumi
Nylon Tubings	Nylon Tubings	PUTNS6-20-W	1	30.23	30.23	Misumi
O Ring	O Ring	NPEG71	8	2.18	17.44	Misumi
Raspberry Pi	Raspberry Pi	X000JYSB7R	2	68.99	137.98	Misumi
GeauxRobot	GeauxRobot	B00BXWXVCI	3	8.99	26.97	www.amazon.com
Rasberry Pi	Raspberry Pi	X000DWK04R	1	40.59	40.59	www.amazon.com
Control Cable	Fast High	200-04222-0036	1	104	104	www.amazon.com
Camera	Wide Angle Macro	X000JPNN3N	1	5.45	5.45	www.amazon.com
Polypropylene	Polypropylene	B00AB5WM32	2	2.75	5.5	www.amazon.com
Tube Fittings	1-1/4"	B008HQ6LBQ	1	1.31	1.31	www.amazon.com
Tube Fitting	Nylon 1-1/4" Male x Barbed	B008TST9QQ	1	2.64	2.64	www.amazon.com
			Total:		**3374.5**	
			Misumi Discount		-317.08	Student Discount
			Seabotix Thrusters		-2085	Already in Possession
			Our Total:		**972.42**	

Appendix 1: Our final Budget for our Project

The figure above shows the total budget for our project. Buying every part new and at full value our project would have been about $3,374.50. Thankfully we were able to get a student credit discount from Misumi and we already had our Seabotix Thruster from previous years. With this help our budget only equaled $972.42. This project is unique because we are able to build a whole functioning underwater ROV for a fairly cheap cost considering other projects and their functions and costs.



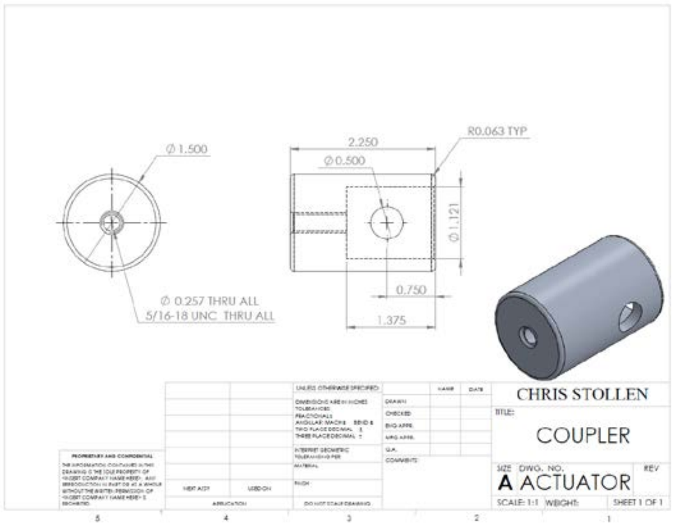
		Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resou
1								
2			Research and Analysis	42 days	Thu 9/26/13	Fri 11/22/13		
3	✓		Design & Analysis	18 days	Mon 10/28/13	Wed 11/20/13		
4	✓		Ordering	15 days	Mon 11/18/13	Fri 12/6/13		
5	✓		Construction	53 days	Wed 12/4/13	Fri 2/14/14	4	
6	✓		Testing & Analysis	48 days	Wed 12/4/13	Fri 2/7/14		
7	✓		Phase 1 Report	4 days	Thu 9/26/13	Tue 10/1/13		
8	✓		Phase 1 Presentation Practice	1 day	Wed 9/25/13	Wed 9/25/13		
9	✓		Phase 1 Presentation	1 day	Thu 9/26/13	Thu 9/26/13		
10	✓		Phase 2 Presenation Create	11 days	Mon 10/21/13	Mon 11/4/13		
11	✓		Phase 2 Presentation Practice	3 days	Mon 11/4/13	Wed 11/6/13		
12	✓		Phase 2 Presenation Final	1 day	Thu 11/7/13	Thu 11/7/13		
13	✓		Phase 3 Report Construction	33 days	Mon 10/21/13	Wed 12/4/13	2	
14	✓		Weekly Monday Meetings with Advisor	46 days	Mon 9/30/13	Mon 12/2/13		
15								
16			Weekly Tuesday Meetings with Advisor	86 days	Tue 1/14/14	Tue 5/13/14		
17			Order Parts	49 days	Mon 12/23/13	Thu 2/27/14		
18			Construction of Frame	6 days	Fri 2/28/14	Fri 3/7/14	17	
19			Construction of ROV	19 days	Tue 2/4/14	Fri 2/28/14		
20			Testing Procedure Outline	11 days	Mon 3/3/14	Mon 3/17/14	19	
21			Testing ROV	22 days	Tue 3/18/14	Wed 4/16/14	20	
22			Phase IV Report	20 days	Tue 1/14/14	Mon 2/10/14		
23			Phase IV Presentation	22 days	Tue 1/14/14	Wed 2/12/14		
24			Phase V Presentation	22 days	Mon 2/17/14	Tue 3/18/14		
25			Senior Design Day	1 day	Wed 3/19/14	Wed 3/19/14	24	
26			Final Report	31 days	Mon 3/24/14	Mon 5/5/14		
27			Final Presentation Due	39 days	Mon 3/17/14	Thu 5/8/14		
28			Present Final Presentation	1 day	Fri 5/9/14	Fri 5/9/14	27	
29			Regional Demo	1 day	Sat 5/10/14	Sat 5/10/14		
30			Competition	4 days	Wed 5/14/14	Mon 5/19/14	16,21,29	

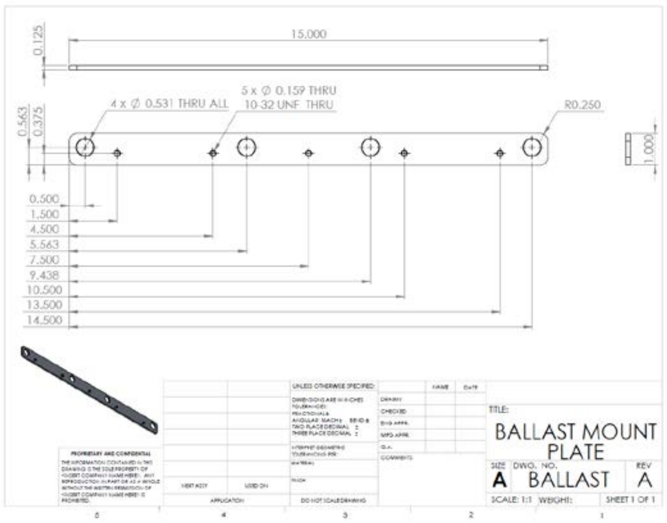
Appendix 2: Final Gantt Chart of our Project over the course of 2 semesters, condensed for easy visibility

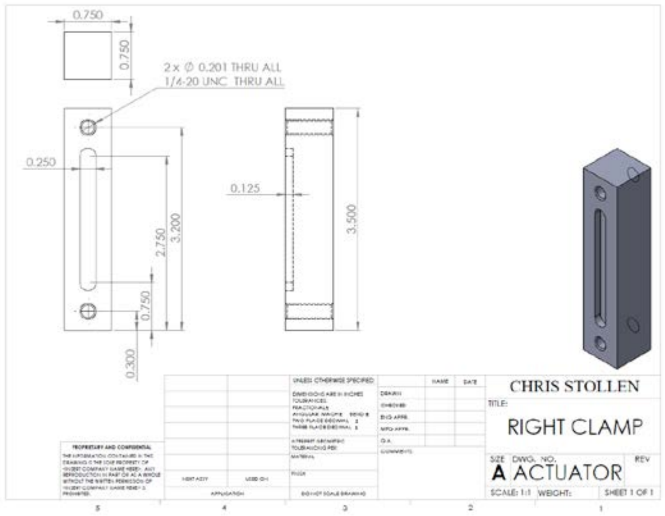
Gantt Chart:

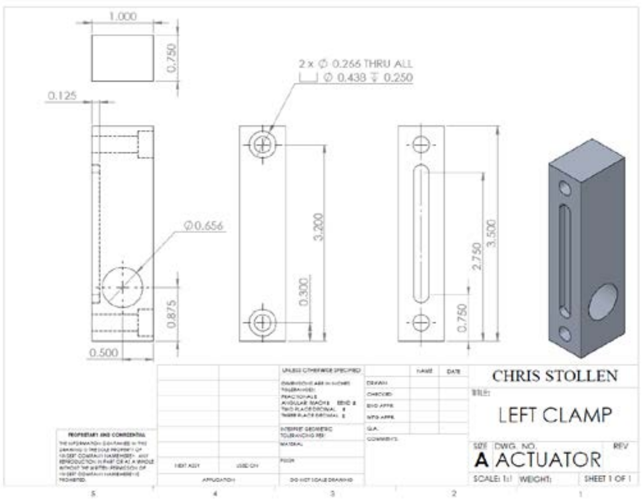
Pictured above is our team's Gantt Chart. Throughout the course of the semester tasks were modified to keep with the needs of our project. This work break down scheduled help to keep our group focused and organized on completing each phase of our project and keep a good pace to make sure we allowed for adequate testing of our project in water to make sure it was ready for competition.

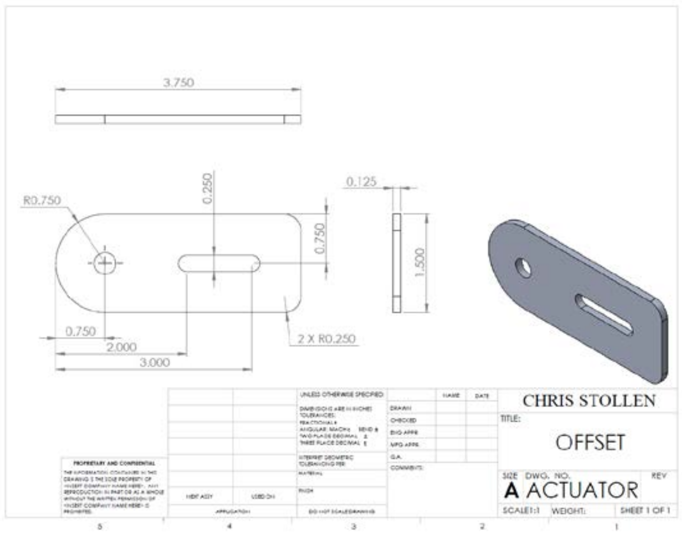
Drawings of Components of the ROV:

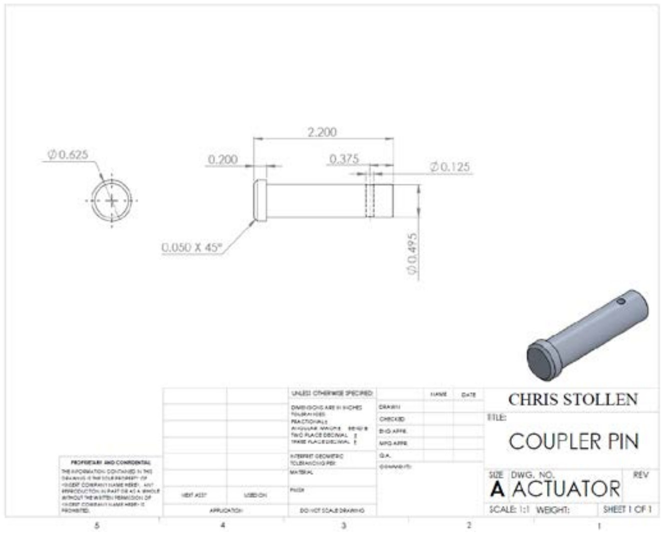


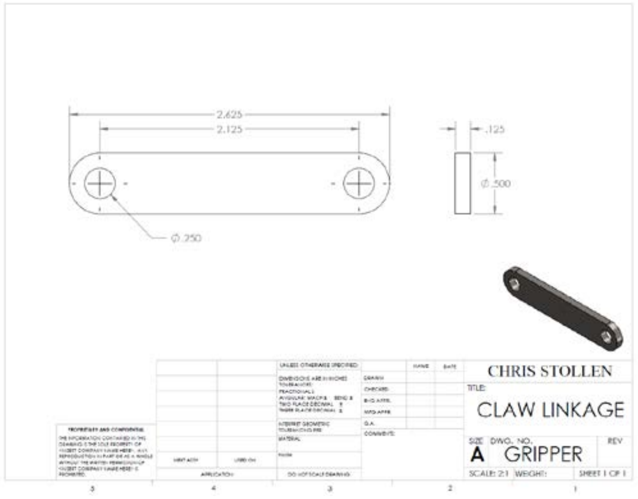


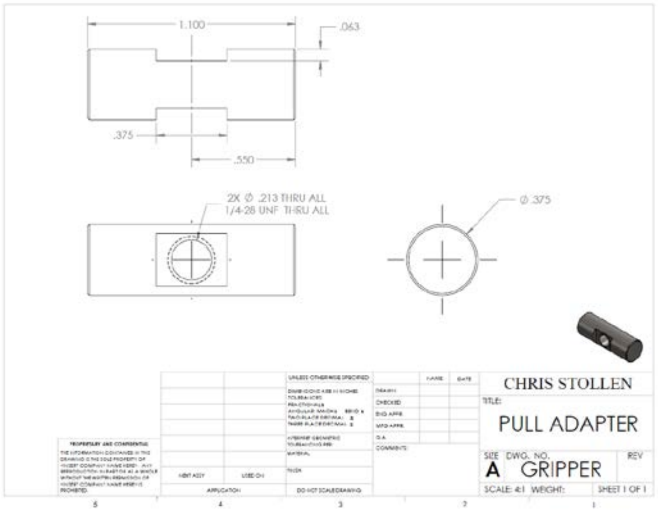


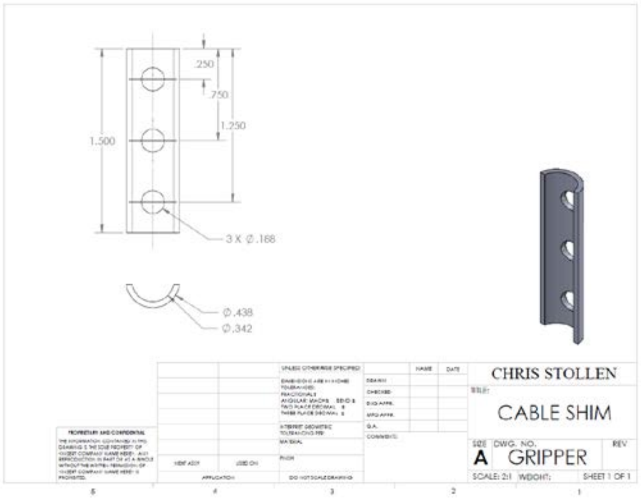


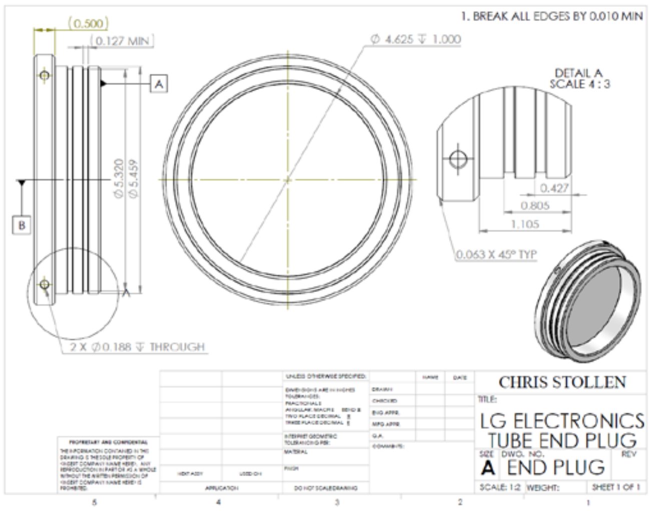


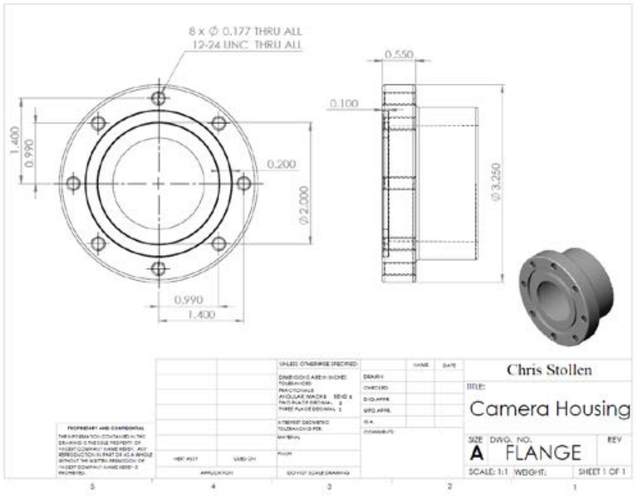


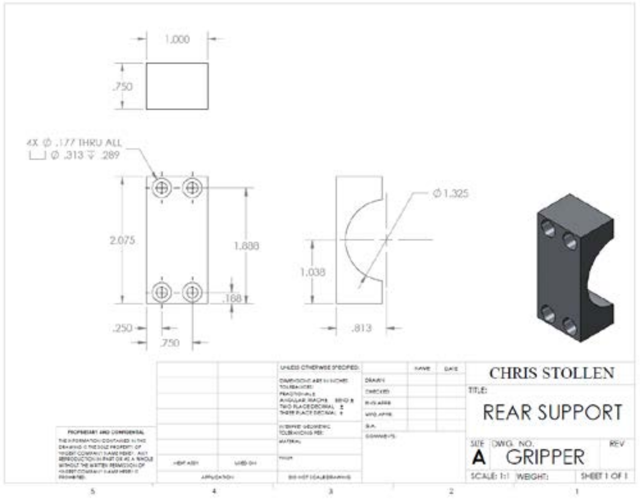

2.625
2.125
.125
Ø.500
Ø.250
CHRIS STOLLEN
CLAW LINKAGE
A GRIPPER
SCALE: 2:1

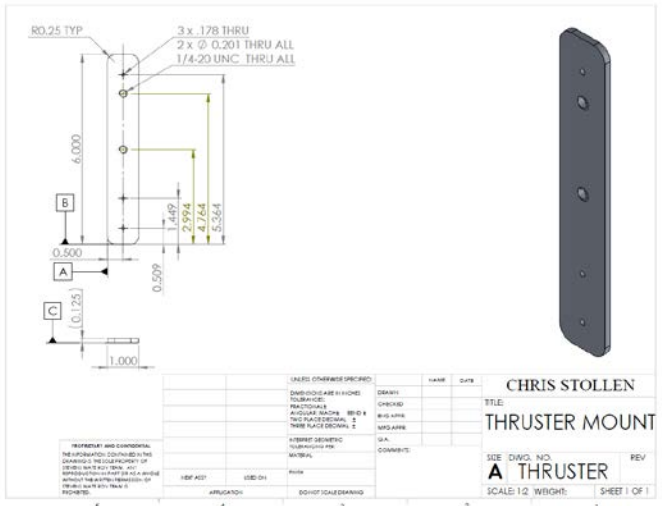

1.100
.063
.375
.550
2X Ø .213 THRU ALL
1/4-28 UNF THRU ALL
Ø.375
CHRIS STOLLEN
PULL ADAPTER
A GRIPPER
SCALE: 4:1

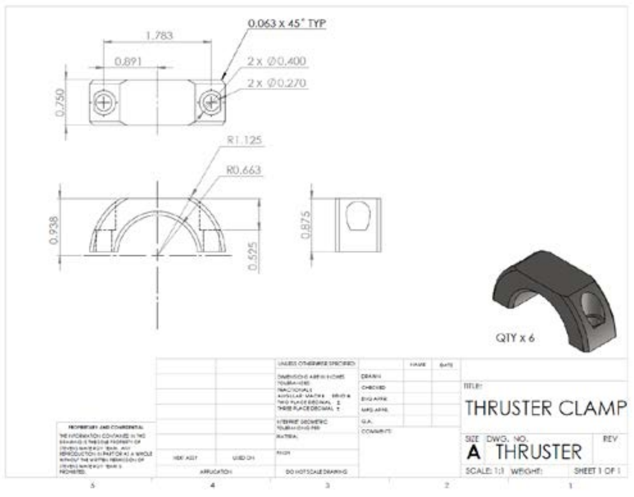

.250
.750
1.250
1.500
3 X Ø.188
Ø.438
Ø.342
CHRIS STOLLEN
CABLE SHIM
A GRIPPER
SCALE: 2:1

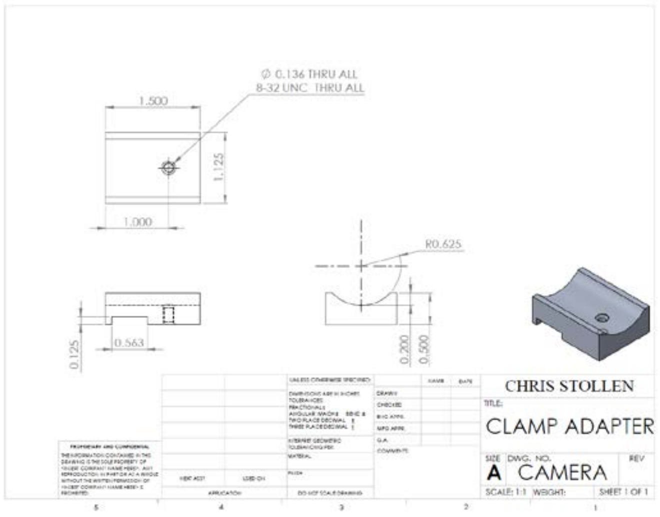

1. BREAK ALL EDGES BY 0.010 MIN
(0.500)
0.127 MIN
Ø 4.625 ↧ 1.000
A
B
Ø5.320
Ø5.459
DETAIL A
SCALE 4 : 3
0.427
0.805
1.105
0.063 X 45° TYP
2 X Ø0.188 ↧ THROUGH
CHRIS STOLLEN
LG ELECTRONICS TUBE END PLUG
A END PLUG
SCALE: 1:2

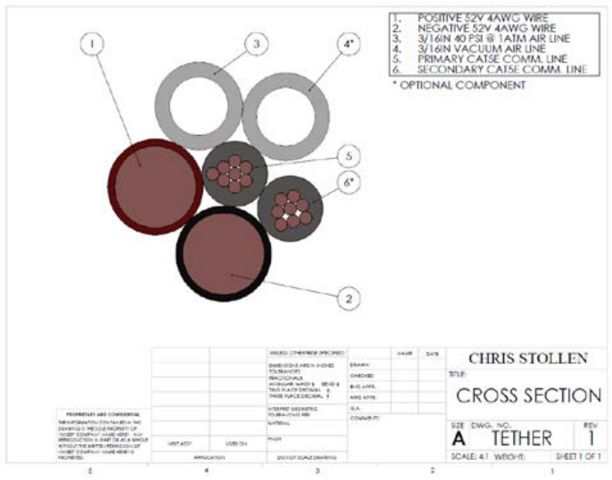

8 x Ø 0.177 THRU ALL
12-24 UNC THRU ALL
0.550
0.100
1.400
0.990
0.200
Ø2.000
Ø3.250
0.990
1.400
Chris Stollen
Camera Housing
A FLANGE
SCALE: 1:1

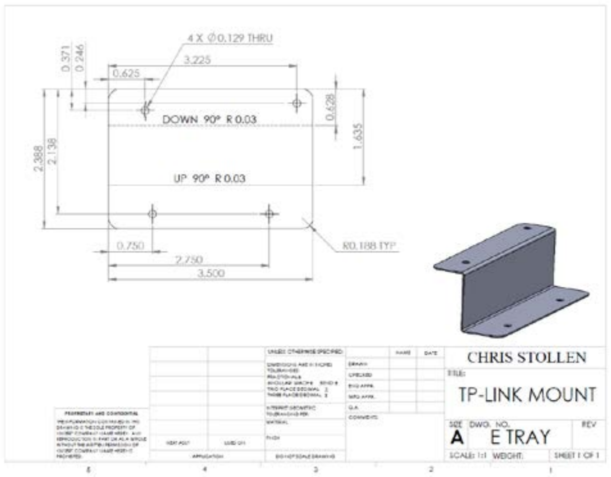

1.000
.750
4X Ø .177 THRU ALL
⌴ Ø .313 ↧ .289
Ø1.325
2.075
1.888
1.038
.188
.250
.750
.813
CHRIS STOLLEN
REAR SUPPORT
A GRIPPER
SCALE: 1:1

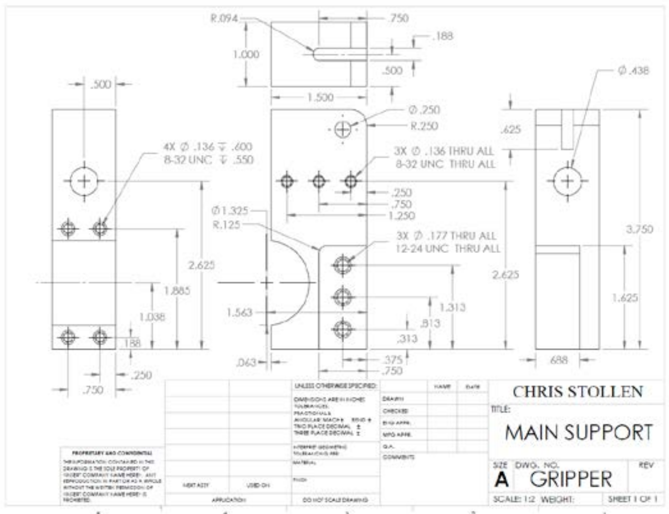

R0.25 TYP
3 x .178 THRU
2 x Ø 0.201 THRU ALL
1/4-20 UNC THRU ALL
6.000
1.449
2.994
4.764
5.364
0.500
0.509
0.125
1.000
B
A
C
CHRIS STOLLEN
TITLE:
THRUSTER MOUNT
SIZE DWG. NO.
A THRUSTER
REV
SCALE: 1:2 WEIGHT:
SHEET 1 OF 1

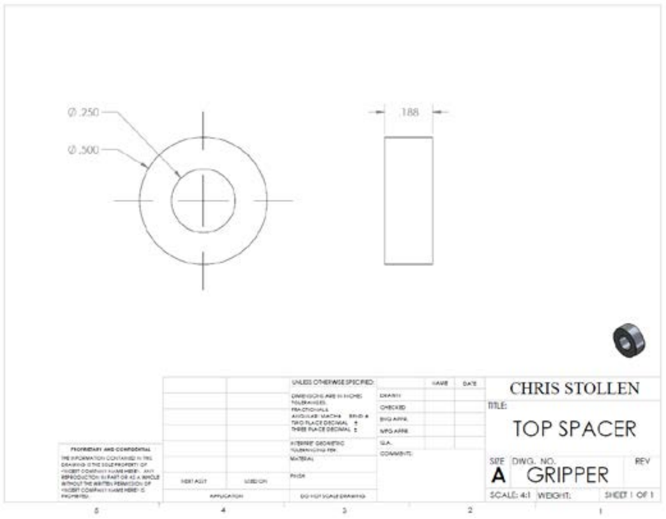

0.063 x 45° TYP
1.783
0.891
2 x Ø0.400
2 x Ø0.270
0.750
R1.125
R0.663
0.938
0.875
0.525
QTY x 6
TITLE:
THRUSTER CLAMP
SIZE DWG. NO.
A THRUSTER
REV
SCALE: 1:1 WEIGHT:
SHEET 1 OF 1


Ø 0.136 THRU ALL
8-32 UNC THRU ALL
1.500
1.125
1.000
R0.625
0.125
0.563
0.200
0.500
CHRIS STOLLEN
TITLE:
CLAMP ADAPTER
SIZE DWG. NO.
A CAMERA
REV
SCALE: 1:1 WEIGHT:
SHEET 1 OF 1


1. POSITIVE 52V 4AWG WIRE
2. NEGATIVE 52V 4AWG WIRE
3. 3/16IN 40 PSI @ 1ATM AIR LINE
4. 3/16IN VACUUM AIR LINE
5. PRIMARY CAT5E COMM. LINE
6. SECONDARY CAT5E COMM. LINE
* OPTIONAL COMPONENT
1
2
3
4*
5
6*
CHRIS STOLLEN
TITLE:
CROSS SECTION
SIZE DWG. NO.
A TETHER
REV
1
SCALE: 4:1 WEIGHT:
SHEET 1 OF 1

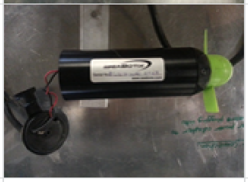

4 X Ø0.129 THRU
3.225
0.625
0.371
0.246
DOWN 90° R 0.03
UP 90° R 0.03
0.628
1.635
2.388
2.138
0.750
2.750
3.500
R0.188 TYP
CHRIS STOLLEN
TITLE:
TP-LINK MOUNT
SIZE DWG. NO.
A E TRAY
REV
SCALE: 1:1 WEIGHT:
SHEET 1 OF 1


R.094
.750
.188
1.000
.500
1.500
Ø.438
Ø.250
R.250
.625
4X Ø .136 ⌵ .600
8-32 UNC ⌵ .550
3X Ø .136 THRU ALL
8-32 UNC THRU ALL
.250
.750
1.250
Ø1.325
R.125
3X Ø .177 THRU ALL
12-24 UNC THRU ALL
3.750
2.625
1.885
1.038
1.563
1.313
.813
.313
.188
1.625
.063
.375
.688
.250
.750
CHRIS STOLLEN
TITLE:
MAIN SUPPORT
SIZE DWG. NO.
A GRIPPER
REV
SCALE: 1:2 WEIGHT:
SHEET 1 OF 1



Thruster Test:

Thruster Status

No.	Src	image	Serial No.	Status	Notes	TEST
1	ASV		BTD-150-0412-00	WORKING	Lubricated on 4/10/14	
2	ASV		BTD-150-0412-00	WORKING	Propeller is lacking the proper lubrication, shaft is fouled near external bushing. Lubricated on 4/10/14	
3	Bluedart		BTD-150-0310-00	WORKING	Lubricated on 4/10/14	
4	Bluedart		BTD-150-1208004	WORKING	Kort nozzle removed; modified nozzle mount. Lubricated on 4/10/14	
5	Bluedart		BTD-150-01-09-0	BROKEN	Kort nozzle removed; modified nozzle mount; 3" long pigtail	
6	Bluedart		BTD-150-0310-00	BROKEN	Kort nozzle removed; modified nozzle mount; missing retention strap; missing propeller pin; possible case flooding	

Code:

The entire project code can be seen over the next few pages

Appendix – Code Listings

Listing 1: surface.py

```
import sys
import socket
import json
import datetime
import time
import os
import urllib2
import cStringIO
import threading
import thread
import pygame
import math
from pygame.rect import Rect

import receivedata
import widgets


UDP_IP = "192.168.1.10"
UDP_PORT = 1870


#window initilization
sidebarwidth = 220
pygame.init()
pygame.display.set_caption('ROV Control')
size = width, height = 960+sidebarwidth, 720


##Set up widgets
screen = pygame.display.set_mode(size)
onstatus = widgets.toggleable("Running (1)", sidebarwidth)
turningdisplay = widgets.toggleable("Quick Turn (2)", sidebarwidth)
fpsdisplay = widgets.display("FPS", sidebarwidth)
tempdisplay = widgets.display("Temp", sidebarwidth)
presdisplay = widgets.display("Pressure", sidebarwidth)
screenshot = widgets.display("Screenshot (P)", sidebarwidth)

clawdisplay = widgets.display("Claw", sidebarwidth)
clawdisplay.value = "Idle"
airoutdisplay = widgets.toggleable("Air Out", sidebarwidth)
airindisplay = widgets.toggleable("Air In ", sidebarwidth)
depthholddisplay = widgets.toggleable("Depth Hold ", sidebarwidth)

ahdisplay = widgets.ahorizon(sidebarwidth)
compassdisplay = widgets.compass(sidebarwidth*2/3)

zslider = widgets.sliderdisplay("zslider", 75, 160)
mleftslider = widgets.sliderdisplay("Leftslider", 75, 160)
mrightslider = widgets.sliderdisplay("Rightslider", 75, 160)

image = widgets.ipimage("http://192.168.1.11/cam_pic.php")


#init joystick
joystick = None
if pygame.joystick.get_count() == 0:
```

1

```
    print "No Joysticks Detected"
else:
    joystick = pygame.joystick.Joystick(0)
    joystick.init()

#Start ROV data getter
rec = receivedata.receivedata('Surface', '', UDP_PORT)
rec.start()


##Main Loop
currenttime = pygame.time.get_ticks() - 1
running = True
while running:
    prevtime = currenttime
    currenttime = pygame.time.get_ticks()

    ## Get input from joystick and keyboarrd, update widgets
    pygame.event.pump()
    key = pygame.key.get_pressed()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_p:
                image.screenshot()
        if event.type == pygame.JOYBUTTONDOWN:
            if event.button == 0:
                onstatus.toggle()
            if event.button == 1:
                turningdisplay.enable()
            if event.button == 2:
                clawdisplay.setValue("Closing")
                clawdisplay.bgcolor = (200, 200, 0)
            if event.button == 3:
                clawdisplay.setValue("Opening")
                clawdisplay.bgcolor = (0, 200, 0)
            if event.button == 4:
                airindisplay.toggle()
            if event.button == 5:
                airoutdisplay.toggle()
        if event.type == pygame.JOYBUTTONUP:
            if event.button == 1:
                turningdisplay.disable()
            if event.button == 2:
                clawdisplay.setValue("Idle")
                clawdisplay.bgcolor = (255, 255, 255)
            if event.button == 3:
                clawdisplay.setValue("Idle")
                clawdisplay.bgcolor = (255, 255, 255)

    fpsdisplay.setValue('{:3.1f}'.format((1000.0 / (currenttime - prevtime))))

    if image.can_screenshot:
        screenshot.setValue("Ready")
    else:
        screenshot.setValue("Taken")

    #Create commands to send to ROV
    commands = {}

    if joystick is not None:

        y = joystick.get_axis(1)
        x = joystick.get_axis(0)
        twist = joystick.get_axis(3)

        #only run motors of system is on - don't burn out those thrusters!
        if onstatus.state:
            if turningdisplay.state:
                #turning mode
                commands['tleft'] = twist
                commands['tright'] = -twist
            else:
                #normal mode
                ratio = abs(x)
                power = -y
                if math.copysign(1, x) > 0:
                    commands['tleft'] = power
                    commands['tright'] = power * (1 - ratio)
                else:
                    commands['tright'] = power
                    commands['tleft'] = power * (1 - ratio)

            commands['tup'] = -joystick.get_axis(2)

            if clawdisplay.value == "Opening":
                commands['claw'] = 0.65
            elif clawdisplay.value == "Closing":
                commands['claw'] = -0.65
            elif clawdisplay.value == "Idle":
                commands['claw'] = 0

            commands["airout"] = airoutdisplay.state
            commands["airin"] = airindisplay.state

        else:
            commands['tleft'] = 0
            commands['tright'] = 0
            commands['tup'] = 0
            commands['claw'] = 0
            commands['airout'] = False
            commands['airin'] = False

        mleftslider.value = commands['tleft']
        mrightslider.value = commands['tright']
        zslider.value = commands['tup']

    #Communications
    received = rec.get()
    if received:
        try:
            tempdisplay.setValue('{:2.1f} C'.format(received.get('temp')))
            presdisplay.setValue('{:4.2f} mBar'.format(received.get('pres')))
        except ValueError:
            pass
        x = received.get('x')
        y = received.get('y')
        z = received.get('z')
        if x is not None:
            ahdisplay.roll = x
        if y is not None:
```

3

```
            ahdisplay.pitch = -y
        if z is not None:
            compassdisplay.yaw = z
    MESSAGE = json.dumps(commands)
    #print MESSAGE

    sock = socket.socket(socket.AF_INET,  # Internet
                         socket.SOCK_DGRAM)  # UDP
    sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))

    ##Drawing Stuff
    dheight = onstatus.get_height()
    screen.blit(onstatus.render(), (0, 0))
    screen.blit(turningdisplay.render(), (0, dheight))
    screen.blit(fpsdisplay.render(), (0, dheight*2))
    screen.blit(screenshot.render(), (0, dheight*3))
    screen.blit(tempdisplay.render(), (0, dheight*4))
    screen.blit(presdisplay.render(), (0, dheight*5))
    screen.blit(clawdisplay.render(), (0, dheight*6))
    screen.blit(airindisplay.render(), (0, dheight*7))
    screen.blit(airoutdisplay.render(), (0, dheight*8))
    screen.blit(depthholddisplay.render(), (0, dheight*9))
    screen.blit(ahdisplay.render(), (0, dheight*10))
    screen.fill((255, 255, 255), pygame.Rect(0, dheight*10+sidebarwidth, sidebarwidth, sidebarwidt
    screen.blit(compassdisplay.render(), (220/6, dheight*10+sidebarwidth))
    screen.blit(zslider.render(), (0, dheight*10+sidebarwidth*(1+2./3)))
    screen.blit(mleftslider.render(), (73, dheight*10+sidebarwidth*(1+2./3)))
    screen.blit(mrightslider.render(), (146, dheight*10+sidebarwidth*(1+2./3)))
    screen.blit(image.render(), (sidebarwidth, 0))

    pygame.display.flip()
    time.sleep(0.01)

pygame.quit()
```

Listing 2: widgets.py

```
## Provides methods for drawing various widgets on the screen

import pygame
import threading
import thread
import time
import cStringIO
import urllib2
import datetime
import math


class toggleable:
    def __init__(self, name, width):
        self.name = name
        self.width = width
        self.myfont = pygame.font.SysFont("monospace", 16)
        self.state = False

    def render(self):
        if self.state:
            text = self.myfont.render(self.name + ":  On", True, (0, 0, 0))
        else:
            text = self.myfont.render(self.name + ": Off", True, (0, 0, 0))
        background = pygame.Surface((self.width, text.get_height()))
```

4

```
        background.fill(((not self.state) * 150, self.state * 150, 0))
        background.blit(text, (0, 0))
        return background

    def get_height(self):
        return self.myfont.get_height()

    def toggle(self):
        self.state = not self.state

    def enable(self):
        self.state = True

    def disable(self):
        self.state = False


class display:
    def __init__(self, name, width):
        self.name = name
        self.width = width
        self.myfont = pygame.font.SysFont("monospace", 16)
        self.value = 0
        self.bgcolor = (255, 255, 255)

    def render(self):
        text = self.myfont.render(self.name + ": " + str(self.value), True, (0, 0, 0))
        background = pygame.Surface((self.width, text.get_height()))
        background.fill(self.bgcolor)
        background.blit(text, (0, 0))
        return background

    def get_height(self):
        return self.myfont.get_height()

    def setValue(self, value):
        self.value = value


class sliderdisplay:
    def __init__(self, name, width, height):
        self.name = name
        self.width = width
        self.height = height
        self.value = 0
        self.myfont = None

    def render(self):
        bar = pygame.Surface((self.width, self.height))
        bar.fill((230, 230, 230))

        #draw bar
        if self.value < 0:
            bar.fill((70, 70, 240), (0, self.height * 0.5, self.width, -self.value * self.height *
        else:
            bar.fill((70, 70, 240),
                     (0, (1 - self.value) * self.height * 0.5, self.width, self.value * self.heigh

        #draw tick marks
        for i in range(1, 10):
            pygame.draw.line(bar, (0, 0, 0), (0, self.height * i * 0.1), (self.width * .25, self.h
        pygame.draw.line(bar, (0, 0, 0), (0, self.height * 0.5), (self.width * 0.5, self.height *

        pygame.draw.rect(bar, (50, 50, 50), pygame.Rect(0, 0, self.width, self.height), 2)

        return bar


class compass:
    def __init__(self, size):
        self.width = size
        self.height = size
        self.yaw = 0
        self.compass = pygame.transform.smoothscale(pygame.image.load("heading.png"), (size, size)
        self.arrow = pygame.transform.smoothscale(pygame.image.load("arrow.png"), (size, size))

    def render(self):
        """rotate an image while keeping its center and size"""
        orig_rect = self.compass.get_rect()
        rot_image = pygame.transform.rotate(self.compass, self.yaw)
        rot_rect = orig_rect.copy()
        rot_rect.center = rot_image.get_rect().center
        out = rot_image.subsurface(rot_rect).copy()
        out.blit(self.arrow, (0, 0))
        return out


class ahorizon:
    def __init__(self, width):
        self.width = width
        self.height = width
        self.roll = 0
        self.pitch = 0
        self.myfont = pygame.font.SysFont("monospace", 14)

    @property
    def render(self):
        ah = pygame.Surface((self.width, self.height))
        ah.fill((30, 144, 255))
        if -90 <= self.pitch <= 90:
            pygame.draw.rect(ah, (255, 140, 0),
                             pygame.Rect(0, (self.pitch + 90) * self.height / 180.0, self.width, s
        elif -180 <= self.pitch < -90:
            pygame.draw.rect(ah, (255, 140, 0),
                             pygame.Rect(0, 0, self.width, (90 + (self.pitch + 180)) * self.height
        elif 90 < self.pitch <= 180:
            pygame.draw.rect(ah, (255, 140, 0), pygame.Rect(0, 0, self.width, (self.pitch - 90) *
        else:
            ah.fill((255, 0, 0))

        pygame.draw.polygon(ah, (0, 0, 0), [
            (self.width*0.475, self.height*0.05),
            (self.width/2, self.height*0.01),
            (self.width*0.525, self.height*0.05)])

        #rotate an image while keeping its center and size
        orig_rect = ah.get_rect()
        rot_image = pygame.transform.rotate(ah, self.roll)
        rot_rect = orig_rect.copy()
        rot_rect.center = rot_image.get_rect().center
        ah = rot_image.subsurface(rot_rect).copy()

        pygame.draw.line(ah, (0, 0, 0), (self.width * 7 / 20, self.height / 2), (self.width * 9 /
                         2)
        pygame.draw.line(ah, (0, 0, 0), (self.width * 11 / 20, self.height / 2),
                         (self.width * 13 / 20, self.height / 2), 2)

        for i in xrange(4, 16):
            pygame.draw.line(ah, (255, 255, 255), (self.width * 9 / 20, self.height * i / 18.0),
                             (self.width * 11 / 20, self.height * i / 18.0), 1)

        for i in xrange(21, 34):
            pygame.draw.line(ah, (255, 255, 255),
                             (self.width/2+0.475*self.width*math.cos(math.pi*i/18),
                              self.height/2+0.475*self.width*math.sin(math.pi*i/18)),
                             (self.width/2+self.width*math.cos(math.pi*i/18),
                              self.height/2+self.width*math.sin(math.pi*i/18)), 2)

        cover = pygame.Surface((self.width, self.height))
        cover.fill((255, 255, 255))
        pygame.draw.circle(cover, (0, 0, 0), (self.width / 2, self.height / 2), self.width / 2)
        cover.set_colorkey((0, 0, 0))
        ah.blit(cover, (0, 0))
        text = self.myfont.render('P:{:4.1f} deg'.format(self.pitch), True, (0, 0, 0))
        ah.blit(text, (0, self.height - text.get_height()))
        text = self.myfont.render('R:{:4.1f} deg'.format(self.roll), True, (0, 0, 0))
        ah.blit(text, (self.width - text.get_width(), self.height - text.get_height()))
        return ah


class ipimage:
    def downloader(self):
        while True:
            starttime = time.time()
            output = cStringIO.StringIO()
            try:
                output.write(urllib2.urlopen(self.url, timeout=1).read())
                output.seek(0)
                img = pygame.image.load(output)
                with self.thread_lock:
                    self.image = img
            except urllib2.URLError, e:
                myfont = pygame.font.SysFont("monospace", 16)
                with self.thread_lock:
                    self.image = myfont.render(str(e), True, (255, 0, 0))
            endtime = time.time()
            timetosleep = 1.0 / self.max_rate - (endtime - starttime)
            if timetosleep > 0:
                time.sleep(timetosleep)

    def render(self):
        return self.image

    def screenshot(self):
        if self.can_screenshot:
            self.can_screenshot = False
            filename = datetime.datetime.now().strftime("%y-%m-%d-%H-%M-%S-%f") + ".png"
            pygame.image.save(self.image, filename)

            def reenable():
                self.can_screenshot = True

            t = threading.Timer(1.0, reenable)
            t.start()

    def __init__(self, url):
        self.url = url
        self.image = pygame.Surface((1, 1))
        self.can_screenshot = True
        self.max_rate = 30  # per second
        self.thread_lock = threading.Lock()
        thread.start_new_thread(self.downloader, ())
```

Listing 3: receivedata.py

```
import threading
import errno
import socket
import datetime
import time
import json


class receivedata(threading.Thread):
    def __init__(self, name, IP, port):
        threading.Thread.__init__(self)
        self.datalock = threading.Lock()
        self.data = ""
        self.name = name
        self.timerec = 0
        self.sock = socket.socket(socket.AF_INET,  # Internet
                                  socket.SOCK_DGRAM)  # UDP
        self.sock.bind((IP, port))
        self.sock.setblocking(False)
        self.newData = False
        self.shouldStop = threading.Event()

        self.daemon = True

    def run(self):
        print "Starting " + self.name
        while True:
            try:
                with self.datalock:
                    self.data, addr = self.sock.recvfrom(1024)  # buffer size is 1024 bytes
            except socket.error, e:
                err = e.args[0]
                if err == errno.EAGAIN or err == errno.EWOULDBLOCK:
                    time.sleep(0.01)
                else:
                    print e
                    sys.exit(1)
            else:
                self.timerec = datetime.datetime.now()
                self.newData = True

    def get(self):
        if self.newData:
            self.newData = False
            with self.datalock:
                return json.loads(self.data)
        else:
            return None

    def __del__(self):
        self.sock.close()
```

Listing 4: rov.py

```
import receivedata
import time
import datetime
import json
import socket
import threading

import sabertooth
import Adafruit_BBIO.GPIO as GPIO

import MS5803
import mpu9150

prevtime = datetime.datetime.now()
timeout = False


class watchdog(threading.Thread):
    def __init__(self, threadID, name):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.daemon = True

    def run(self):
        print "Starting " + self.name
        while True:
            timedif = (datetime.datetime.now() - prevtime).total_seconds()
            global timeout
            if timedif > 0.5:
                timeout = True
            else:
                timeout = False
            time.sleep(0.5)


UDP_IP = "192.168.1.9"
#UDP_IP = "192.168.7.1"
UDP_PORT = 1870

r = receivedata.receivedata("ROV", '', UDP_PORT)
r.start()

watchdogthread = watchdog(1, "Watchdog")
watchdogthread.start()

sock = socket.socket(socket.AF_INET,  # Internet
                     socket.SOCK_DGRAM)  # UDP

s1 = sabertooth.Sabertooth(4, 128)
s2 = sabertooth.Sabertooth(4, 129)

#limit claw voltage, max rating is ~12V, not 20
s1.limitOutput(2, 0.65)

psense = MS5803.MS5803()
imu = mpu9150.mpu9150()
imu.init()
imu.start()

#Air out
GPIO.setup('P9_14', GPIO.OUT)
#Air in
GPIO.setup('P9_16', GPIO.OUT)

while True:
    mydata = r.get()
    if timeout:
        s1.move(1, 0)
        s1.move(2, 0)
        s2.move(1, 0)
        s2.move(2, 0)
    if mydata:
        prevtime = datetime.datetime.now()
        senddata = {}
        temp, pres = psense.read()
        if not (pres < 0 or temp < 0):
            senddata['temp'] = temp
            senddata['pres'] = pres

        senddata['x'] = imu.y
        senddata['y'] = imu.x
        senddata['z'] = imu.z
        s1.move('a', -mydata['tup'])  #s1a - top motor
        s1.move('b', mydata['claw'])
        s2.move('a', mydata['tright'])  #s2a - right motor
        s2.move('b', -mydata['tleft'])  #s1b - left motor

        if mydata['airout']:
            GPIO.output('P9_14', GPIO.HIGH)
        else:
            GPIO.output('P9_14', GPIO.LOW)
        if mydata['airin']:
            GPIO.output('P9_16', GPIO.HIGH)
        else:
            GPIO.output('P9_16', GPIO.LOW)

        MESSAGE = json.dumps(senddata)
        #print MESSAGE
        sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))

    time.sleep(0.01)
```

Listing 5: sabertooth.py

```
##Sabertooth motor controllers in Packetized Serial Mode for BBB

import Adafruit_BBIO.UART as UART
import serial

class Sabertooth:

  #command bytes
  forwardmotor1 = 0x00
  backwardmotor1 = 0x01
  forwardmotor2 = 0x04
  backwardmotor2 = 0x05
  motor17bit = 0x06
  motor27bit = 0x07
  serialtimeout = 0x0e

  def __init__( self, UARTnumber, address, debug = False):
    if not (128 <= address <= 135):
      raise ValueError("Address must be from 128 to 135")
    UART.setup("UART{:d}".format(UARTnumber))
    self.address = address
    self.ser = serial.Serial(port = "/dev/ttyO{:d}".format(UARTnumber), baudrate=9600)
    self.ser.close()
    self.ser.open()
    self.debug = debug

    self.motor1limit = 1.0
    self.motor2limit = 1.0

  def limitOutput(self, motor, limit):
    if motor in [1,'a','A']:
      self.motor1limit = limit
    elif motor in [2,'b','B']:
      self.motor2limit = limit
    else:
      raise ValueError("Invalid Motor")

  def command(self, command, data):
    checksum = (self.address+command+data) & 127
    packet = ''.join([chr(i) for i in [self.address, command, data, checksum]])
    if self.debug:
      print [ord(i) for i in packet]
    self.ser.write(packet)

  def setSerialTimeout(self, milliseconds):
    self.command(Sabertooth.serialtimeout, int(milliseconds/100))

  def move( self, motor, speed):
    if speed > 1 or speed < -1:
      raise ValueError("Invalid Speed")

    if motor in [1,'a','A']:
      if speed >= 0:
        self.command(Sabertooth.forwardmotor1,int(min(speed,self.motor1limit)*127))
      else:
        self.command(Sabertooth.backwardmotor1,int(min(-speed,self.motor1limit)*127))

    elif motor in [2,'b','B']:
      if speed >= 0:
        self.command(Sabertooth.forwardmotor2,int(min(speed,self.motor2limit)*127))
      else:
        self.command(Sabertooth.backwardmotor2,int(min(-speed,self.motor2limit)*127))

    else:
      raise ValueError("Invalid Motor")
```

Listing 6: mpu9150.py

```
##wrapper for the c module
#imu seems to require being constantly read at a steady rate, hence this class

import imu
import threading
import time


class mpu9150(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.x = 0.0
        self.y = 0.0
        self.z = 0.0
        self.daemon = True

    def init(self):
        self.imu = imu.imu()

    def run(self):
        print "Starting IMU data getter"
        while True:
            data = self.imu.get_euler_angles()
            if data:
                self.x, self.y, self.z = data
            time.sleep(0.02)  # Sensor is set for 50 Hz
```

Listing 7: ms5803.c (C program for reading pressure sensor)

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
#include <linux/i2c-dev.h>

int file;
int adapter_nr = 1;
char filename[20];
char buf[10];
int i;

unsigned long   D1                          = 0;    // Stores uncompensated pressure value
unsigned long   D2                          = 0;    // Stores uncompensated temperature value
float           deltaTemp                   = 0;    // These three variable are used for the conver
float           sensorOffset                = 0;
float           sensitivity                 = 0;

float temp = 0;
float press = 0;

unsigned int sensorCoefficients[8];


#define PRESSURE 0b0
#define TEMPERATURE 0b10000

#define OSR256 0b0
#define OSR512 0b10
#define OSR1024 0b100
#define OSR2048 0b110
#define OSR4096 0b1000

int raw_convert(char type, char OSRlevel)
{
   int result;
   char command = 0x40+type+OSRlevel;
   if (write(file, &command, 1) != 1){
        printf("Error sending init conversion to device\n");
    }
    switch ( OSRlevel )
     {
         case OSR256 :
             usleep( 1000 );
             break;
         case OSR512 :
             usleep( 3000 );
             break;
         case OSR1024:
             usleep( 4000 );
             break;
         case OSR2048:
             usleep( 6000 );
             break;
         case OSR4096:
             usleep( 10000 );
             break;
     }

    usleep(3000);

    command = 0x00;
    if (write(file, &command, 1) != 1){
        printf("Error sending read sequence to device\n");
    }
    usleep(1000);
    if(read(file, buf, 3) != 3) {
        printf("Error reading ADC\n");
        exit(1);
   } else {
           result = (buf[0]<<16) + (buf[1]<<8) + (buf[0]);
   }
   return result;
}

void readSensor(){
  // If power or speed are important, you can change the ADC resolution to a lower value.
  // Currently set to SENSOR_CMD_ADC_4096 - set to a lower defined value for lower resolution
  D1 = raw_convert( PRESSURE, OSR4096 );    // read uncompensated pressure
  D2 = raw_convert( TEMPERATURE, OSR4096 );    // read uncompensated temperature

  // calculate 1st order pressure and temperature correction factors (MS5803 1st order algorithm).
  deltaTemp = D2 - sensorCoefficients[5] * pow( 2, 8 );
  sensorOffset = sensorCoefficients[2] * pow( 2, 16 ) + ( deltaTemp * sensorCoefficients[4] ) / po
  sensitivity = sensorCoefficients[1] * pow( 2, 15 ) + ( deltaTemp * sensorCoefficients[3] ) / pow

  // calculate 2nd order pressure and temperature (MS5803 2st order algorithm)
  temp = ( 2000 + (deltaTemp * sensorCoefficients[6] ) / pow( 2, 23 ) ) / 100;
  press = ( ( ( ( D1 * sensitivity ) / pow( 2, 21 ) - sensorOffset) / pow( 2, 15 ) ) / 10 );
}

int readPROM(int pos)
{
    //read PROM command
    char command = 0xA0+2*pos;
    int result;

    if (write(file, &command, 1) != 1){
        printf("Error sending read PROM to device\n");
```

```
      //exit(1);
    }

  if(read(file, buf, 2) != 2) {
      printf("Error reading PROM\n");
      //exit(1);
  } else {
         result = (buf[0]<<8) + buf[1];
  }
  return result;
}

void reset(){
   char command = 0x1E;

   if (write(file, &command, 1) != 1){
      printf("Error sending reset to device\n");
   }
   usleep(10000);
}

unsigned char CRC(unsigned int cn_prom[]){
    int cnt;
    unsigned int n_rem;
    unsigned int crc_read;
    unsigned char  n_bit;

    n_rem = 0x00;
    crc_read = sensorCoefficients[7];
    sensorCoefficients[7] = ( 0xFF00 & ( sensorCoefficients[7] ) );

    for (cnt = 0; cnt < 16; cnt++)
    { // choose LSB or MSB
        if ( cnt%2 == 1 ) n_rem ^= (unsigned short) ( ( sensorCoefficients[cnt>>1] ) & 0x00FF );
        else n_rem ^= (unsigned short) ( sensorCoefficients[cnt>>1] >> 8 );
        for ( n_bit = 8; n_bit > 0; n_bit-- )
        {
            if ( n_rem & ( 0x8000 ) )
            {
                n_rem = ( n_rem << 1 ) ^ 0x3000;
            }
            else {
                n_rem = ( n_rem << 1 );
            }
        }
    }

    n_rem = ( 0x000F & ( n_rem >> 12 ) );// // final 4-bit reminder is CRC code
    sensorCoefficients[7] = crc_read; // restore the crc_read to its original place

    return ( n_rem ^ 0x00 ); // The calculated CRC should match what the device initally returned.
}

int init(char adapter_nr, char address)
{
  snprintf(filename, 19, "/dev/i2c-%d", adapter_nr);
  file = open(filename, O_RDWR);

  if (file < 0){
    printf("Error opnening file\n");
    return -1;
  }

  if (ioctl(file, I2C_SLAVE, address) <0){
    printf("Error setting up bus for slave operation\n");
    return -2;
  }

  reset();

  //Read and store coefficients from PROM
  for(i=0; i<8;i++){
    sensorCoefficients[i] = readPROM(i);
    usleep(10000);
  }

  unsigned char p_crc = sensorCoefficients[ 7 ];
  unsigned char n_crc = CRC( sensorCoefficients ); // calculate the CRC

  // If the calculated CRC does not match the returned CRC, then there is a data integrity issue.
  // Check the connections for bad solder joints or "flakey" cables.
  // If this issue persists, you may have a bad sensor.
  if ( p_crc != n_crc ) {
    printf("CRC ERROR\n");
    return -3;
  }else{
    //printf("CRC OK\n");
  }
  return 1;
}

int main(int argc, char *argv[])
{
  const char bus = 1; // I2C bus the sensor is on
  const char addr = 0x76; // Address of the Pressure Sensor

  if(init(bus,addr)>0){

    printf("Starting Read\n\n");
      struct timeval tp;
    while(1){
      gettimeofday(&tp,NULL);
      unsigned long long millisecondsSinceEpoch =
      (unsigned long long)(tp.tv_sec) * 1000 +
      (unsigned long long)(tp.tv_usec) / 1000;
      readSensor();
      printf("\rTemperature: %f C, Pressure: %f mBar, Time: %llu",temp,press,millisecondsSinceEpoc
      fflush(stdout);
    }
  }
  return 0;
}
```

Listing 8: ms5803module.c (Python extension for the pressure sensor)

```
#include <Python.h>
#include "structmember.h"
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
```

```c
#include <linux/i2c-dev.h>

#define PRESSURE 0b0
#define TEMPERATURE 0b10000

#define OSR256 0b0
#define OSR512 0b10
#define OSR1024 0b100
#define OSR2048 0b110
#define OSR4096 0b1000

int i;
unsigned int sensorCoefficients[8];

typedef struct {
  PyObject_HEAD
  char adapter_nr;
  char address;
  int file;
} MS5803;

void reset(MS5803* self){
   char command = 0x1E;

   if (write(self->file, &command, 1) != 1){
      printf("Error sending reset to device\n");
   }
   usleep(1000);
}

int raw_convert(MS5803* self, char type, char OSRlevel)
{
   int result = 0;
   char buf[3];
   char command = 0x40+type+OSRlevel;

   if (write(self->file, &command, 1) != 1){
      printf("Error sending init conversion to device\n");
   }
   switch ( OSRlevel )
    {
        case OSR256 :
            usleep( 1000 );
            break;
        case OSR512 :
            usleep( 3000 );
            break;
        case OSR1024:
            usleep( 4000 );
            break;
        case OSR2048:
            usleep( 6000 );
            break;
        case OSR4096:
            usleep( 10000 );
            break;
    }

   usleep(3000);

   command = 0x00;
```

16

```
    if (write(self->file, &command, 1) != 1){
       printf("Error sending read sequence to device\n");
    }
    usleep(1000);
    if(read(self->file, buf, 3) != 3) {
       printf("Error reading ADC\n");
   } else {
          result = (buf[0]<<16) + (buf[1]<<8) + (buf[0]);
   }
   return result;
}

int readPROM(MS5803* self, int pos)
{
    //read PROM command
    char command = 0xA0+2*pos;
    int result = -1;
    char buf [2];

    if (write(self->file, &command, 1) != 1){
       printf("Error sending read PROM to device\n");
    }

   if(read(self->file, buf, 2) != 2) {
       printf("Error reading PROM\n");
   } else {
          result = (buf[0]<<8) + buf[1];
   }
   return result;
}

unsigned char CRC(MS5803* self){
     int cnt;
     unsigned int n_rem;
     unsigned int crc_read;
     unsigned char  n_bit;

     n_rem = 0x00;
     crc_read = sensorCoefficients[7];
     sensorCoefficients[7] = ( 0xFF00 & ( sensorCoefficients[7] ) );

     for (cnt = 0; cnt < 16; cnt++)
     { // choose LSB or MSB
         if ( cnt%2 == 1 ) n_rem ^= (unsigned short) ( ( sensorCoefficients[cnt>>1] ) & 0x00FF );
         else n_rem ^= (unsigned short) ( sensorCoefficients[cnt>>1] >> 8 );
         for ( n_bit = 8; n_bit > 0; n_bit-- )
         {
             if ( n_rem & ( 0x8000 ) )
             {
                 n_rem = ( n_rem << 1 ) ^ 0x3000;
             }
             else {
                 n_rem = ( n_rem << 1 );
             }
         }
     }

     n_rem = ( 0x000F & ( n_rem >> 12 ) );// // final 4-bit reminder is CRC code
     sensorCoefficients[7] = crc_read; // restore the crc_read to its original place

     return ( n_rem ^ 0x00 ); // The calculated CRC should match what the device initally returned.
```

17

```
}

static PyObject *
readSensor(MS5803* self){

  // If power or speed are important, you can change the ADC resolution to a lower value.
  // Currently set to SENSOR_CMD_ADC_4096 - set to a lower defined value for lower resolution
  unsigned long D1 = raw_convert(self, PRESSURE, OSR4096 );   // read uncompensated pressure
  unsigned D2 = raw_convert(self, TEMPERATURE, OSR4096 );    // read uncompensated temperature

  // calculate 1st order pressure and temperature correction factors (MS5803 1st order algorithm).
  double deltaTemp = D2 - sensorCoefficients[5] * pow( 2, 8 );
  double sensorOffset = sensorCoefficients[2] * pow( 2, 16 ) + ( deltaTemp * sensorCoefficients[4]
  double sensitivity = sensorCoefficients[1] * pow( 2, 15 ) + ( deltaTemp * sensorCoefficients[3]

  // calculate 2nd order pressure and temperature (MS5803 2st order algorithm)
  double temp = ( 2000 + (deltaTemp * sensorCoefficients[6] ) / pow( 2, 23 ) ) / 100;
  double press = ( ( ( ( D1 * sensitivity ) / pow( 2, 21 ) - sensorOffset) / pow( 2, 15 ) ) / 10 )

  return Py_BuildValue("dd", temp, press);

}

static void MS5803_dealloc(MS5803* self)
{
  self->ob_type->tp_free((PyObject*)self);
}

static PyObject *
MS5803_new(PyTypeObject *type, PyObject *args, PyObject *kwds)
{
  MS5803 *self;

  self = (MS5803 *)type->tp_alloc(type, 0);
  if (self != NULL) {
    self->file = 0;
    self->adapter_nr = 1;
    self->address = 0x76;
    for(i = 0; i<8; i++){
      sensorCoefficients[i]=0;
    }
  }

  return (PyObject *)self;
}

static int
MS5803_init(MS5803 *self, PyObject *args, PyObject *kwds)
{
  static char *kwlist[] = {"adapter_nr", "address", NULL};

  if(! PyArg_ParseTupleAndKeywords(args, kwds, "|ii", kwlist,
            &self->adapter_nr, &self->address))
    return -1;

  char filename[20];
  snprintf(filename, 19, "/dev/i2c-%d", self->adapter_nr);
  self->file = open(filename, O_RDWR);

  if (self->file < 0){
    printf("Error opening file\n");
```

18

```
    return -1;
  }

  if (ioctl(self->file, I2C_SLAVE, self->address) <0 ){
    printf("Error setting up bus for slave operation\n");
    return -1;
  }

  reset(self);

  //Read and store coefficients from PROM
  for(i=0; i<8;i++){
    sensorCoefficients[i] = readPROM(self,i);
    usleep(10000);
  }

  unsigned char p_crc = sensorCoefficients[ 7 ];
  unsigned char n_crc = CRC(self); // calculate the CRC

  // If the calculated CRC does not match the returned CRC, then there is a data integrity issue.
  // Check the connections for bad solder joints or "flakey" cables.
  // If this issue persists, you may have a bad sensor.
  if ( p_crc != n_crc ) {
    printf("CRC ERROR\n");
    return -1;
  }else{
    //printf("CRC OK\n");
  }

  return 0;
}

static PyMemberDef MS5803_members[] = {
  {"adapter_nr", T_INT, offsetof(MS5803, adapter_nr), 0,
   "first name"},
  {"address", T_INT, offsetof(MS5803, address), 0,
   "last name"},
  {NULL} /*Sentinel */
};

static PyMethodDef MS5803_methods[] = {
  {"read", (PyCFunction)readSensor, METH_NOARGS,
  "Returns the sensor readings"

  },
  {NULL}  /*Sentinel */
};

static PyTypeObject MS5803Type = {
  PyObject_HEAD_INIT(NULL)
  0,         /* ob_size        */
  "MS5803.MS5803",    /* tp_name        */
  sizeof(MS5803),     /* tp_basicsize   */
  0,         /* tp_itemsize    */
  (destructor)MS5803_dealloc, /* tp_dealloc     */
  0,         /* tp_print       */
  0,         /* tp_getattr     */
  0,         /* tp_setattr     */
  0,         /* tp_compare     */
  0,         /* tp_repr        */
  0,         /* tp_as_number   */
```

19

```
  0,          /* tp_as_sequence */
  0,          /* tp_as_mapping  */
  0,          /* tp_hash        */
  0,          /* tp_call        */
  0,          /* tp_str         */
  0,          /* tp_getattro    */
  0,          /* tp_setattro    */
  0,          /* tp_as_buffer   */
  Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE, /* tp_flags       */
  "MS5803- Pressure Sensor",  /* tp_doc          */
  0,
  0,
  0,
  0,
  0,
  0,
  MS5803_methods,
  MS5803_members,
  0,
  0,
  0,
  0,
  0,
  0,
  (initproc)MS5803_init,
  0,
  MS5803_new,
};

static PyMethodDef module_methods[] = {
  {NULL}
};


PyMODINIT_FUNC
initMS5803(void)
{
  PyObject* m;

  if (PyType_Ready(&MS5803Type) < 0)
    return;

  m = Py_InitModule3("MS5803", NULL,
         "Example module that creates an extension type.");
  if (m == NULL)
    return;

  Py_INCREF(&MS5803Type);
  PyModule_AddObject(m, "MS5803", (PyObject *)&MS5803Type);
}
```