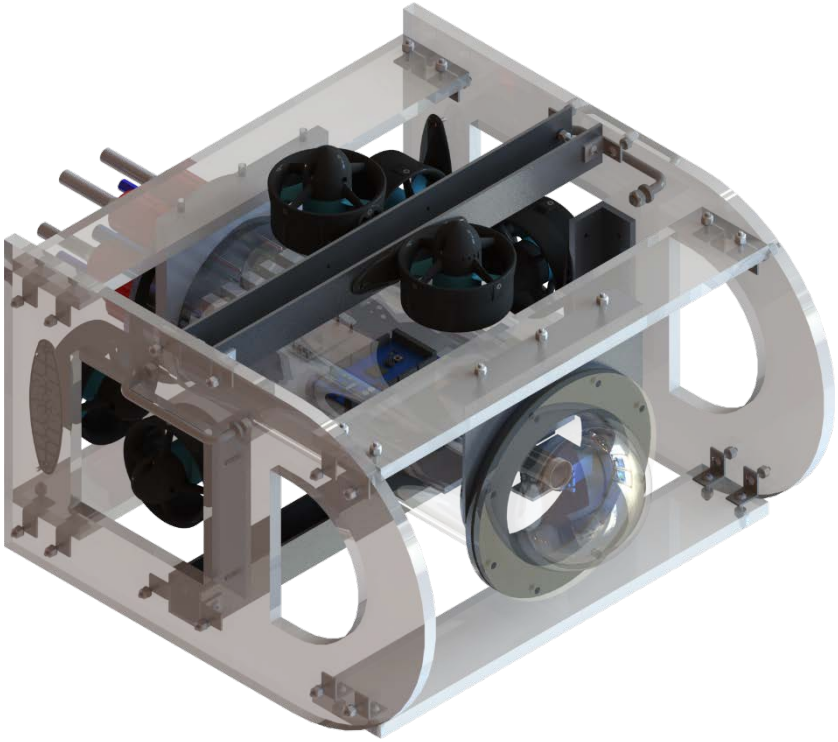
UNH ROV



Team Members:

Project Leader: Jarrett Linowes

Controls Team:

Nathaniel Cordova (Operations Manager)

Gerald Rosati (TL)

Loker Xu

Jiaqing Ye

Sathya Muthukkumar

Alex Sarassin

Chassis Team:

Shaun Hespelein (TL)

Sayward Allen (Treasurer)

Alex Dzengeleski

Contents

Abstract:

UNH ROV is an interdisciplinary engineering team focused on designing, fabricating, and competing with an underwater Remotely Operated Vehicle (ROV). UNH ROV pairs as a senior design project and a student organization while also supporting graduate level research. The design incorporates a high degree of modularity by accounting for constraints defined by the international MATE ROV Competition as well as PhD research specifications involving autonomous Unmanned Underwater Vehicle (UUV) control. The MATE ROV Competition features 3 missions which have unique objectives relating to an arctic theme, while the research involves using an optical feedback system for pose control of multiple UUVs. Each part of the design is conceived using 3D modeling software, analyzed using finite element simulation packages, and verified through a prototyping and testing process. UNH ROV also focuses on maintaining a diverse team of freshman to graduate level students to maximize innovative and creative design. By using cutting-edge methods in team dynamics, analytic tools, and engineering design, UNH ROV is able to maintain continued success in developing underwater robotic systems used for multiple different platforms for research and competition.

Purpose and Goals

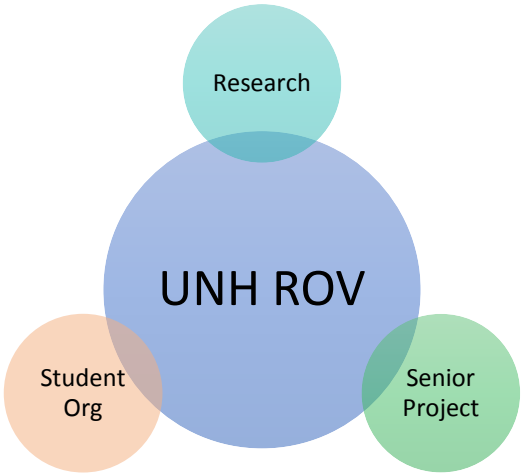


Figure 1 – Concept Web of UNH ROV team design

The UNH ROV team is a multi-facetted group which defines its purpose and goals from many sources. First, as a research support team, UNH ROV designs ROV platforms capable of supporting autonomous control for graduate-level research. Second, as a senior design project, UNH ROV designs highly capable ROVs to compete in the International MATE ROV Competition. Third, as a Student Organization, UNH ROV is committed to advocating STEM education in the throughout the University of New Hampshire, area high schools, and the greater Durham community.

Research

One cornerstone of the UNH ROV team is its support of graduate level research. PhD student Firat Eren has been composing control algorithms for a leader-follower system for UUVs entitled: “Pose Detection and Control of Unmanned Underwater Vehicles (UUVs) Utilizing an Optical Detector Array”. These algorithms will utilize an optical feedback mechanism designed as an array of photodiodes to be attached to the front of the UUV. Using this photodiode array, movement in the X,Y,Z, and Yaw directions can be controlled to either dock a UUV (static-dynamic case) or follow a leader UUV (dynamic-

dynamic case). The control system uses a method called Spectral Angle Mapping (SAM) where a pose angle, α, is calculated by taking the dot product of a light intensity array and shape array based on the geometry of the sensor. This angle is then used in the control algorithms by creating a matrix of moment invariants, which is a non-dimensionalized ratio describing the error in the pose of the follow UUV with respect to the leader UUV's light source. Finally, the control systems used Sliding Mode Control (SMC) based on these moment invariants to prescribe the amount of thrust needed correct the pose of the follower UUV.

The job of the UNH ROV team is to supply Firat Eren with modular ROV platforms to be used in his research by satisfying a few design constraints derived from research requirements. These platforms must feature a big enough area on the front of the ROV, with necessary mounting points on the frame, to accommodate the photodiode array. As a result of this, the ROV must retain a proper ballast by counter-balancing the added weight of the array. With respect to the electronics, the end cap of the electronics tube must also feature a waterproof USB interface where the electronics from the photodiode array can integrate with the rest of the control system. Lastly, space on the electronics board for an on board computer (OBC) must be present so the ROV can compute the thrust outputs of the control algorithms needed to maintain proper pose.

With future goals of creating a fleet of UUVs to be linked with Autonomous Surface Vehicles (ASV), the UNH ROV team develops two ROVs over the course of one calendar year, one during the school year, and another during the summer. This enables the team to improve upon their design for a future iteration quickly.

Senior Project

The UNH ROV team has been a Senior Project since 2008. Each year the team designs and builds a new ROV using the past generations as a foundation for new advancements. Participation in the Marine Advancement Technology Education (MATE) Center competition has become an annual tradition, and the competition's mission tasks push teams to build a versatile machine.

The cross-disciplined team recruits across many CEPS majors including Mechanical Engineering, Electrical Engineering, Computer Engineering, and Computer Science. The multi-major outreach is necessary given that the ROV cannot be successfully built with just a single discipline. Although the team accepts all undergraduates, those entering their senior year may choose to have UNH ROV as their Senior Capstone Project. The organization fills its ranks by reaching out to underclassmen through its weekly open-invitation meetings as well as participating in open-house and recruitment events such as U-Day and Ocean Day. Interested underclassmen are free to be involved and help the seniors with the ROV build every step of the way and potentially utilize the ROV team as their Senior Capstone Project. The involvement of underclassmen fosters team leaders by allowing them gain early access to the team's knowledge base.

The ROV's chassis must be designed and machined such that the least amount of water displacement occurs during its movement, while maintaining the strength to withstand the propulsive forces generated from the thrusters. Successful implementation of this design requires an understanding of fluid and body dynamics as well as 3D-modeling simulations, knowledge of which is exclusive only to those of the Mechanical Engineering discipline.

Given the competition's design constraints, the ROV must be remotely operated from the surface using a tether. Power must be properly stepped down from its original 48V and supplied to the microcontrollers, electronic speed controllers, thrusters, and as well as the sensor suite. A task well suited to those of the Electrical Engineering discipline.

Finally, the human interface controllers such as the Xbox and Leap Motion controllers utilized in this year's design must properly control the ROV's movements. Camera feed must be sent up to the driver station and sensor data must be updated in real time and sent to the microcontrollers as well as the graphic user interface. This task can only be accomplished by a knowledgeable Computer Engineer or Computer Scientist. Given the required design specifications, and interdisciplinary nature of the design project, all three majors are integral to the success of the UNH ROV.

Student Organization and Outreach

UNH ROV is a student organization focused on interdisciplinary involvement in a fast paced engineering underwater robotics team. Students from freshmen through graduate levels work in conjunction with each other, focused on furthering the development of individual understanding of robotics and cutting edge research for UUVs. The team works with local high schools and groups such as SeaPearch with goals of outreach to younger students interested in STEM programs. Students working with the UNH ROV team are involved in projects relating to mechanical engineering, electrical engineering, ocean engineering, and computer science.

Team Dynamics and Structure

Effectiveness

The UNH ROV team operates effectively in a business format in order to thoroughly manage everyone's abilities, interests, and workload. As a business style management system, dynamic changes and improvements to team structure are constantly underway, allowing members with deeper knowledge or interest in certain areas to take leadership roles. Due to the interdisciplinary nature of the team, each member is able to work on team responsibilities that interest them.

The primary mode of communication between team members is associated with the design process and engineering team management. Pros and cons, potential designs with respect to constraints, and potential control interface and logic were raised by members throughout the design process. Leadership shifts between members of higher knowledge in areas of the design process led the discussions.

Team Diagram

As a model business, the team diagram depicts the general leadership roles from faculty advisors through individual team leadership. Although the leadership structure is documented, the team dynamics allow members to move through teams fluidly and provide leadership where necessary. The team's business model revolves around thorough financial management as well as communication. The CEO and Operations Manager focus their efforts on project management, team image, and scheduling constraints in order to ensure a smoothly operating team. The financial management is led by the treasurer, in charge of sponsorship fundraising, grant applications, as well as cash flow documentation.

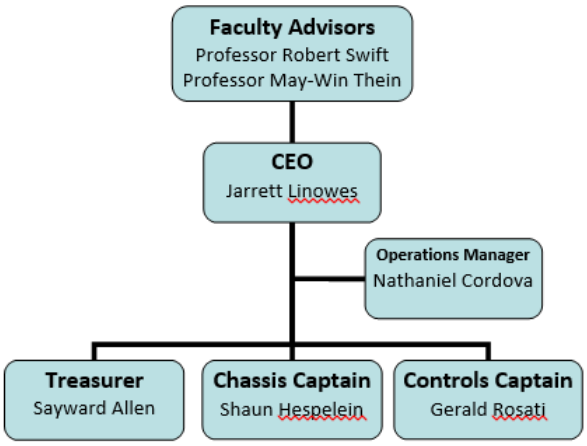


Figure 2 - Team diagram flow chart

In order to efficiently organize engineering tasks to enhance the effectiveness of the team, the members were grouped into a chassis team and a controls team. The chassis team focused primarily on the design, improvement, and manufacturing of a frame and propulsion system. The controls team focused on the intuitive control of the robotic system using innovative human interaction devices and implementation of the onboard sensors and manipulators. Because there is a direct correlation between chassis design and the effectiveness of the control system, these teams work closely together to establish the cross-team design constraints. An important aspect of parallel team management is the distribution of information and developments between members of each team. It is the responsibility of the leaders of the team to interface both chassis and control designs at a systems level to create an effective ROV platform.

Conceptual Design

Initial Design

The overall design was inspired by other ROVs that are manufactured worldwide. Most ROVs have 6 degree of freedoms to control movement in all orientations. This year's design allows for 5 degrees of freedom and mobility, excluding roll. Outside of the thruster design to ensure mobility, it is desired to have unrestrained fluid flow through the chassis. This was a major concern as fluid flow enables the ROV to push water, which according to Newton's 3rd Law, will push the ROV back allowing restricting the ROV from maneuvering through the water.

Another design attribute was to create a compact size with few failure points. Limiting failure points is one of the most important and emphasized design aspects. One failure point is waterproofing. This year's design addressed this by building the chassis around a single enclosed tube that would house all of the electronics in a single dry environment. This limited the points of failure to two, the electronics tube end cap and the dome on the front of the electronics tube. As for the size of the chassis, it was ideal for the design to be a size fit for traveling, as it will be traveling to the MATE ROV Competition. In order to transport the ROV to the competition, it is best to keep the overall chassis as compact as possible.

Design Goals through MATE Competition

The Marine Advanced Technology Education center hosts an annual international collegiate underwater ROV competition. Each year the competition is held at a different location, this year it is held in St. John's, Newfoundland Canada. At the competition, each team must be operate as an engineering company, and their ROV is the product to be marketed. There are 580 total points attributed to the competition, 300 of which belong to the product demonstrations, 250 from a sales demonstration, and 30 points for safety analysis. The competition's mission tasks are split between 3 different tanks filled where each one of these tanks are considered to be product demonstrations showcasing the ROV's capabilities. During the entire demonstration period, the driver will not be able to see the ROV, and must maneuver the ROV using only the visual guidance of the camera and sensor values displayed to the driver station.

The first product demonstration, labeled "Science Under the Ice", tasks the team to maneuver their ROV and complete missions while under an ice sheet found on the surface of the water. Companies must first launch their ROV through a 75 cm by 75 cm hole in the ice sheet. The following tasks in this product demonstration can be conducted in any order. Companies will be required to retrieve samples of algae and sea urchins, identify sea stars, deploy an acoustic sensor, survey and measure the dimensions of an iceberg and determine the iceberg's threat level to a variety of surface and subsea assets.

- The samples of algae are simulated by ping pong balls and are found under the ice sheet.
- The sea urchins are simulated by a 4 inch O-ball, and are found on the sea floor.
- The sea stars will vary in color and will be constructed out of ½ inch PVC and are also found on the sea floor. Identification will be conducted by visual inspection and comparison to a handbook.
- The passive acoustic sensor must be deployed upright into a 50cm by 50cm area. The sensor will be constructed out of ½ inch PVC pipe and connected with a cable to a 3 inch float. The sensor will weigh less than 30 N underwater.
- The iceberg will be constructed out of ½ inch PVC as well, and a completed survey is accomplished by showing the judge the labeling found on the sides of the iceberg through the ROV's camera. Once a survey is completed, the judge will release the heading and coordinates of the iceberg.
- The iceberg's diameter and depth must be measured. Using these dimensions, the volume of the iceberg must be calculated.
- The iceberg's location must be plotted by the company onto a map of the demonstration area.
- Companies must then calculate the iceberg's threat level to 4 surface and 4 subsea assets, given the knowledge of the heading and position of the iceberg.

The second product demonstration, labeled "Subsea Pipeline Inspection and Repair", tasks the company to maneuver their ROV to complete missions in an offshore engineering basin. The basin will have rough waters and windy conditions, and tasks in this demonstration must be completed in order. Companies will be expected to conduct a close visual inspection of a pipeline, identify the corroded area on this pipeline, turn a valve to top flow through the pipeline and check a gauge to ensure that this is the case, then attach a lift line to this corroded area, cut the corroded area off, bring the corroded section to the surface, and finally install a flange over the cut ends of the pipeline that is still underwater. Next the company must use their ROV to remove a wellhead's protective cover, install a gasket, replace the

cover, then insert a hot stab into the side of the wellhead, and then bring the hot stab back up to the surface.

- The pipeline is simulated with both 1 ½ inch PVC and ½ inch PVC.
- Corrosion will be simulated by a brown circle.
- The valve will be constructed of ½ inch PVC and a brass gate valve, and must be turned 3.25 times around.
- The pressure gauge dial reading of zero must be displayed to the judge.
- Cutting the pipeline is simulated by pulling two U-bolts.
- The wellhead is simulated by a 2 inch PVC pipe with a 2 to 3 inch adapter on the top. The protective cover is simulated by a 4 inch PVC end cap.
- The hot stab is constructed out of 1 ½ inch PVC and a ½ inch PVC grab point.

The final demonstration, titled "Offshore Oilfield Production and Maintenance", takes place in a flume tank in which the ROV must endure a constant current of up to 0.25 meters per second. In this demonstration, companies must test the grounding of anodes by measuring the voltage of four points along the leg of an oil platform. They must then calculate the angle at which a wellhead emerges from the sea floor. Companies must also evaluate a pipeline map, turn valves to open or close specific pathways, and then move water through the pipeline to verify that flow through the correct pathway has been established. Finally companies must deploy their own sensor to determine the flume tank's constant flow rate.

- ROV's can only be pushed 'out of bounds' by the constant flow rate 3 times before being disqualified in this demonstration.
- The leg of the oil platform will be constructed with 2 inch PVC and 2 inch couplings, and will be rising out of the sea floor at an angle between 65° and 80°.
- The four test points are labeled A, B, C, and D. The points will be constructed from 10-24 bolt with wires will be attached to them from inside the pipe. Judges will have control of switches to control the current to the test points.
- To calculate the angle of the wellhead, which is constructed from 2 inch PVC, the height and length must first be calculated at the appropriate angle.
- The pipeline map has one input, and 4 outputs. The pipe will be constructed from ½ inch PVC and the valves from a ½ inch gate valve.

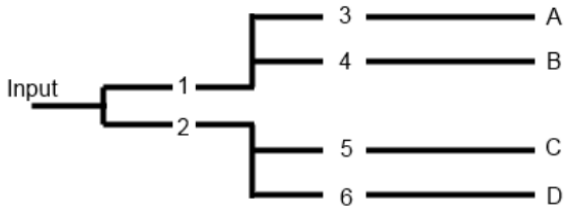


Figure 3 - Mission task oil pipeline representation

- The judge will specify which output is desired, and the company must use their ROV to close or open the six valves by rotating them 3.25 times around.

- Once the valves have been set to the proper position, companies must use their ROV to move water through the input of the pipeline to the desired output using their own device.

Considering Constraints

The MATE competition provides multiple design constraints that introduce unique engineering challenges. The first constraint states that the ROV must fit through a 75x75cm hole in the ice, limiting the size of the overall design. The second constraint states that a tethered connection must carry power and data to and from the ROV. Visual guidance from the ROVs perspective is not explicitly required, however it would be very helpful with navigation. The purpose of the driver station is to simulate driving the ROV from a scenario in which the robot will not be visible. In order to drive under such situations it was determined that a camera must be mounted to the front of the ROV in order to stream real time video feed to the driver station. The last constraint that was considered from the start was to keep the design modular enough to maintain the ability to manipulate objects and the environment underwater. While at competition, in June, it will be necessary to complete the mission tasks to prove superiority amongst the other teams. In order to complete the task the ROV must be driven through the water with ease as well as pick up and deliver objects with precision to and from the sea floor.

Cost, Schedule, Risk

Each product demonstration conducted during the competition can receive a maximum of 100 points, for a total of 300 points across all demonstrations. Mission tasks can be very elaborate and may demand the incorporation of a component not normally found on the ROV, except to only complete that specific task. Such is the case of measuring the grounding of anodes during the third demonstration. However, something simple like a hook can be utilized to complete many of the tasks. Thus the hook is more valuable than the underwater multi-meter attachment in terms of accruing points, while being much less costly to implement. Given time constraints during the academic year, the budget restraints from a finite set of donors, and the travelling costs to participate in the competition; not all mission tasks at the MATE competition can be feasibly completed by the UNH ROV team this year. As such, each demonstration task has been broken down into their respective points and weighed on the expense it would cost to pursue the points.

Through analysis of the competition, the UNH ROV team was able to give priority to certain ROV subsystems to effectively complete mission tasks. This was done by adding up all the points for each mission task that would require use of specific category of subsystem.

Table 1 - Point value analysis

Name of Subsystem	Number of Points	% Total From Product Demo
Manipulator	110	36.7
Camera	80	26.7
Sonar	60	20.0
Voltmeter	40	13.3
Flow Sensor	10	3.3

The UNH ROV team decided to pursue the top three most heavily weighted subsystems to compete with in the MATE competition. However in further analysis of the requirements of each specific mission task, it was determined that a single manipulator would not be sufficient enough to obtain all of the points

relevant to the manipulator subsystem. In the mission tasks, there is a wide range of weights and geometries that the manipulator subsystem must be able to handle. It would be too costly to develop a single robust robotic manipulator to be able to life the heavier objects as well have the precision for the objects with more complex geometry. It would also involve too much scheduling time that would take away time from the other significant portions of the ROV. And lastly, having a more complex manipulator increases the number of both hardware and software based failure modes.

Two separate manipulators were developed in addition to the static hook; the robotic manipulator and the static manipulator. With respect to the mission tasks, heavier objects which require a robust design make up 45.5% of the manipulator-related mission tasks while lighter objects with more complex geometry make up the remaining 54.5%. Because the quantification of these manipulator based mission tasks are nearly equal in point value, it was deemed necessary to split the tasks between two more specific manipulator subsystems. The cost of using these two systems may at first seem greater than that of one manipulator, however leveraging the design of last year's robotic manipulator alleviated most of the cost of prototyping and designing a new manipulator. With respect to schedule it's was unnecessary to allocate man hours to design a new prototype. And also since the last year's prototype worked, there was little risk involved with choosing this design again for this ROV. In contrast, from the robotic manipulator, the static manipulators are relatively rigid apart from fasteners or springs. They do not require a power source to operate, and this reduces a potential mode of failure for the appendage.

The camera quality was of the next highest priority given that it was the next highest in the point value analysis. An off-the-shelf Microsoft HD Lifecam was chosen as the camera of choice, given its low cost and 1080p quality. Furthermore there is little risk in using this camera because the drivers automatically worked with the Microsoft based operating system for the driver station. Also because of the camera's point priority, the camera feed was given its own isolated USB extender to minimize loss of resolution.

The last subsystem that was implemented on the ROV is the sonar. The sonar of choice was the MaxBotix 7076 sonar system. This sonar system was very easy to further waterproof, considering it was already water resistant with a rating of IP67. Because of the all-in-one nature of this sensor, it is very easy to use and therefore reduces the amount of time it takes to implement the sensor.

The remaining subsystems in the point value analysis were deemed not worth the overall cost, schedule, and risk associated with implementing them on the ROV design. This is because of their low point value and the time that it would take to develop and prototype these subsystems from scratch. In conclusion, the point value analysis paired with a cost, schedule, and risk analysis enabled the UNH ROV team to make efficient and worthwhile design decisions in developing this year's ROV.

Priority

The main priority for the ROV design was a waterproof electronics housing, as without one the project would be dead in the water. Waterproofing has been a major struggle for this project with past designs commonly resorting to using a lot of messy putties and tape to seal water and air interfaces. In order to form a well-sealed design the team did some preparation and research on gaskets and O-rings as these are the proper ways to seal a water tight enclosure. The design consists of 4 O-rings on the end caps as well as a wide flanged gasket on the face of the dome and the fixed dome connect aluminum mount. These can be seen below in Figure 4.

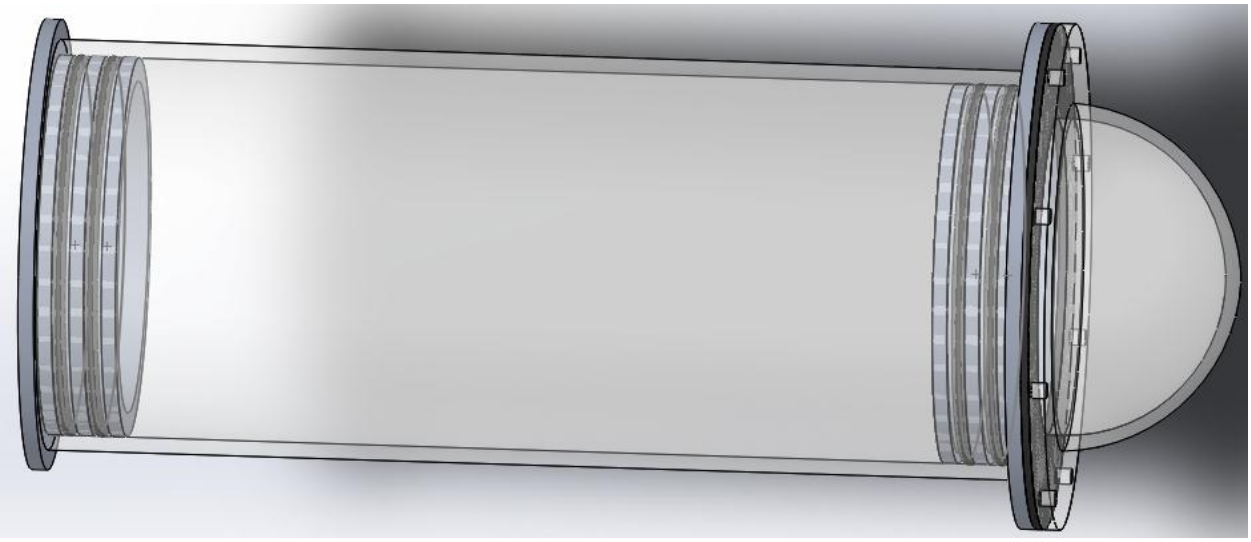


Figure 4 - screenshot of only O-rings, gasket, front and read aluminum tube enclosures.

This design has been a huge success for the ROV team this year. There is yet to be an instance where water has entered the tube, in fact the vessel is so air tight that air must be leaked out of the interior so that the inside pressure of the tube won't push the end cap out of the tube.

Another large priority of the design was modularity. This year's competition features an array of different tasks to be completed in 3 different pools at the testing facility. In each of the pools, teams are allotted 15 minutes to complete all of the mission tasks at that stage. It was discovered early on that the design may want to include more than one mounting and grabbing apparatus in order to be successful with all of the missions. Because of this, two manipulators have been designed; one is robotic and controlled by the driver of the ROV, while the other manipulator is static and simply used to pick things up off the sea floor and deploy heavy sensors using translational motion of the ROV.

Modeling

Chassis and Frame

To begin modeling this year's iteration of the ROV, three dimensional computer aided design (3D CAD) software was used. At UNH, students are provided with an all-in one CAD software package called SolidWorks. SolidWorks allows for CAD parts and assemblies to be designed and modeled. Additional SolidWorks also allows for drawings, animations, and simulations to be completed all within the same file regime. UNH ROV utilized this software extensively to complete the modeling, designing, and simulating for this project, due to its intuitive nature and well as its ease of access for UNH students.

Basic constraints derived from the MATE ROV Competition design requirements and Graduate Research requirements were used to develop the initial design for the ROV. First and foremost, a neutrally buoyant frame is the best configuration for maneuverability to achieve the maximum propulsive force from the thrusters. However, due to the small possibility of multiple failure modes within the ROV, such as a water leak or loss of communication from the driver station, the frame is designed to be slightly positively buoyant when fully equipped. If for some reason the ROV stopped functioning while under deep sea conditions, the positive buoyancy would bring the ROV up to the surface for easy recovery. The

magnitude of the positive buoyancy is very slight, and close to neutral in order to retain the maneuverability that is demanded.

Another general constraint for any ROV, is that in order to maximize the resulting velocity from a given propulsive force, the ROV must displace the least amount of water possible along its movement. Thus there should be an absolute minimum of surface area that obstructs water flow through the ROV, yet retain structural integrity to withstand the propulsive forces without any deformation. A smaller frame that still has minimal solid obstructions is ideal, yet still places the thrusters far enough apart to ensure maneuverability in the respective degree of freedom. One of the constraints from the MATE competition is that the ROV must be able to fit through a 75 cm by 75 cm hole, and this year's ROV's dimensions are 30.48 x 46.99 x 50.8 cm.

Symmetry of the ROV is also necessitated due to cost effectiveness and ease of machining the components, as well as keeping the ROV's movement intuitive to the symmetric human interface. Furthermore, symmetry guarantees that the water displacement during movement is equally separated across the ROV and there is no maneuverability loss between opposite directions on the same degree of freedom.

In order to minimize the surface area attributed to buoy attachments that keep the frame slightly positively buoyant, the frame must be constructed out of the least dense materials without being overly costly. Therefore Aluminum 6061 was chosen for the endcaps of the electronics tube, the thruster mounts, as well as for the U-channels that make up the cross-member supports. The side panels are made out of polycarbonate and cutouts are machined in the side panels to minimize the obstructing surface area during translational movement. Finally, the single electronics tube is made out of acrylic. All frame components were machined at UNH except for the endcaps which were machined by Eastern Precision Machining. This frame composition is strong enough to withstand the propulsive forces while being less dense than equivalent-cost alternatives, allowing a minimal amount of buoyancy additions to be required, which improves the overall agility of the ROV.

The final design for the frame was heavily influenced on being modular. Modularity of the chassis is incredibly important to the success of this project due to the variety of different sensors and manipulators required to complete all MATE mission tasks as well as housing the photodetector array for graduate research. For competition, the ROV must be able to support one robotic claw, one static manipulator, and a hook attachment. Both manipulators needed to be easily accessible during competition, since they will be swapped out between product demonstrations. Both appendages attached to the ROV at all times would inhibit the movement of the ROV as well as create more modes of failure.

Propulsion

Now that a basic frame for the ROV has been established, a propulsion system needs to be designed in order to make the ROV drive. A propulsion system allows the ROV to utilize underwater thrusters in a specific configuration to grant operational range of motion. The chassis team defined that the ROV requires 5 degrees of freedom to successfully complete all mission tasks during competition as well as the graduate research. In order to achieve this range of motion, a six thruster configuration was designed. Two thrusters were placed on the left and right side of the ROV, which grants translational motion in the Y-direction. It is important that these thrusters are placed specifically through the center of momentum, because if they were off center they would causes an unwanted rotation of the ROV

when only a Y-direction translation was desired. Two thrusters were placed on the back of the ROV which provides translational motion in the X-direction. These thrusters are placed equidistant from the center of momentum in the X-direction to prevent unwanted rotation again. The final two thrusters are placed on the top of ROV which provides the motion in the Z-direction. These thrusters are placed in such a way that they are along the X-axis. This is to properly allow or ROV to obtain pitch rotation by only firing the front thruster for pitch down, and firing only the back thruster for pitch up. In order to obtain yaw rotation code was written to operate either the left or right thruster on the back of the ROV to obtain the desired yaw rotation. Overall this 6 thruster configuration provided the 5 degrees of freedom needed to operate; X, Y, and Z translational motion, as well as pitch and yaw rotation.

For the actual thrusters that would be used on this year's iteration of the ROV the team had to decide between the Seabotix BTD-150 and the Blue Robotics T100 mainly due to the cost difference between the two thrusters. With the competition being in St. John's, Newfoundland, Canada, the team is looking to reallocate funds to travel as much as possible. The Seabotix BTD-150 (Figure 5) is a Brushed DC motor primarily used for submersible marine applications. It features a depth rating of 150 meters, require 19 VDC power, and a 4.25 max continuous amperage rating. It also can provide 6.4 lbf of peak max thrust. The geometry feature a long "stem" (6.29 inches in overall length) with mounting points and a housing for the propeller (3.7 in diameter). The cost of the Seabotix BTD-150 is $780, which adds up when an ROV requires 6 thrusters for precision movement. The total cost for the UNH ROV team to outfit an ROV with this type of thruster is $4680. This approximates to 56% of the average total cost to build.

Figure 5 – Seabotix BTD-150

Figure 6 – Blue Robotics T100

Enter the Blue Robotics T100 (Figure 6). The T100 also is used in submersible marine applications, but features the M100 Brushless motor. The T100 does not have a pressure rating because the thruster doesn't have any encased air or oil. The components are sealed by a protective coating and the motor is sealed with an integrated propeller and nozzle assembly for a more compact design. The T100 also takes 12 VDC power and an 11.5 max continuous amperage rating. It does provide less peak max thrust to that of the BTD-150 but the unique compact design eliminates the need for a "stem," cutting the overall length to 4 inches and an equivalent diameter of 3.7 inches. The T100 costs $144, and when multiplied by a need for 6 thrusters the total cost comes to $864. The cost of this thruster is roughly 6.5 times less than that of the BTD-150, which equates to a cost difference of $3816 that can easily be put towards travel cost for the team.

The help determine which thruster would be more adequate for the ROV a test was conducted as part of the teams senior lab class. Parameters defining thruster performance include data gathered from steady-state and transient responses; rise time, peak time, settling time, and thrust force per given percentage of voltage input. The experiment utilizes the NI-DAQ digital SoftScope and DAQ interface to measure the voltage output of an Interface SM-25 S-Type Load cell to determine the force applied to a rigid vertical member by the thruster in a 20 gallon fish tank.

The Steady States of both the Seabotix BTD-150 and the Blue Robotics T100 were measured by recording the input voltage given to each motor and comparing that to the output reading on the strain gauge. One problem arises when comparing the input voltage of the motors with respect to the other motor. The BTD-150 takes a maximum input voltage of 19.1 volts, while the T100 only takes a maximum of 12 volts. Ideally, a comparison of the two motors, given the same voltage input, would be used to determine which one provided the most thrust output per voltage input. However, this would give an unfair advantage to the motor depending on which max voltage input was used. For Instance, if an experiment was conducted reading output thrusts given an input voltage from 1-12 would be an unfair advantage for the T100 as it reaches its maximum thrust capacity, while the BTD-150 would only reach 63.14% of its maximum capacity. This would be an important measurement if only a certain amount of voltage could be utilized to drive the ROV. UNH ROVs MATE competition specifics that a standard voltage of 48 volts will be provided from the surface, which counteracts the need to prioritize voltage uses, as 48 volts is plenty to drive both thrusters.

To accurately compare both motors together, a normalization of the input voltage was used. This was done by changing the caparison from voltage input to the motor, to percent voltage input to the motor. For the BTD-150 the 19.1 maximum voltage was divided evenly into percent of inputs. Therefore 10% voltage input for the BTD-150 actually produced a 1.91 V voltage input, while the 10% input for the T100 corresponded to a 1.2 V voltage input. This normalization allows UNH ROV to compare both motors together without giving one an advantage over the other.

Originally the experiment was designed to be conducted in the Jere A. Chase Ocean Engineering building, in order to utilize the full scale engineering tank, however this option did not pan out. Instead the solution of a portable 20 gallon fish tank was implemented. The small testing platform caused some limitations in terms of the testable range of the thrusters. The limited volume of water caused the thrusters to begin displacing too much water at a thrust of around 50% of their respective thrust capacities. Thrusts corresponding to input voltages greater than 50% caused a vortex to form between the water surface and the inlet of the thruster propeller, triggered by a large displacement of water. To avoid introducing disturbances in the readings of the force transducer, the input voltage for the experiment was limited to a range between -40% and 40%. A performance relationship can be produced for each motor by directly comparing the ratio of the output thrust and the percent voltage input. Although the range of input voltages was limited due to the experimental setup, it is likely the thrusters will not be used to their full potential on the ROV, in order to allow for precise movements.

Both forward and backwards thrust for the Seabotix BTD-150 and Blue Robotics T100 thrusters were produced and are graphically represented in Figure 7 and Figure 8 respectively.

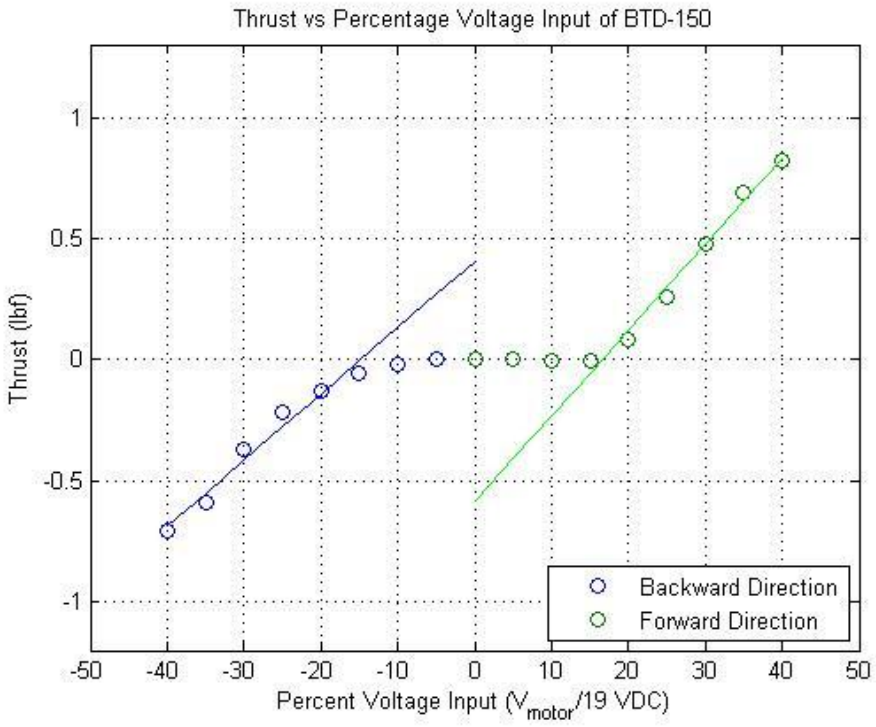


Figure 7: Output Thrust vs Percent Voltage Input of Seabotix BTD-150

Figure 7 shows the thrust output of the Seabotix BTD-150 thruster. The blue region of the graph represents the backward thrust of the motor, and the green region of the graph represents the forward thrust of the motor. It is important to note that both the -5% and the 5% input voltage readings returned a thrust output of 0 lbf due to a "dead zone" programmed into the Arduino code. The programmed 'dead zone' is necessary in order to reduce accidental excitation of the thrusters on the ROV (controlled using an Xbox controller joystick which is never at a true neutral position).

The linear representation of the collected output thrust and percent voltage input data is determined in order to compare the thrusters using the performance ratio. The forward and backward thrust performance lines have slopes of 0.0352 lbf/%V_{in} and 0.0274 lbf/%V_{in} respectively. The recorded forward and backward thrusts were fairly similar in magnitude, although the performance slope suggests that the thrust in the forward direction reacts better to increased or decreased input voltages. It was found that there is a 22.16% difference between the performance of the forward and backward thrust capabilities of the Seabotix BTD-150 thruster.

Figure 8 represents the plotted data of the thrust output for the Blue Robotics T100 thruster. The same methodology was applied to the T100 as the BTD-150. The forward and backward thrust performance lines have slopes of 0.0429 lbf/%V_{in} and 0.0366 lbf/%V_{in} respectively. Similarly with the BTD-150 thruster, the T100 thruster also exhibits a better forward thrust performance. The difference in performance between forward and backward thrust capabilities was determined to be 14.69%. It was concluded from the results of the experiment that the Blue Robotics T100 thruster's output thrust magnitude was larger than the Seabotix BTD-150 thruster per percent voltage input. The differences

between the thrust output per given percent voltage input for each thruster were calculated to be 33.57% for the backward thrust and 21.88% for the forward thrust.

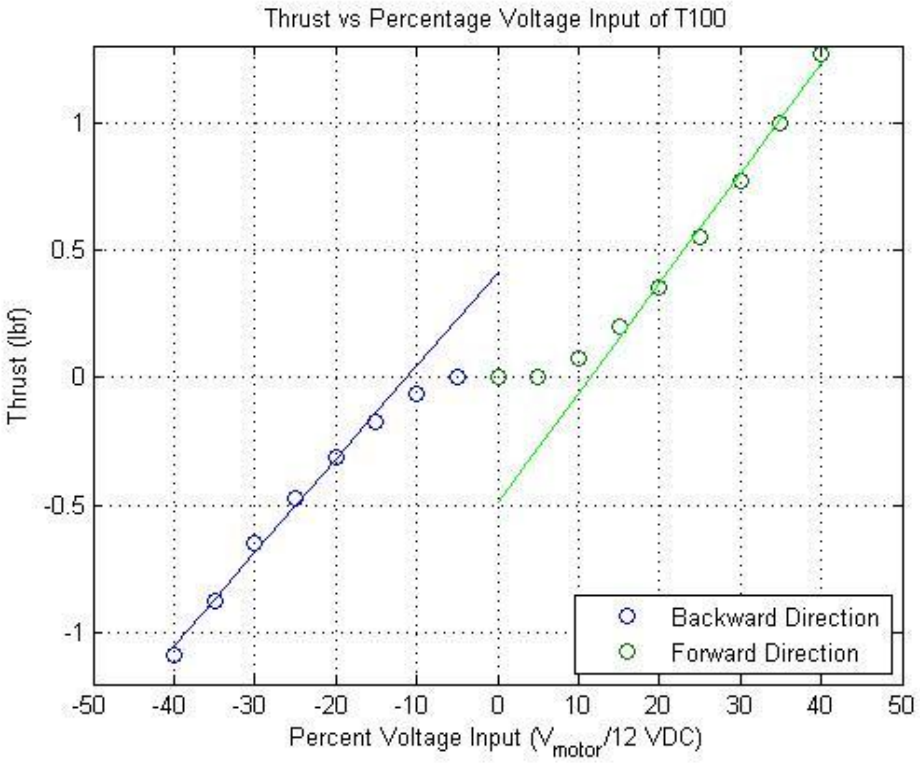


Figure 8: Output Thrust vs Percent Voltage Input of Blue Robotics T100

Transient responses were performed in order to poll data from the dynamic systems. Both of the motors were modeled as second order systems therefore, each motor will have characteristics of second order systems. In order to collect data from the thrusters to compare their responses, the thrusters were connected to the experimental setup used for the steady-state input recordings. In order to record the transient response of both thrusters, the output thrust was recorded with respect to time. The transient response for the T100 thruster was recorded using a step input from 20% thrust to 40% thrust. The BTD-150 thruster was recorded using a step input from 20% to 40% thrust as well.

This was achieved with the BTD-150 thruster by starting with the voltage at 20% of its potential power and then spiking it by turning the dial on the voltage supply provided in the lab. Once 20% voltage was stable in the tank, the dial was spun to a voltage nearest the 40% mark for the BTD-150 thruster. This step input into the system resulted in the following plot seen in figure 9 below.

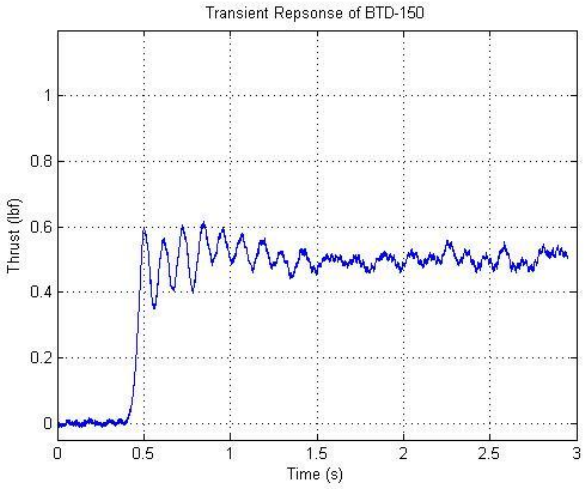


Figure 9: Transient response of the BTD-150 thruster

The step input of the T100 thruster was achieved by prompting the Arduino to control the thruster from 20% thrust to 40% thrust. Represented in figure 10, the T100 thruster generated a much larger overshoot than the BTD-150, suggesting the damping ratio of the BTD-150 thruster is slightly larger than the T100 thruster. The overshoot, although generally unwanted, aids in providing a quick response to reaching peak thrust.

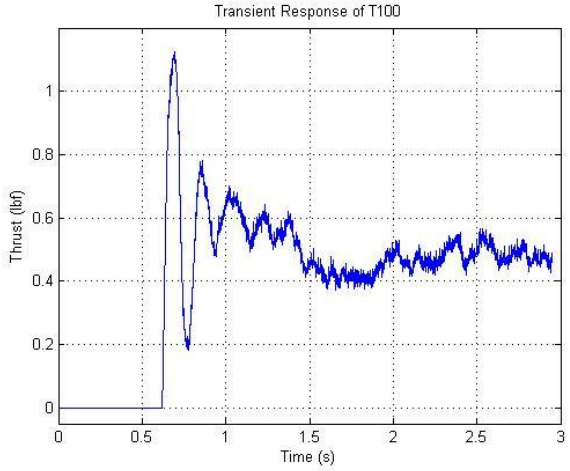


Figure 10: Transient response of the T100 thruster

These transient responses are very important statistics that relate to the build of the UNH ROV. This is because the rates of response are critical to determining which motor will perform better when trying to maneuver precisely. These fine movements will be necessary in many mission tasks at the MATE international ROV competition. Many factors play into the decision to use either the Blue Robotics T100 thruster or the Seabotix BTD-150 thruster including cost, thrust capabilities, performance with respect to voltage input, and percent overshoot given a step input. Because the thrusters will be changing 'speeds' constantly, it is important they do not overshoot their goal. The transient response data can be used to create a better control system for the ROV as well as allow the driver to react to known flaws in the propulsion system.

The rise time of the thrusters provides information about how each thruster initially reacts to a step input. ROV thrusters will be subject to quickly changing speeds and directions, therefore an understanding of how quickly they reach the maximum thrust at the given final voltage input will allow the driver to better react to the propulsion system, similarly to the overshoot. The settling time refers to the time at which the response settles to a value ranging within 2% of the steady-state value. This information is important when changing from one constant speed to another. By knowing the settling time of the thruster, it is possible to understand better how much thrust is being executed; therefore, a better estimation of the velocity of the ROV can be calculated.

It was found that the rise time of the BTD-150 thruster was .0944 seconds, while the rise time of the T100 thruster was .0209 seconds. A quicker rise time of the thruster will result in a quicker response of the robot, though it may result in overshoot. As seen in Figure 10, the T100 thruster has considerably more overshoot than the BTD-150 thruster. The T100 had a max percent overshoot of 90.23%. The BTD-150 also had a varying amount of overshoot. The percent overshoot seen with the BTD-150 was 26.45%. For UNH ROV it was necessary to decide whether or not the system would desire overshoot. It is thought that overshoot in this scenario would hinder the capabilities of stable driving. Therefore, in this case it is best to go with the BTD-150 which has less overshoot for more precise driving.

The Seabotix BTD-150 and Blue Robotics T100 thrusters both performed well throughout the experimentation. The performance of the thrusters were compared using a common ratio between the two; the relationship between percent input voltage and total output thrust. Based on this performance relationship, the T100 thruster performed better both forward and backward than the BTD-150 thruster by 17.95% and 33.57% respectively. It was found that the output thrust of the T100 thruster was higher than the BTD-150 thruster for identical percent input voltages, based on the recorded experimental data. This relationship is important for determining the capabilities of the thrusters in comparison to the provided datasheets from the respective companies. Because the output thrust per percent input voltage was found to be greater with the T100 thruster, it was determined that the thruster provided necessary output to be viable as a replacement for the BTD-150 thruster on the ROV propulsion system.

Systems Diagram

The UNH ROV team utilizes a similar systems structure for each ROV. This year's design goals place emphasis on human and robot interaction. To achieve this, more focus was placed on the Human Interface portion of the system. The rest of the system diagram follows much of the legacy ROV system designs, however it features new add-ons specific to the mission tasks of the 2015 MATE ROV Competition.

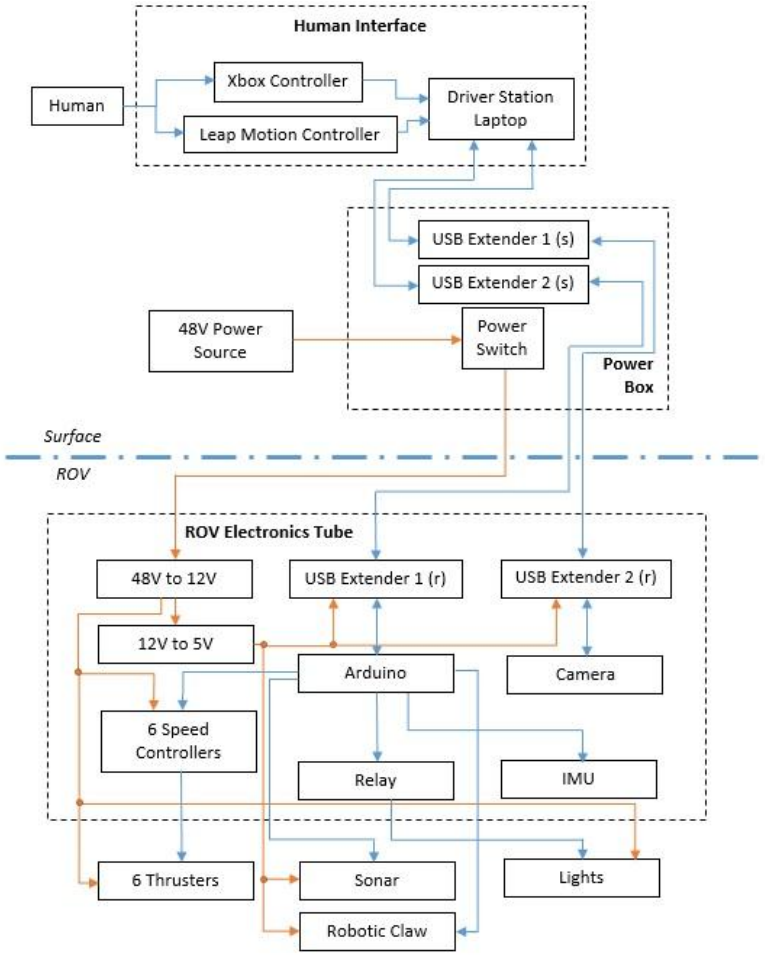


Figure 11 - System integration diagram depicting the control and power path from the surface through the ROV

The human, or Pilot, first interacts with the ROV through the Human Interface Devices (HID): the wired Xbox 360 controller, the Leap Motion controller and the Driver Station Laptop. Each of these HIDs offer a unique way to interact with the ROV system. The Xbox 360 controller is the human input control for the drive control of the ROV and control of the robotic manipulator. This controller exists at the top of the input devices, with respect to control hierarchy, giving it the ability to override any other input to the drive control and robotic manipulator. The Leap Motion controller allows the control of the robotic manipulator and precision drive control at low speeds by creating a digital representation of the pilot's hand to be mapped to servos and other motors. The Driver Station Laptop displays is where the HID software is located as well as the live camera feed from the ROV. It packages data gathered from the Xbox 360 controller and the Leap Motion controller and sends it down via serial communication to the ROV. The Driver Station also displays a Heads Up Display (HUD) that will be located to show the camera feed and the sensor data in real-time to the Pilot.

The data from the Driver Station Laptop is passed through USB cables to the USB extenders in the Power Box, which acts as an integrated connection box for power and data to interface with the surface electronics and the tether. The data and communications cable is kept separate from the camera cable in order to maximize the bandwidth for each line. The Power Box also features a power switch with a 40

ampere circuit breaker for safety and analog voltmeter and ammeter. Data traveling through the power box performs a handshaking interface between the control station and the onboard components of the robotic system. The constant interaction between the HIDs and the control system allows for a more stable connection and continual monitoring.

The power through the tether from the integrated connection box is stepped down to 12 volts followed by 5 volts as it enters the ROV's end cap. The propulsion system, powered by the 12 volt step down, is controlled by electronic speed controllers (ESC), which provide three phase power to the Blue Robotics T100 thrusters. The ESCs also provide logic for variable control of forward and backward thrust. Controlling the propulsion system as well as the onboard logic control for sensors and manipulators is the Arduino Mega, powered through the 5 volt step down. Housing the control system is the ROV electronics housing tube which provides a watertight space with waterproof connections between the external propulsion system, sensors, and manipulators, as well as the tethered link to the surface.

Mission Task Items

The ROV had to be designed with the mission tasks of the 2015 MATE ROV Competition in mind. General MATE competition rules require that the ROV be tethered to the surface and powered through the tether by a 48 V source on the surface. This years ROV has a 75' tether that carries power and driver input downstream to the ROV and carries video feedback and sensor data upstream. The tether is protected by a robust sheath and is connected to the back of the electronics tube through 9 waterproof SubConn connectors. Each connector has specific job for either transferring data or powering an electronics component, with one connector free for powering modular lights or sensors.

The first tank at competition has a "Science under the Ice" theme where the ROV has to navigate under a synthetic sheet of ice and collect samples from the bottom of the pool and from under the ice. For this mission the ROV needs to be less than 75 cm x 75 cm in order to fit through the allotted hole in the ice where the ROV will enter the water. The ROV is about 30.48 x 46.99 x 50.8 cm which is a significant reduction in size from the previous ROV, mainly due to more compact brushless thrusters and a single 6" electronics tube. The small size not only allows the robotic system to get through the hole but makes the ROV a very nimble vehicle which will simplify navigation through each mission task.

View of the ROV will be obstructed by the ice for the majority of the mission, so video feedback from the ROV is imperative. For competition the ROV will have a camera on a gimbal located in the front dome section of the electronics tube and send high definition video through the tether to a monitor. A modular underwater light will also be mounted in case the ice sheet blocks the light from underwater.

Another design feature that will help collect samples under the ice and manipulate a variety of features throughout the competition is the robotic manipulator. The robotic manipulator is controlled by a Leap Motion controller that uses a mapping of your hand to control the rotation of the manipulator and the pinching of the claw at the end of the manipulator, as well thruster output for small translational movements. Activated easily with the Xbox controller, the intuitive Leap Motion controller will help secure smaller features during mission tasks with precision.

A general design feature that will help throughout the mission tasks is the large frame of the ROV. With smaller thrusters, there is an abundance of space on the frame to attach modular sensor mounts and static manipulators specialized for specific mission tasks. To qualify for the competition, the ROV must successfully insert a T-shaped hot stab sensor into a mock wellhead. A modular static manipulator was

designed for carrying the sensor at the correct angle to insert into the wellhead, releasing the sensor while in the wellhead, and smoothly securing and removing the sensor before returning to the surface. Modular manipulators like this one can be designed for specific mission tasks if the robotic manipulator cannot support the load of a certain feature or cannot be adapted to manipulate certain things. The ROV will have to carry loads of up to six pounds during competition, and some loads will be too bulky for the robotic manipulator.

The large frame will also eventually carry a multitude of sensors. The first tank involves a mission task where the ROV must measure the dimensions of a mock iceberg. To do this a sonar sensor will be attached that will triangulate the distance between the ROV and the iceberg. During testing the sonar provided 10 cm accuracy in a range between 20 cm and 25 m. This allows the measurement of the iceberg from below the surface based on the video feedback matched with the distance the ROV is from the iceberg. The frame is also sufficient to carry a photodiode array for graduate research. Sensor mounts can be designed as needed and easily mounted to the frame.

Simulations

Stress Analysis on Side Panels

The side panels are a very important component to the ROV. The side panel serves as are the vertical supports, and protect the electronics tube in the middle. The ROV is designed to sit nicely on top of a table while being operated on. The side support act as skids that allow the ROV to be raised off of the surface in which it sits in order to separate from any dangerous objects underneath it that could scratch or crack the electronics tube. Another purpose of the side panel is to protect the vital electronics tube. The tube houses all of the electronics on board the ROV. These electronics must never get wet. It is in the best interest to keep the center of the ROV forever out of harm's way. A third and arguably most important purpose that the side panel supports serve is to act as a mounting surface for the thrusters.

In order to provide support for the mounting of the thrusters the side support panels have been built to have adequate surface space to mount the thruster mounting brackets onto the sides. In order to build a maneuverable ROV, the UNH ROV team requires 6 fixed thrusters where 2 thrusters will be mounted to thrust in each of the three Cartesian coordinate axes. The mounting surfaces for the z and y-directions will be on the side panels. Because of this, complex loading will be placed on each of the side panels, so it is important to understand how to design for these forces (Figure 12).

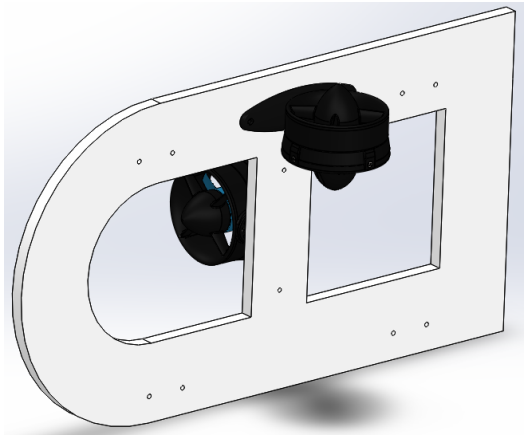


Figure 12 - Side panel with mounted Blue Robotics T100 thrusters

First the thruster oriented in the z-direction is analyzed and axial failure in the z-direction was calculated, which is identified as yielding in the center member.

$$\sigma = \frac{F}{A} \qquad \text{Equation 1}$$

Where F is the force applied by the thruster and A is the cross-sectional area of the center member. The maximum force of the thruster, 23.13 Newtons, exerts an axial stress of 2390 Pa on the center member. For each calculation, a factor of safety was calculated. The material used for the side panel is polycarbonate, which has a yield strength of 45 MPA, which results in a factor of safety of 18828.

Tear out was then calculated as well.

$$\tau = \frac{F}{A} = \frac{F}{4td} \qquad \text{Equation 2}$$

Where A is the tear area, t is the thickness of the center member, and d is the distance between the edges of the mounting screw holes to the edge of the panel. The tear out stress was calculated to be 15321 Pa, which corresponds to a factor of safety of 2937.

Torsional loading was also calculated.

$$\tau_r = \frac{T_r}{\alpha wt} \qquad \text{Equation 3}$$

Where T_r is the torsional loading, w is the width of the center member, and t is the thickness of the center member. The torsional load calculated was 4492 Pa with a factor of safety o 10018.

Next, the second thruster was analyzed. Axial failure in the x-direction was calculated (Equation 1). The axial stress exerted by the thruster was calculated to be 1195 Pa with a factor of safety of 37600. Tear out was also calculated (Equation 2) and found to be 15322 Pa with a factor of safety of 2937. Torsional loading is the last calculation performed on this thruster (Equation 3) and was found to be 4492 Pa which corresponds to a factor of safety of 10018. The factors of safety are very high, which relates to the high rigidity of polycarbonate.

Simulations were performed in SolidWorks to compare theoretical values to values approximated by the finite element software. Because the Blue Robotics T100 thruster was downloaded from the Blue Robotics website, it was unknown how the model would react to a complex geometry. Unfortunately, the model was a “compressed” assembly file, in the form of a STEP file and SolidWorks presented problem with interfering contact sets. Because of this, the mesh could not be calculated, thus a simulation could not be performed using the T100 model.

The solution to this problem was to create an object of similar size and mass and apply it to the thruster mounting plate shown in each of the simulation figures. For each thruster, von Mises, and the stresses in the critical directions were simulated. Convergence was also achieved for both thrusters as well.

Below are the results for the thruster oriented in the vertical direction.

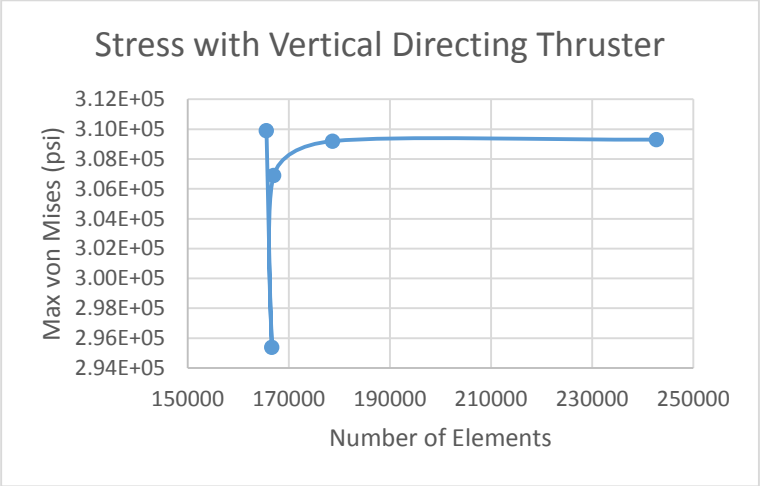


Figure 13 – Convergence Plot of Maximum von Mises Stress

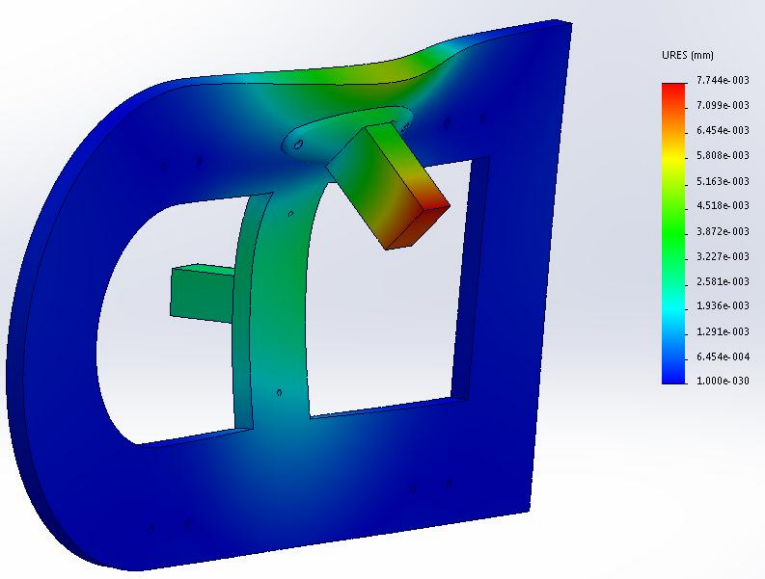


Figure 14 – Strain in millimeters

Below are the results for the thruster oriented in the horizontal direction.

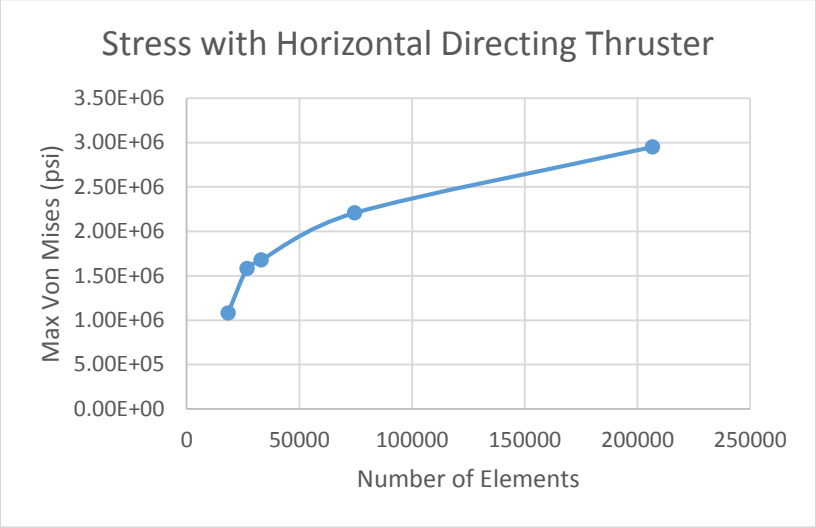


Figure 15 – Convergence Plot of Maximum von Mises Stress

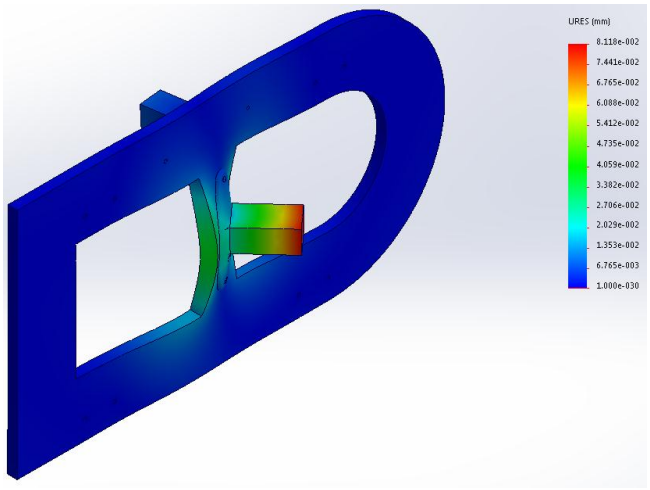


Figure 16 – Displacement in millimeters

After looking at all of the analysis of the experiment it was found that the side panel supports would not be affected by the torques generated by the thrusters. This can be concluded because the safety factors on all of the failure modes are incredibly high. If it was deemed necessary to reduce the weight of the ROV, the chassis frame could be crafted of .5” polycarbonate to .25” polycarbonate. Because of this, the design remains with the more robust .5” polycarbonate side panel supports even though the data shows that this much material is considerable over engineering.

Flow Analysis on Frame

From the detailed MATE manual it was determined that the ROV would need to maneuver through water with a max velocity of 1mph (17.6 in/s). Taking into account these parameters and the costs of the different thrusters a decision needs to be made for the final design for the ROV.

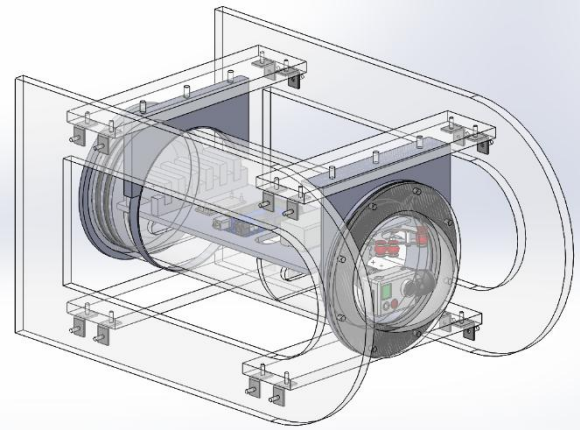


Figure 17 shows the design for this year's ROV in the state it was in November 2014 without the thrusters configured in. It consists of a single electronics tube which houses all of the necessary components to drive and operate the ROV. The electronics tube is suspended from the chassis frame, which acts as both a mounting surface as well a protective shell. The chassis design is important to protect the expensive "brain" of the ROV and thus needs to be a complete protective frame around the electronics tube.

Figure 17: Current UNH ROV Design [4]

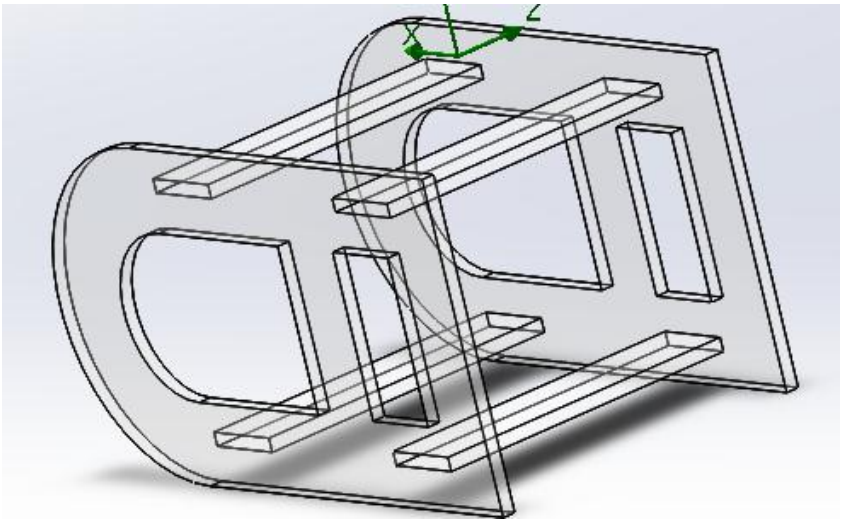


Figure 18: SolidWorks representation of the simplified model [4]

Using SolidWorks, the ROV design represented in figure 17 was simplified and remodeled, as seen in figure 18. Due to time and capabilities of calculations, the model was simplified to just the frame of the ROV. The model was simplified in order to compare the SolidWorks Flow Simulation calculated drag force which were compared with hand calculations. It would be nearly impossible to determine the drag coefficient while incorporating components such as bolts and screws. The tubular mid-section was also excluded as the focus of this study was to evaluate the flow through the frame of the chassis. Another study was conducted to evaluate the fluid disruption caused by the tubular mid-section.

The frame of the model was constructed out of acrylic (Medium-High impact) plastic for the four cross member supports and two side plates. The models dimensions are shown in figure 19. The model was built as one part instead of an assembly. Doing this allows for the simulation to take the model as a whole body as will the hand calculations.

The frame design was built in order to encase the electronics tube. Cavities were added to the frame to allow better flow in sideways motions. The fame is designed to protect the important electronics from colliding with objects, but also allowing it to traverse effectively under water.

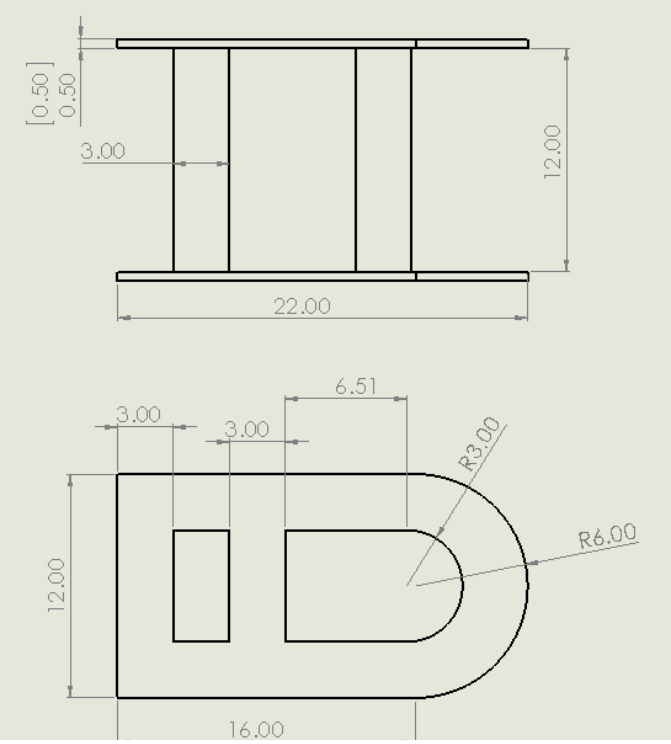


Figure 19: Schematic of model [4]

Using SolidWorks, flow simulations were conducted to determine the drag force acting on the model in a sideways motion (Flow in the direction normal to the side plates). The unit system defined in the simulation is IPS (in-lb-s). The simulation is an external analysis with closed cavities and without any physical properties. Liquid water was chosen as the fluid because the ROV will be driven in aquatic conditions. The roughness of the model will be considered zero, as most plastics are very smooth and therefor negligible. It was determined that the operation depth would be at 10ft with a pressure of 19.1396 lbf/in^2. The ROV will also be operating in a pool with a temperature of 65°f. The velocity is acting in the Z direction into the face of the plate. Different velocities were input for a series of trials to determine the force acting on the model, seen below in figures 20 and 21.

A plot showing the simulation flow trajectories is shown in figure 20. The max tested velocity interacts with the frame at 22.0 in/sec. Once the flow contacts the side plate the simulated flow of water is mainly dispersed around the outer perimeter of the frame. It can be seen that water will pass through the interior of the frame. Note there is a decrease in velocity and flow gets trapped between the two plates.

The velocity that interacts with the first side plate was needed to determine the maximum force acting on the frame. This simulation plot was conducted for each velocity to analyze the optimum velocity and

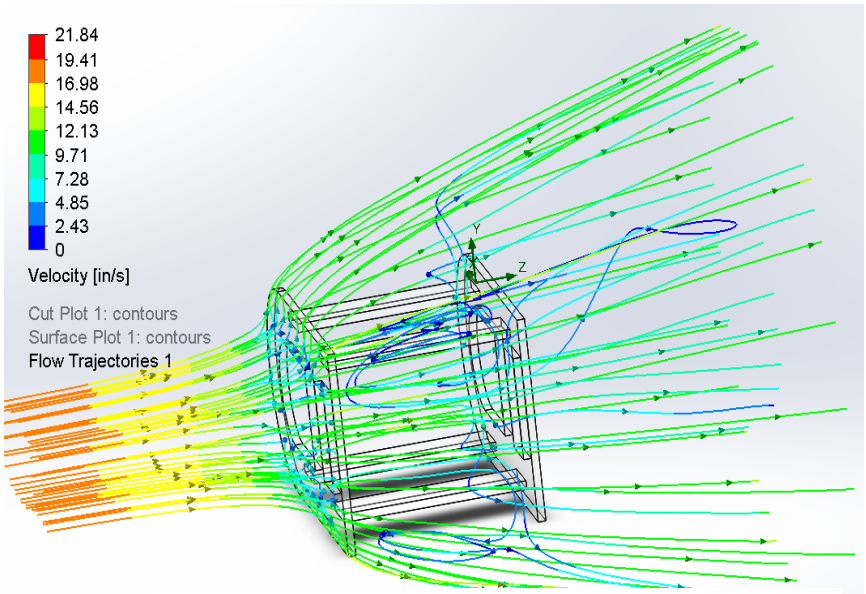


Figure 20: Flow simulation trajectories for a velocity of 22.0 in/s. [4]

force to operate efficiently.

A cut plot was also used to visually analyze the velocities acting on the frame. The velocity acting on the midsection of the frame is presented in figure 21, with a top-view orientation.

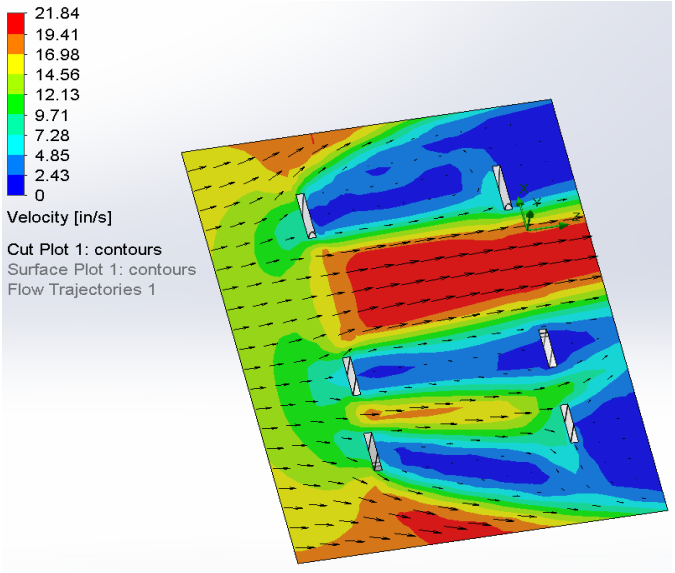


Figure 21: A top-view midsection velocity cut plot for a velocity of 22.0 in/s [4]

A global goal was set to determine the force in acting on the face of side plate. The Goal was calculated for each velocity and recorded in Table 2.

Table 2: Goal plot result [4]

Velocity (in/s)	Force [lbf]
4.4	0.355091389
8.8	1.438678674
13.2	3.292741521
17.6	6.995641254
22.0	9.225133453

In order to validate the calculations from SolidWorks simulation hand calculations were performed. The drag force acting on an object can be defined by equation (4):

$$F_d = \frac{1}{2}\rho v^2 C_d A \qquad \text{Equation 4}$$

Where,

F_d is the total drag force acting on the object
ρ is the density of fluid
v is the velocity of object relative to the fluid

C_d is the drag coefficient of the object
A is the projected area of the object that sees flow

For the ROV's purposes, the final drag force was the desired value. The density used was the density of water at 65 degrees due to MATE specifications. The velocity used was varied from 0.25-1.25 MPH (4.4-22.0 in/sec) to get a graph of the drag forces at various velocities, but most importantly reaching and exceeding the max required velocity as specified by the MATE competition. The drag coefficient was modeled as a rectangular box. This is because the configuration of the chassis can be simplified to a square box with various holes cut out which inhibit flow, giving the design a large coefficient of drag. The area was calculated using one of the side plates since this is the projected area seen by the flow. The drag forces were calculated and tabulated in Table 3.

Table 3: Hand calculated values of drag force.

Velocity (in/s)	Force [lbf]
4.4	0.331365
8.8	1.325315
13.2	2.98223
17.6	5.301765
22.0	8.283941

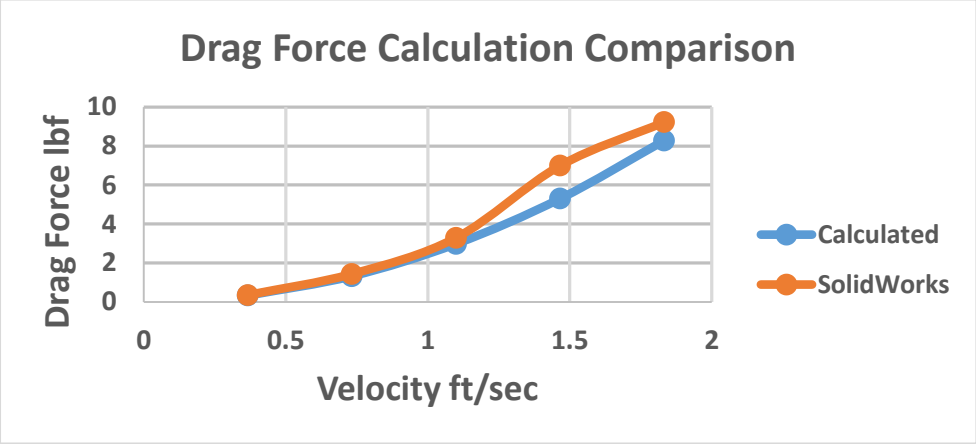


Figure 22: Graph Comparing the Calculated and Simulated Drag Force

Table 4: Table of Percent Errors between Hand Calculations and Simulations

Velocity (in/sec)	Hand Calc F_d (lbf)	SolidWorks F_d (lbf)	Percent Error
4.4	0.331364862	0.355091389	6.681808581
8.8	1.325314858	1.438678674	7.879717547
13.2	2.982229536	3.292741521	9.430196167
17.6	5.301765497	6.995641254	24.21330219
22	8.283940811	9.225133453	10.20248268

The MATE competition specifies that the ROV must reach a maximum speed of 17.6 in/sec (1 MPH). Hand calculations shows that the force on the side plates of the ROV, resulting from the 17.6 in/sec velocity would be 5.302 lbf. This value is expected to be a rough estimate due to the non-exact approximation of the drag coefficient. The results of the drag force from the SolidWorks Simulation resulted in a required thrust of 6.99 lbf. It must be noted that this value may not match perfectly with real life drag forces. This number is sure to be close though, as all of the assumptions included are fitting for the purposes of the operations of the ROV. Ultimately since two thrusters are oriented for each translational direction of motion, the Blue Robotics T100 thrusters are perfectly capable of handling these drag forces with a factor of safety nearing 2.

Prototype Testing

Testing Drive Code

Before the chassis of the ROV was machined and assembled, much of the code for controlling the ROV was developed and needed to be tested for effectiveness. Electronics were mounted to a prototyping

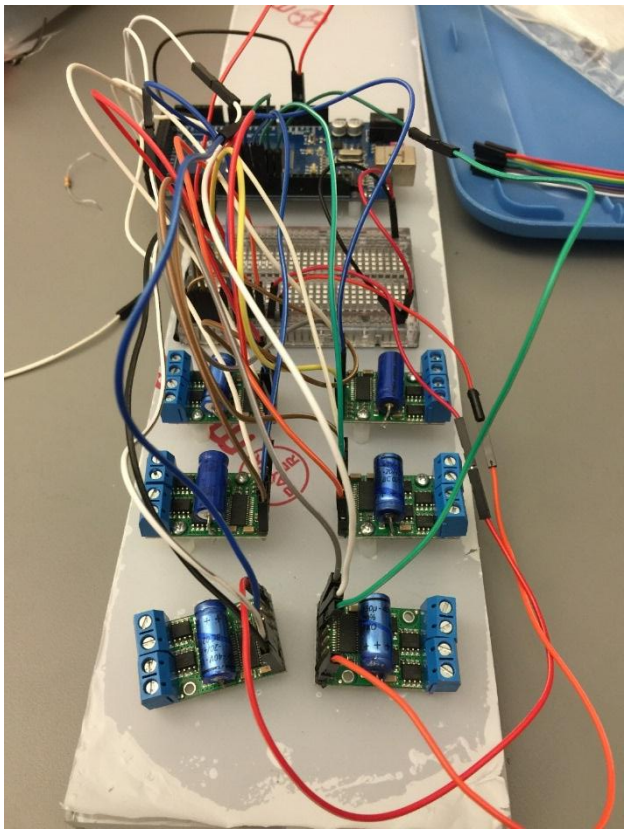


Figure 23 - ROV prototype board (v1)

board to experiment with layout for the actual electronics board and keep wiring organized. Figure (23) shows the prototype board during early stages of development with an Arduino and six speed controllers wired through a breadboard.

To test the thruster drive code, six LED lights were attached to the electronics system to represent thruster output. When the Xbox 360 controller received a rotational or translational input, the brightness of the LED's represented the fraction of full power output to the thrusters. This visual

representation of the control system allowed the team to see that the code was performing as expected.

Once the ROV built from last summer was complete, the team was able to use it as a platform to test the code now used for this year's ROV. Once a few minor logical adjustments needed to be made, the summer ROV was driving as expected.

Buoyancy Testing

Buoyancy tests were done several times throughout the later stages of the build to maintain level ballast and keep the vehicle only slightly positively buoyant. Because different modular manipulators and sensors are continually being taken on and off, weight distribution of the vehicle changes often. The team use a combination of weights and floats to keep the ROV level and at the right buoyancy during testing.

Full Electronics Testing

Before the ROV was put in the water a dry test of the full electronics setup and thrusters was performed. Figure (24) shows the final ROV electronics board and Blue Robotics T100 thrusters mounted to the prototyping board. Thrusters were run at low power because they were designed to use water as a lubricant and running them out of the water created excess vibrations.

Figure 24 - ROV prototype board (v2)

Before the electronics board was installed in the ROV, a waterproofing test of the electronics tube had to be done to avoid damage to the expensive hardware. With a soft neoprene gasket on the dome end and a double O-ring design on the back end, the testing was success down to 10 ft. on the first try. A final test was completed in the Chase Ocean Engineering tank at a maximum depth of 20 ft., which provided a depth rating adequate for the specifications provided by the MATE ROV Competition.

Controls

Drive control

The ROV is controlled from the surface using an XBOX 360 controller as well as the Leap Motion Controller to make driving intuitive. The digital values from the controller are read in to the driver station and are subsequently sent to the Arduino using serial communication. An Arduino script then utilizes the controller values to control different thrusters based on the desired movement. The left analog stick moves the ROV translationally, while the right analog stick controls pitch and yaw motions. The right and left triggers cause the ROV to ascend and descend, respectively.

Pressing the right bumper on the XBOX 360 controller initiates Leap Motion Control, which is used to control the external manipulator. The Leap Motion Controller maps a user's hand in 3D space using infrared light. Activating Leap Motion Control caps the thrusters to 5% thrust for fine-tuned movement of the ROV. The ROV control system utilizes the palm velocity function of the Leap Motion SDK to control each degree of freedom that the XBOX 360 controller can. Of course, the XBOX 360 controller is the master controller in the HID hierarchy and overrides any Leap Motion controller input as a safety mechanism.

GUI

Since the view of the ROV will be obstructed, the Driver Station is equipped with a Graphical User Interface (GUI) to assist the driver while maneuvering. The GUI is programmed in Java and acts as a Heads-up Display (HUD), containing different elements that alert the driver to the status of the ROV. A high definition camera feed is displayed in the background, using video data from the webcam in the dome of the ROV which is transferred through the tether. Overlaid on top of the camera feed is a table that displays values from the sensors and measurements in real-time. A cube is also displayed over the video stream to show the driver the orientation of the ROV and direction that it is currently traveling in. Graphics are also displayed to show the status of the dive light, as well as the manipulator. The GUI is also used to measure objects using different methods.

The video stream is the primary method of precise navigation for the driver and acts as a platform for the multiple image processing capabilities of the GUI. The GUI's image processing capabilities were developed using the Open Source Computer Vision Library (OpenCV), an open-source image processing library that has support for Java. OpenCV provides a function to obtain video data from webcam peripherals. After connecting to the desired camera, individual camera frames are obtained as matrices. The matrices are converted to a Java BufferedImage type so that the image is easy to manipulate. BufferedImages are represented using 3 bytes, using 1 byte for red, green, and blue colors. When each frame is converted to a BufferedImage, the GUI is updated so that the current frame is shown. The frames are sequentially shown at a rate of 30 fps to create the video. Depending on the size of the screen and the supported resolutions of the connected webcam, the resolution of the camera stream can be changed to provide video using 1280x720 pixels (720p) or 1920x1080 pixels (1080p).

Lag occurred in the GUI when video streaming was originally implemented due to the large amount of computations being done simultaneously. In order to prevent lag and allow for concurrency, multithreading was added. Instead of using the Java Thread class, the SwingWorker class was used because it handles cases that are specific to Swing, one of the Java libraries used to create user interfaces. The Swing library utilizes three types of threads in concurrent programs: initial threads, the event dispatcher thread, and worker threads. Swing events occur on the event dispatcher thread, so the

video capture and other processes that require their own threads are executed using worker threads. When worker threads finish, code is required to merge them with the event dispatcher thread, as well as thread cleanup. SwingWorker intrinsically executes cleanup to improve the clarity and simplicity of code. Each task that is responsible for a large amount of computations is run in its own worker thread in order to ensure that the GUI remains responsive.

The user interface is designed so that the video stream will be displayed in varying resolutions in the back of the frame used for the GUI. Various components and interactions occur on a transparent panel (called a glass pane) that is placed over the video. The first component that's placed on the glass pane is a semi-transparent table, which displays different measurements and sensor values for the robot.

The depth of the robot is the first value that's displayed in the table, which is obtained from the pressure transducer sensor. Since the values from the sensors are obtained through the C++ program that also runs on the Driver Station, Transmission Control Protocol (TCP) was implemented in the Java program to communicate with the C++ program. The Java program running the GUI acts as a TCP server, while the C++ program acts as a client. The C++ program sends the relevant sensor values as packets in a specific order, which the Java program receives and displays. Since the programs are executed on the same computer, the IP address remains constant at 127.0.0.1 (localhost), while the port can be changed.

User Datagram Protocol (UDP) was also considered as a means of communication between the programs, since data is continually sent while the ROV is on. While UDP continually sends information and is intrinsically simpler than TCP, UDP provides no guarantee that information will arrive correctly and sequentially. Since the GUI is the only way to accurately determine the status of the ROV, it's important to receive accurate information. TCP was ultimately used instead of UDP because of the drawbacks that UDP possessed.

In addition to the depth of the ROV, real-time object measurements are displayed in the table, including the distance to an object, the height of the object and the width of the object. The distance to the object is obtained by using a sonar which is connected to the ROV. The sonar sends out a pulse and outputs a voltage that corresponds to the distance. The sonar information is sent to the C++ program from the Arduino, so it's also sent to the GUI through TCP communication. One of the mission tasks for the MATE competition will be to measure underwater objects, which is why the height and the width of objects are placed in the table. Using the distance provided by the sonar, the GUI can measure objects in multiple ways. One method is for the driver to draw a bounding box around an object by dragging the mouse. Since the field of view of the camera is known, the pixel distance of the height and width of the bounding box is used in conjunction with the distance obtained from the sonar to calculate the dimensions of the object.

Components other than the table are located on the GUI to alert the driver to the status of the ROV. A cube was created using the Java binding for the OpenGL API (JOGL), a library for 2D and 3D graphics. The cube gives the current orientation of the ROV and the direction that it's going. Since the view of the ROV will be obstructed, the driver will have to rely on the camera to maneuver the ROV, relying on reference points and objects. If there are no reference points, the driver can utilize the orientation cube, the depth measurement and the dimensions of the pool to determine where to go.

Graphics also show the driver the status of the Leap Motion Controller and the dive lights. Using JOGL, a graphic was created that resembles the dynamic robotic manipulator that the driver can controller using

the Leap Controller. The graphic changes depending on whether the manipulator is open or closed, which tells the driver what needs to be done to grab an object. The C++ program controls the Leap Controller, so it continually sends a Boolean value to the GUI through TCP, which says whether the manipulator is open or closed, as well as an integer value reporting on the current rotation of the manipulator.

An image is also displayed on the GUI that resembles a light. The color of the graphic changes based on the current status of the dive light: yellow signifies that the light is on, while white signifies that the light is off. The dive lights are controlled by the XBOX 360 controller through the C++ program, so the status of the light will also be sent in a packet to the GUI through the TCP connection.

Ultimately, the graphical user interface is the primary connection that the driver has with the ROV. The GUI provides real-time status updates about the robot, including video; object measurements; sensor values; ROV orientation; and the statuses of the manipulator and the dive lights. Using Transmission Control Protocol allows the GUI to communicate with the C++ program that runs on the Driver Station to obtain pertinent information for calculations and measurements for the GUI's components. Concurrency was added to ensure that the GUI remains responsive. Computationally-intensive tasks such as video streaming and server communication are assigned their own threads to ensure that they don't collide with other tasks. The GUI is one of the several human interface devices that is used to help the drivers operate the ROV in an intuitive and easy way.

Robotic Manipulator Control

As part of UNH ROV's goal in bringing seamless integration of humans and robotic technologies, the Leap Motion controller was chosen as the controller to control to the robotic manipulator. It creates a three-dimensional representation of a human hand and maps specified values to the servos on the robotic manipulator. This creates a very fluid and intuitive interaction with between the human pilot and the ROV, thus allowing for a more natural method of remote object manipulation.

The Leap Motion controller is a small, commercially available product use for development of virtual reality technology. It uses 3 infrared (850 nm) LEDs and 2 cameras to shine an array of infrared dots and uses the cameras in the sensor to detect the three-dimensional position of each joint of the skeleton of a human hand. The sensing range of the device is limited to only 2 feet above the device, but boasts an impressive 200 Hz data refresh rate through USB 3.0 and a one-hundredth of a millimeter accuracy. The Leap SDK offers a wide range of tools for developers to use in many different programming languages, which is what the UNH ROV team took advantage of.

This year's robotic manipulator is the MKII Robotic Claw from DAGU. It features two degrees of movement that represent a wrist motion and a pinching motion. Each metal gear servo rotates 180°, which allows for the same freedom as seen with a human wrist and pinch. To prevent damage to the pinching servo, a spring–loaded clutch is included to allow the claw to close completely without the servo burning out.

The Leap Motion Sensor then is mapped using the corresponding joints in a human hand. Using the Leap SDK, the values for palm rotation about the center of the palm, while using the thumb as reference, are mapped directly to the servo values for the wrist movement of the robotic manipulator. Next, the Cartesian coordinates for the tip of the index finger and thumb are retrieved from the Leap SDK and the three-dimensional distance is calculated and mapped directly to the pinching servo of the robotic manipulator.

Figure 25 - Leap Motion controller

IMU Feedback for Mission Tasks

The Inertial Measurement Unit (IMU) and pressure transducer are integral parts of the ROV control system. These sensors are able to give critical information to the Pilot concerning the orientation of the ROV as well as its depth underwater. Feedback control is also important because it alleviates work that the Pilot must do to keep the ROV in a stationary position.

The MPU-6050 IMU measures 6 degrees of freedom utilizing both the MEMS accelerometer and MEMS gyro contained in a single chip on the GY-521 breakout board. It contains a 1024 byte FIFO buffer as well as a Digital Motion Processor (DMP) which acts as a small computer on board the breakout board to do fast calculations of the read outs from the MPU-6050 chip. The DMP is accessed by the ROV to read filtered acceleration and angular displacement values and use it in pitch feedback control.

The ROV requires pitch feedback control because during each mission task, the ROV may be picking up different items with varying weights, which the weights for ballast system cannot account for as the ROV is underwater and ballast to a specific load. In this case, the Pilot can activate an auto-leveling function on the Xbox 360 controller where the ROV uses Sliding Mode Control (SMC) to pitch the ROV so that the gyroscope vale for pitch reads 0°. This auto-leveling function remains active as long as the Pilot does not use additional pitch control, which is controlled with the up and down motions on the right analog stick. This overrides the auto-leveling function until the right analog stick is returned back to its neutral position, where the auto-leveling function takes over. This mode can be toggled on and off with a push of a face button.

The pressure transducer is used for depth feedback control of the ROV. This is necessary because the ROV is designed to be positively buoyant as an emergency recovery system in case communication to the ROV fails. Due to this, the Pilot would need to continuously be descending with the ROV by holding down the right trigger on the Xbox 360 controller. To alleviate the Pilot of this task, the pressure transducer sends a voltage corresponding to the pressure at which it is exposed to. Using hydrostatics,

the depth in water can be found and can be utilized by the control system. Again, Sliding Mode Control (SMC) is used to maintain constant depth while the Pilot does not have to ascend or descend triggers active. Once they become active, this mode is overridden by the Pilot's input, just like the pitch feedback control.

These control feedback methods are extremely helpful to the Pilot so that he can focus on orienting the ROV to complete mission task without worrying about these degrees of freedom. This limits the complexity of operating the ROV in critical, mission task situations.

Sonar Feedback

The Sonar system offers another place to implement intelligent controls as well as image processing. It is necessary to measure the size and therefore distance of objects underwater for certain mission tasks to receive 20% of the total points for mission tasks. The Sonar system works in conjunction with the camera and the HUD to give real-time size and distance data to the Pilot.

The Sonar system used on the ROV is the MaxBotix 7076 Sonar system. It is an all-in-one system acting as both the transmitter and receiver of the 42 kHz acoustic signal. It is designed for outdoor distance measuring in mal-weather, with an Ingress Protection Rating of IP67, but can easily adapted for underwater conditions by epoxying the leads and using a silicon based sealant around the transducer.

The ROV control system reads the voltage from the Sonar, which correspond to the distance an object is away. This calibration constant was found by placing objects in the water at given distances and measuring the slope of the line created by those data points. With the calibration constant in place, the Sonar sensor has a 10 cm resolution over a range of 25 meters in water.

To measure the size of objects underwater, the data from the Sonar is used in image processing performed by the Driver Station of the surface. The GUI receives human input by identifying two pixels at the ends of a line by clicking, dragging and drawing a line between two points of an object underwater on the video feed. The GUI takes these two pixel locations, the field of view of the camera, and the distance to the object and uses trigonometry to find the length of the line and the corresponding scaled size of the object underwater.

The primary limitation of this method is the distance that the Sonar returns to the size measuring algorithm. The Sonar must be perpendicular to the object it is trying to measure underwater to achieve the least amount of error in its measurement. Also, as the Pilot is drawing a line of an object to measure, the Pilot must orient the ROV to be perpendicular to measure the full length of the height or width of the object, instead of an auxiliary view of it. This limitation is only solvable through Pilot control. It is up to the Pilot to place the ROV in orientations to best measure underwater objects.

Final Design

The multitude of constraints provided for objective employment of the remotely operated vehicle provided a challenging engineering problem that involved a thorough design matrix. The design matrix was utilized for thorough integration of design constraints with a seamless control interface, a modular frame design, an efficient propulsion system and a cost effective strategy. The primary design constraints of limiting size, tethered power and data stream, visual guidance, and ability to manipulate objects led UNH ROV towards a fully modular system. The modularity of the frame and control system allowed for the mounting of various sensors and manipulator components to the exterior of the robotic system. Due to the immersive experience of a pilot station, it is necessary to provide the driver with as much positional and area awareness data. Therefore, a small and compact, modular and single camera design paired with sensors and manipulators as well as a robust HUD allow for a seamless pairing between human interaction and robotic control.

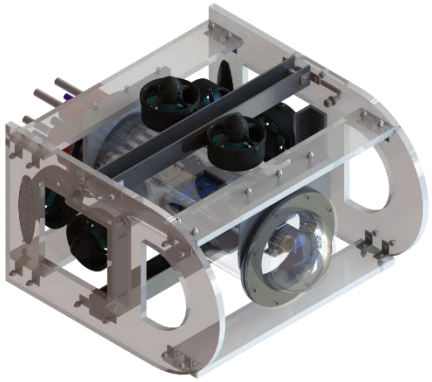


Figure 26 - SolidWorks representation of the final design

Team Finances

As stated previously, this year's expenses were initially predicted to total $16,672.84 for the build and competition. At first this was a shockingly large number, but as time went on the team worked out ways to reaching financial goals. In the middle of the fall semester the team came close to completing the model of what was planned on being built. Dimensioned drawings of modeled parts allowed for the determination of material quantities and sizes for ordering purposes. The more that was designed the more the price would fluctuate, some for better and some for the worse. In the end, the budget initially predicted turned out to be rather on point with what the team has expensed thus far, 8 months later.

Funding

Our finances were kept in check by a simple stocking system. All of the team's expenses were saved and catalogued in a binder. The expenses were then put into an Excel master file to track the teams balance throughout the year. The most challenging part of the financial cataloguing was outsourcing expenses to separate accounts. One of the team's advisor May-Win Thein, was happy and willing to fund a large majority of the build out of a grant that she had won. This grant was provided by Naval Engineering Education Consortium (NEEC) and Naval Sea Systems Command (NAVSEA). This organization is heavily invested in the graduate research in which UNH ROV folds into multiple UUV autonomous leader-follower control in which the follower UUV's follow a leader using optical pose detection. Because the

final design will be used as the platform in which all of the research equipment is mounted UNH ROV was allotted a budget from the NEEC grant to found the ROV build.

Before the funds from the NEEC grant were available, the team had to rely on other accounts for startup costs before big orders were necessary. Fortunately, for being such a large team in numbers, the team acquired adequate resources to cover expenses while the government funds were being transferred to the CEPS department which took until about late November. As a senior capstone project, the team was allotted an amount of money per person from the department in which each student is enrolled. Being that the UNH ROV team consists of eight mechanical engineers and one electrical and computer engineer, that is how the team was paid initially. The ME department valued each student at $200 a head. The ECE department on the other hand valued each student at $100 a head. This left the team with $1600 in an ME team account and $100 in an ECE team account. On top of the academic accounts the team was also granted money from Sea Grant. Sea Grant was gracious enough to donate $1000 to the team, expensed through a separate TECH 797 account. Because the team's budget was split between 5 different accounts, managing balances proved rather tricky.

Even though the team was gifted a lot of money, $2700 and NEEC funds, it was still evident that the team members would need to reach out to local companies and contacts to raise more money in order to cover the expenses of traveling to St. John's Newfoundland and competing in the international competition. The team treasurer took up the task of starting the team fundraising to get the ball rolling. In efforts to reach maximum support potential, a sponsorship package was created and sent out in numbers to companies in the field of ocean engineering as well as companies not related to the project at all. The sponsorship package was also leveraged into documentation to support filing for grants. This year's UNH ROV team applied for two grants. The first grant that the team applied for was the Parents Association Grant. The Parents Association declined UNH ROV's application and looked to other projects to provide funding. The next grant that was applied for was the CEPS Dean's Grant. This Application was submitted in the hopes to be granted a few thousand dollars and fortunately the team received just that. The Dean's Grant offered UNH ROV a comfortable $2000 to invest into the successes and performance of this year's ROV.

Expenses

The majority of the sponsorship packages were sent out over the winter break. UNH ROV reached out to over 160 companies for funding in hopes to cover the travel expenses to competition. The team anticipated a lot of refusals and therefor reached out to the masses to raise funds. In the end only 3 companies were gracious enough to grant the team funds. Collectively, Kepware Technologies, Edgetech and Green Pages granted UNH ROV a total of $2500. These funds were sure to be put towards good causes and being so have been sitting in the ME ROV accounts waiting until the moment in which the team will be purchasing flights to Canada.

Predicted vs Actual Costs

Table 5: The comparison of ROV predicted costs vs. ROV resulting expenses

Table 5 - Predicted vs. actual cost analysis

Category	Initial Predictions (Fall Semester)	Actual Expenses
Chassis	$2,222.84	$1,471
Controls	$3,580	$4,053
Tether	$270	$97
Travel	$10,600	~$6,285 (flights not exact)
miscellaneous	-	$119
Total	$16,672.84	$12,025

As seen in the table above, the team expenses resulted in a savings via cutting expected costs. Some of this resulted by cutting costs from the analysis completed on the BTD 150 thrusters vs. the Blue Robotics T100 thrusters. Most of the savings is soon to be a result in flight costs dropping from online predictions found in the fall. The ROV team has been tracking flights to attempt to book at the cheapest moment in order to result in the most savings.

Future Improvements

Water proofing technology for ROV has been significantly improved by the team this year compared to last year. Multiple water proofing tests were completed in the Chase Ocean Engineering Lab and all were successful. There was no obvious indication that the waterproofing was going to fail. However, there will always be some potential problem with the waterproofing, especially for the sealed part of the electronic tube. Some tiny gaps may exist on within those parts which cannot be seen with the naked eye. Therefore, some professional testing could be completed to predict the potential failure in order to prevent it. One useful and simple test is called 'helium leak test'. Helium is used to find small leaks or possibly larger leaks in bigger volumes. It is used as a tracer gas and its concentration is measured. In an ROV's case, a helium leak test could be implemented. In the meantime, the pressure sensor inside the tube can continuously keep monitoring the pressure. If the pressure keeps dropping inside of the tube, it is a clear indication that there is a gap somewhere at the surface of the tube. There are several reasons to choose helium as the leaking test gas. First of all, Helium is one of the smallest gas molecules and is inert, being inert it is relatively safe to use compared to hydrogen. In addition, it will not react with any of the materials within the part to be tested. In most helium leak testing applications, one uses a mass spectrometer tuned to detect helium although it is possible to use a residual gas analyzer. Helium leak testing can be generally be between one thousand and one million times more sensitive than using pressure decay techniques.

Another important test that can be conducted is a humidity test. It is very crucial to keep the inside chamber dry in order to make sure all wires and boards will be working in a dry and safe environment. Similar to the helium test, a sensor needs to be installed inside the tube and keep monitoring the humidity all the time.

Overall, the ROV is working appropriately this year. However, there are still operating issues in the control system as well as power management. First of all, the buoyancy and leveling properties of the ROV system are slightly unstable. Due to unbalanced weight distribution and tether management the ROV pitches slightly when ascending and descending. Second, physically looking at the ROV is not possible when operating it during deployment. Therefore, it may be necessary to consider multiple viewing angles.

Conclusions

The UNH Remotely Operated Vehicle Team is a student organization that serves as an outreach group for STEM projects, a senior capstone design project traveling to and competing in the MATE ROV Competition, and a platform for graduate level research involving optic based pose detection for swarm control of multiple UUVs. As a legacy engineering team, there is continued improvement and innovation over previous iterations of the underwater robotic system. UNH ROV is focused on the thorough execution of the learned engineering process for a complete integration of the design matrix, established from the design constraints proposed by the MATE ROV Competition and the future research potential the ROV will be used for.

The primary design goals of the robotic system were a modular and unified connection between the chassis and the control scheme as well as the intuitive interaction between the pilot and the control of the ROV. Through simulation analysis and careful design considerations based around the MATE ROV Competition's lifelike mission tasks an open frame design was constructed with various mounting points for sensors, static and dynamic manipulators, and scientific instrumentation. Using innovative collaboration between intuitive human behavior and adaptive robotic logic, the driver station was designed as an immersive experience with the ability to navigate without information overload. UNH ROV combined the applied engineering challenges with control theory and mechanical simulation analysis to design, manufacture, test, and compete with an underwater remotely operated vehicle.

Acknowledgements

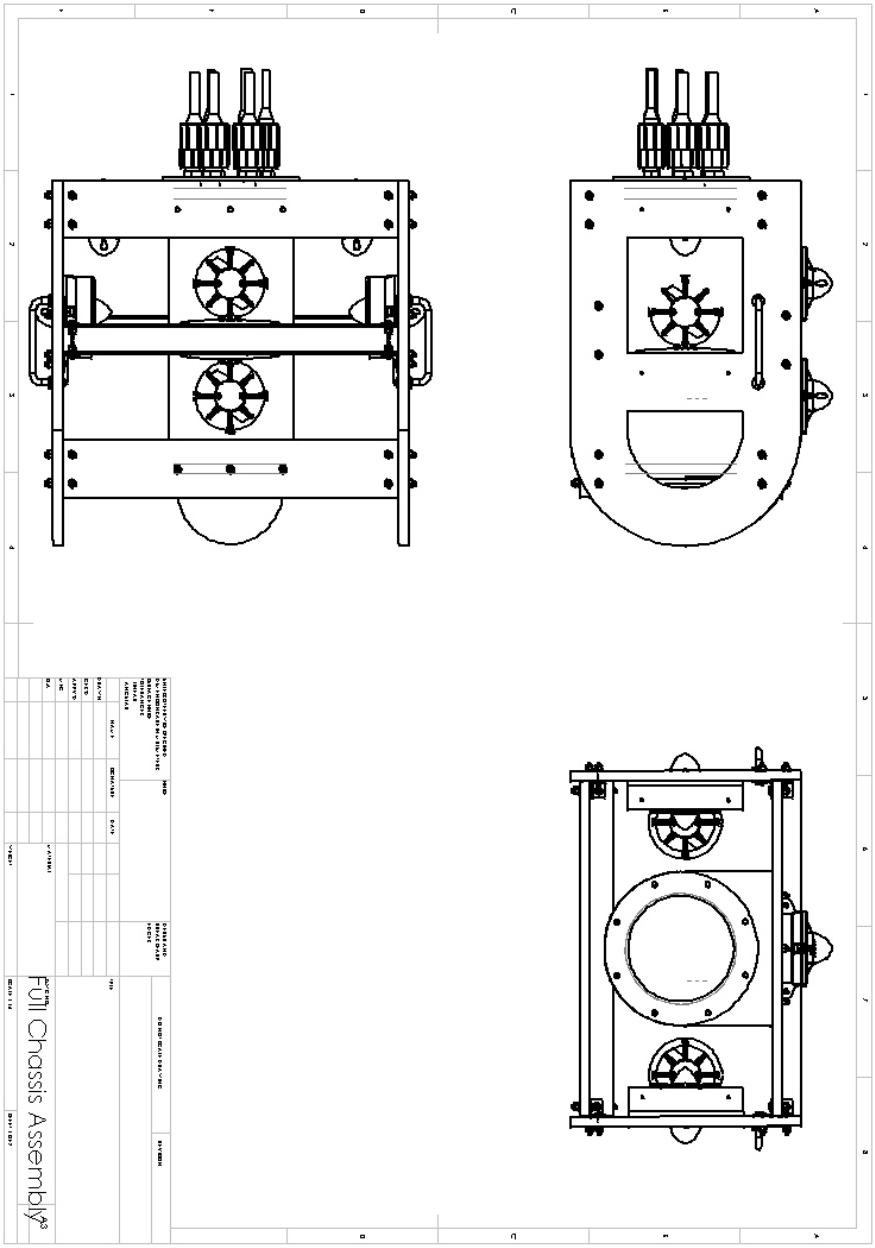
The ROV team would like to express our sincere gratitude towards the project advisors: Professor May-Win Thein, Professor Robinson Swift and graduate advisor: Firat Eren. The team would also like to thank Dr. Martin Renken, Tara Johnson, Scott Campbell, and Kevin Maurer as well as the great support from all fantastic donators. UNH ROV could not have had so much success without all of the generous support.

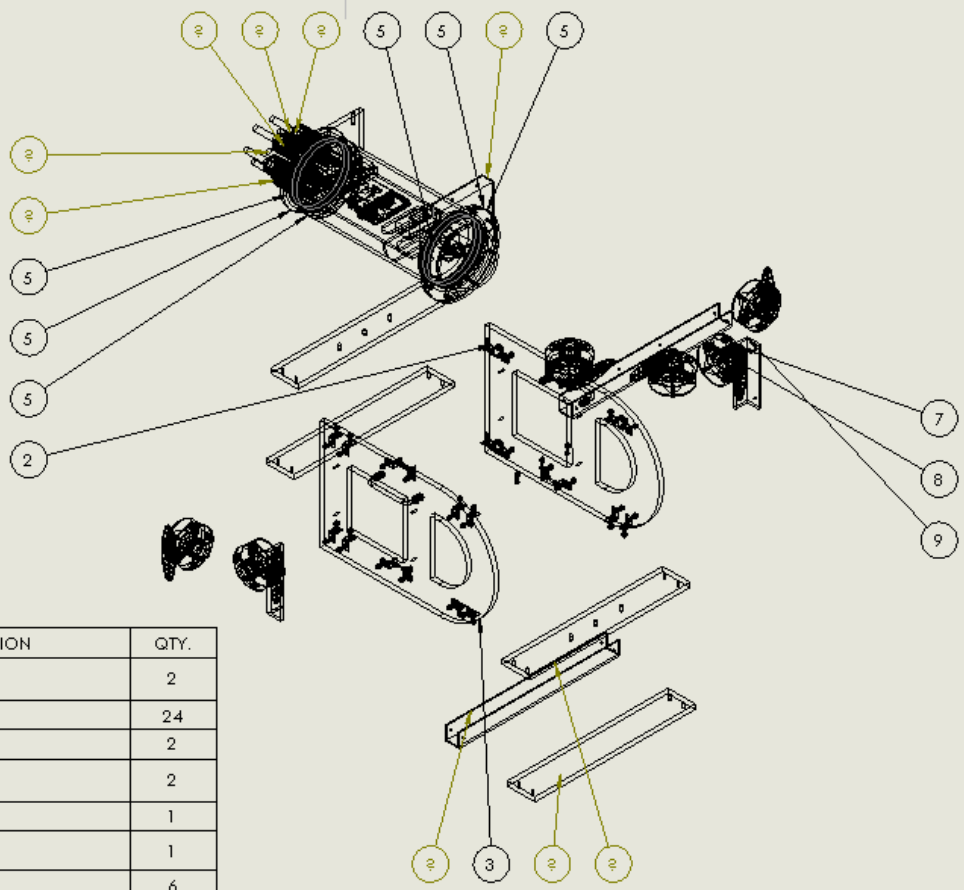
References

1. http://tqc.co.uk/leak-testing/leak-testing-guide-to-helium-leak-testing.php
2. http://en.wikipedia.org/wiki/Human_interface_device

Appendices

Appendix A – SolidWorks Modeled Drawings





ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	cross member supports		2
2	1556A 64		24
3	Side Support Member		2
4	cross member supports lower		2
5	Electronics tube2		1
6	aluminum cross member		1
7	T100-P-BRACKET-R1		6
8	T100-Thruster-R1		6
9	vert aluminum cross member		2
10	aluminum cross member_low		1
11	lexan assm screws		43
12	U chan assm screws		8
13	handle		2

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:

FINISH:

DEBUR AND BREAK SHARP EDGES

DO NOT SCALE DRAWING

REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APP'D			
MFG			
Q.A			

MATERIAL:

WEIGHT:

SCALE:1:16

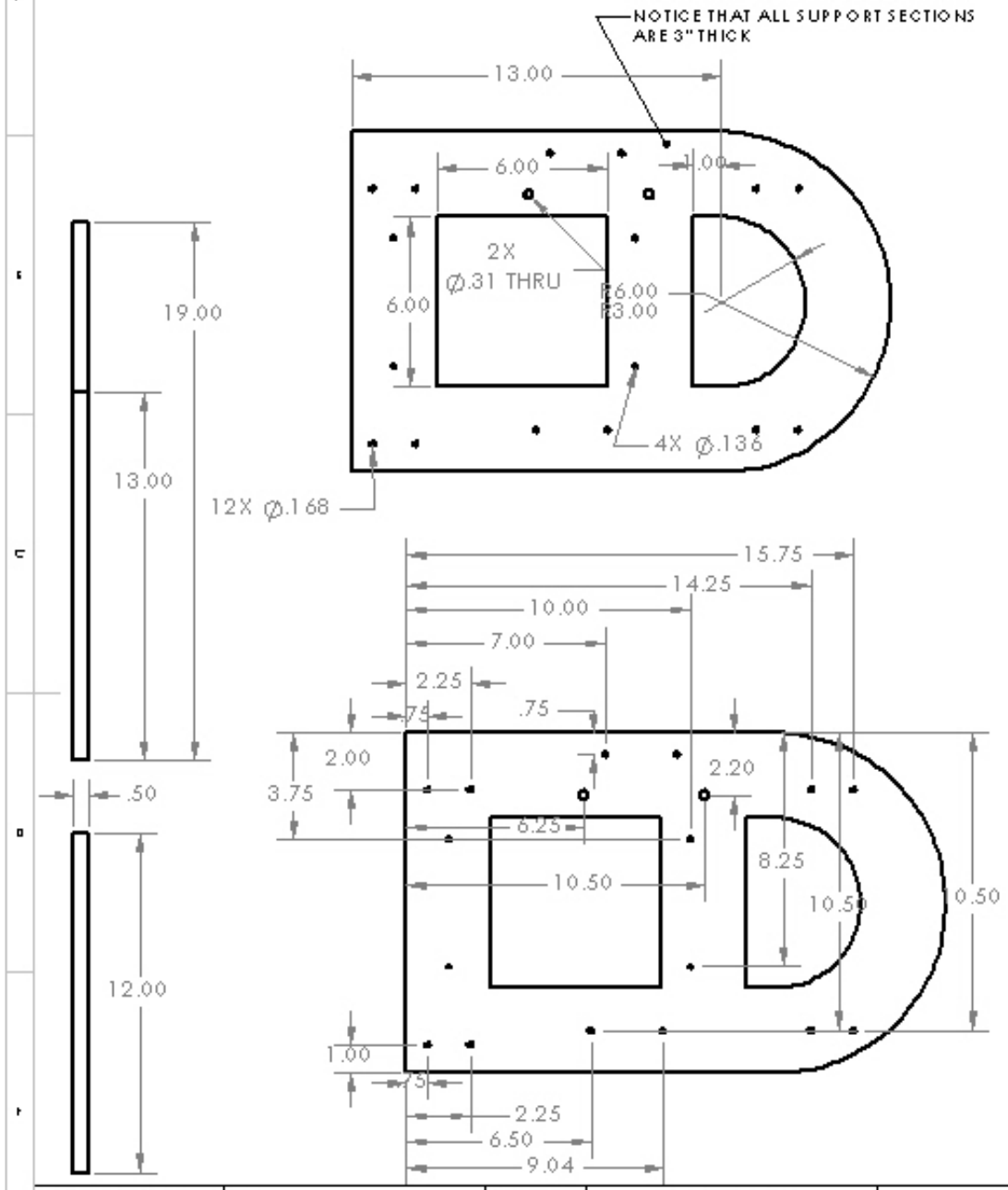
SHEET 2 OF 2

DWG NO.

A3

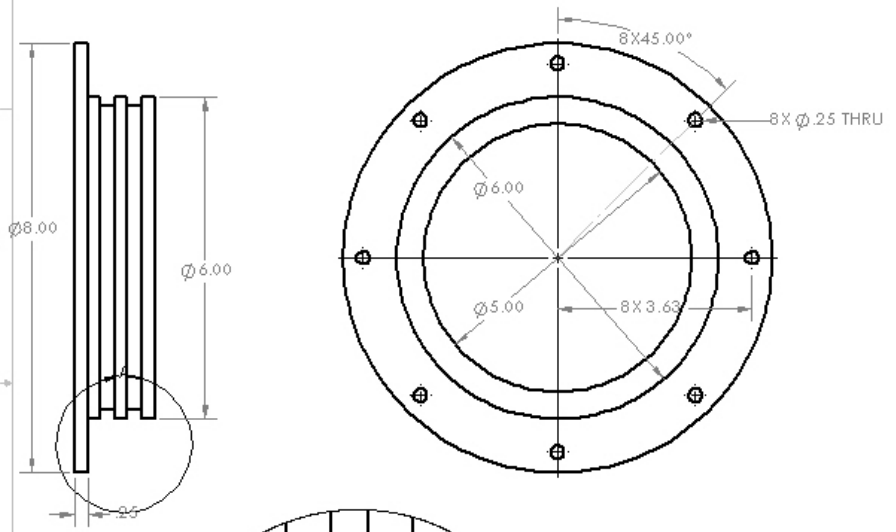
Full Chassis Assembly

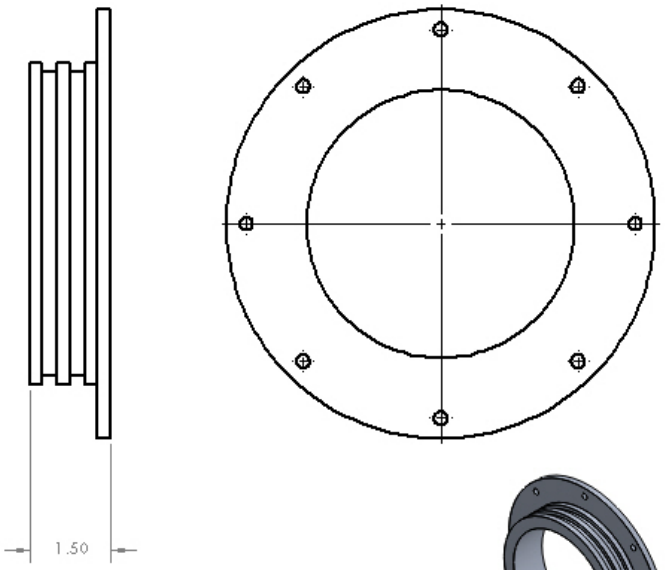
REVISIONS				
ZONE	REV.	DESCRIPTION	DATE	APPROVED
	A	INITIAL DRAWING GENERATION	1/16/2015	SAITEN

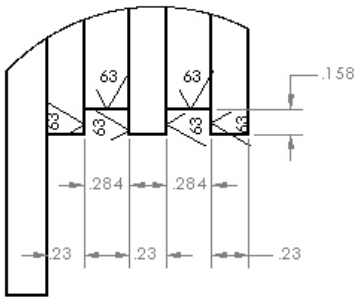


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR ANGULAR		FINISH:		DEBUR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION A
	NAME	SIGNATURE	DATE				TITLE:
DRAWN	SAITEN		1/16/2015				
CHKD							
APPV'D							
MFG							
Q.A				MATERIAL: POLYCARBONATE		DWG NO. Side Support Member	A4
				WEIGHT:		SCALE:1:1	SHEET 1 OF 1

REVISIONS				
ZONE	REV.	DESCRIPTION	DATE	APPROVED
	-	INITIAL DRAWING GENERATION	1/29/2015	S.ALLEN







DETAIL A
SCALE 1.5 : 1

UNLESS OTHERWISE SPECIFIED:		NAME / DATE
DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± .01 THREE PLACE DECIMAL ± .005	DRAWN	S.ALLEN 1/29/2015
	CHECKED	
	ENG APPR.	
	MFG APPR.	
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.	
MATERIAL ALUMINUM 6061	COMMENTS:	
FINISH		
NEXT ASSY / USED ON		
APPLICATION	DO NOT SCALE DRAWING	

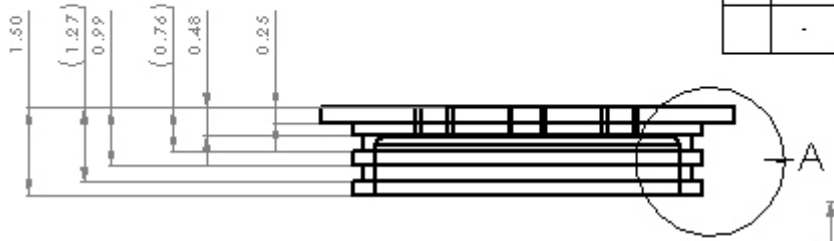
TITLE:

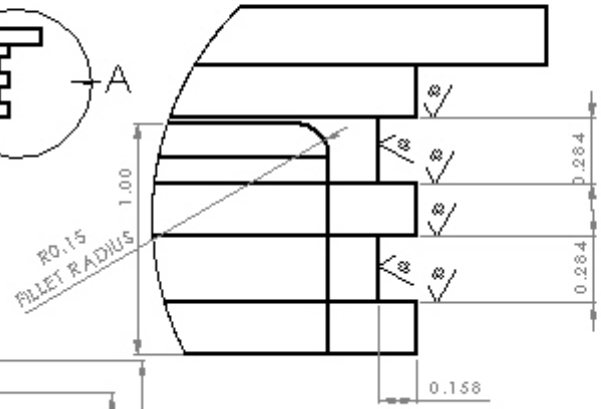
SIZE B | DWG. NO. | REV

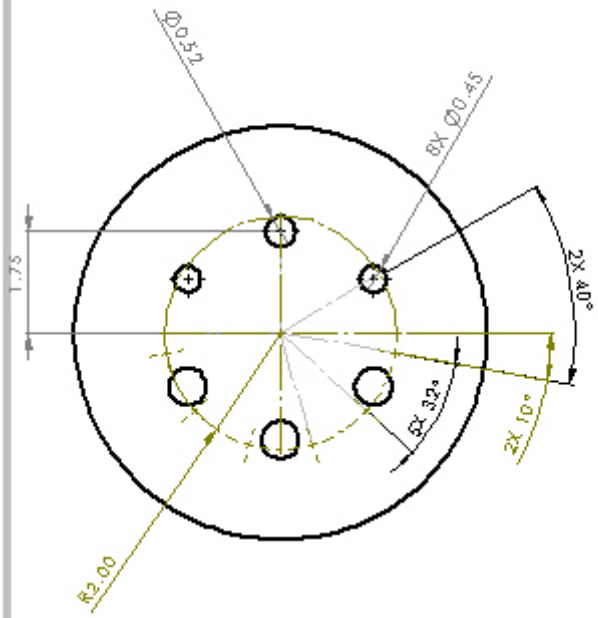
fixed dome connect

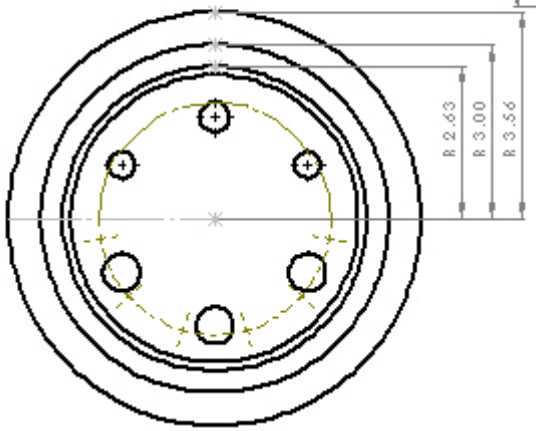
SCALE: 1:2 | WEIGHT: | SHEET 1 OF 1

REVISIONS				
ZONE	REV.	DESCRIPTION	DATE	APPROVED
	-	INITIAL DRAWING GENERATION	1/12/2015	JARRETT



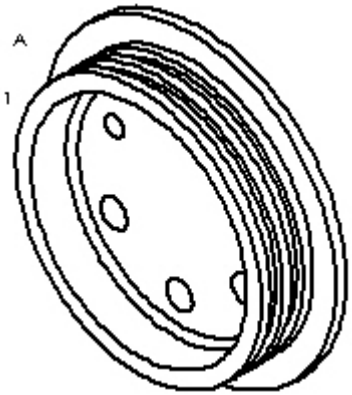






DETAIL A

SCALE 2 : 1



	MARK	DESIGNED	DATE						
DRAWN									
CHKD									
APPVD									
MFG									
Q.A.									

ROV End Cap

MATERIAL: Al 6061 Alloy

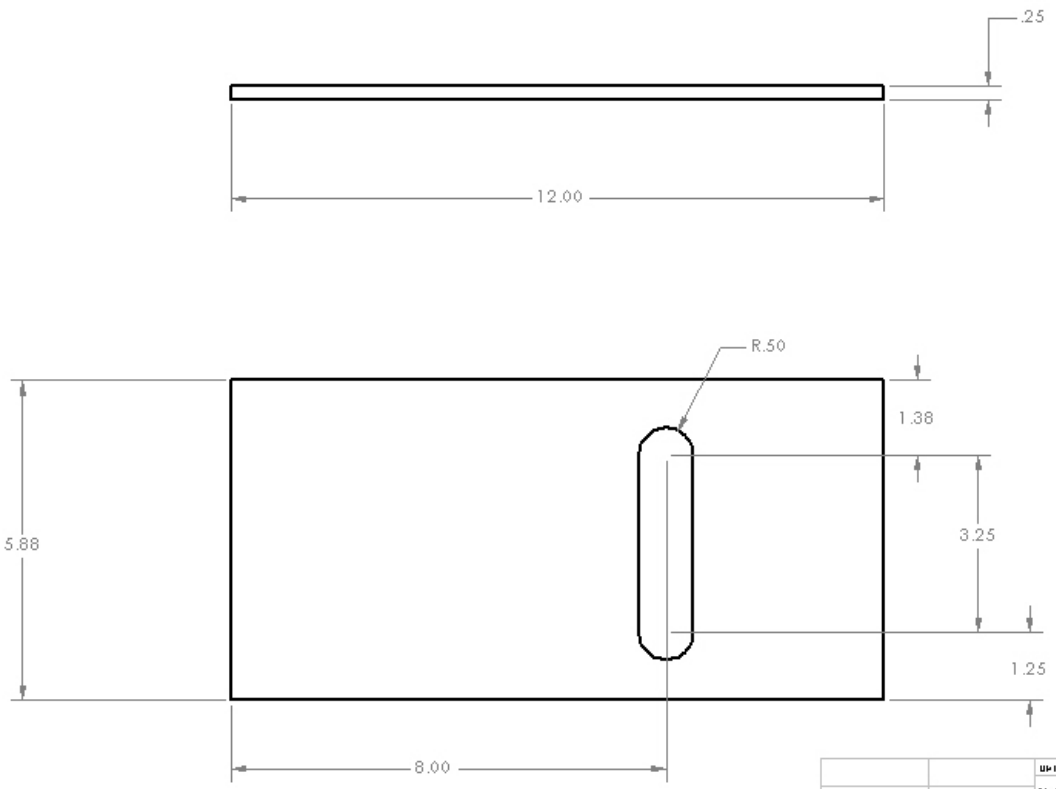
DWG NO.

A3

SCALE:

SHEET

REVISIONS				
ZONE	REV.	DESCRIPTION	DATE	APPROVED
	-	INITIAL DRAWING GENERATION	1/12/2015	J. LINOWES



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN	J LINOWES	1/12/2015
			CHECKED	S. SMITH	1/15/2015
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL 6061 ALUMINUM	COMMENTS:		
NEXT ASSY	USED ON	FINISH BREAK AND DEBUR ALL EDGES			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

Electronics Board

SIZE	DWG. NO.	REV
B		

SCALE: 1:2	WEIGHT:	SHEET 1 OF 1

Appendix B – Control Code Solution

Source Code (C++) From Driver Station Laptop

```
/******************************************************************************\
* Copyright (C) 2012-2014 Leap Motion, Inc. All rights reserved.               *
* Leap Motion proprietary and confidential. Not for distribution.             *
* Use subject to the terms of the Leap Motion SDK Agreement available at       *
* https://developer.leapmotion.com/sdk_agreement, or another agreement         *
* between Leap Motion and you, your company or other organization.            *
\******************************************************************************/
#include "ROV_Drive.h"
#include "Leap.h"

#ifdef USE_DIRECTX_SDK
#include <C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\include\xinput.h>
#pragma comment(lib,"xinput.lib")
#elif (_WIN32_WINNT >= 0x0602 /*_WIN32_WINNT_WIN8*/)
#include <XInput.h>
#pragma comment(lib,"xinput.lib")
#else
#include <XInput.h>
#pragma comment(lib,"xinput9_1_0.lib")
#endif


//-----------------------------------------------------------------------------
// Defines, constants, and global variables
//-----------------------------------------------------------------------------
#define MAX_CONTROLLERS 4  // XInput handles up to 4 controllers
#define CONTROLLER_DEADZONE  ( 0.15f * FLOAT(0x7FFF) )  // Default to 24% of the +/-
32767 range.   This is a reasonable default value but can be altered if needed.
#define CONTROLLER1 0
#define NUMBER_OF_BUTTONS 20

#define NUMBER_OF_LEAP_INPUTS 7
#define LEAP_DEADZONE 50

CSerial SerialPort;

struct CONTROLLER_STATE
{
	XINPUT_STATE state;
	bool bConnected;
};

CONTROLLER_STATE g_Controllers[MAX_CONTROLLERS];
ControllerInput Controller[NUMBER_OF_BUTTONS + 1];
WCHAR g_szMessage[4][1024] = { 0 };
HWND    g_hWnd;
bool    g_bDeadZoneOn = true;

LeapInput Leapvalues[NUMBER_OF_LEAP_INPUTS];
bool leapConnected = false;

char recieved[20][20];
```

```
int main()
{
	Leap::Controller controller;
	Leap::Frame frame;
	Leap::HandList hands;
	Leap::Hand h1;
	Leap::FingerList fingers;
	Leap::Finger index;
	Leap::Finger thumb;
	Leap::PointableList pointables;
	float indexX = 0, indexY = 0, indexZ = 0, thumbX = 0, thumbY = 0, thumbZ = 0, sum
= 0;
	unsigned long cycles = 0;

	int i = 0;

	// Setup serial port connection and needed variables.
	SerialPort.Open(7, 115200);

	Controller[20].value = 9;   //Verification Byte sent to make sure everything else
ends up in the right location
	FillByteSize();

	while (true)
	{
		cycles++;
		UpdateControllerState();    //Updates all values on the controller
		WORD wButtons = g_Controllers[CONTROLLER1].state.Gamepad.wButtons;

		//Stores all of the values from the controller into the controller
structure
		Controller[0].value = g_Controllers[CONTROLLER1].state.Gamepad.sThumbRX;
		Controller[1].value = g_Controllers[CONTROLLER1].state.Gamepad.sThumbRY;
		Controller[2].value = g_Controllers[CONTROLLER1].state.Gamepad.sThumbLX;
		Controller[3].value = g_Controllers[CONTROLLER1].state.Gamepad.sThumbLY;
		Controller[4].value =
(g_Controllers[CONTROLLER1].state.Gamepad.bRightTrigger);
		Controller[5].value =
(g_Controllers[CONTROLLER1].state.Gamepad.bLeftTrigger);
		Controller[6].value = (wButtons & XINPUT_GAMEPAD_RIGHT_THUMB);
		Controller[7].value = (wButtons & XINPUT_GAMEPAD_LEFT_THUMB);
		Controller[8].value = (wButtons & XINPUT_GAMEPAD_RIGHT_SHOULDER);
		Controller[9].value = (wButtons & XINPUT_GAMEPAD_LEFT_SHOULDER);
		Controller[10].value = (wButtons & XINPUT_GAMEPAD_DPAD_UP);
		Controller[11].value = (wButtons & XINPUT_GAMEPAD_DPAD_DOWN);
		Controller[12].value = (wButtons & XINPUT_GAMEPAD_DPAD_LEFT);
		Controller[13].value = (wButtons & XINPUT_GAMEPAD_DPAD_RIGHT);
		Controller[14].value = (wButtons & XINPUT_GAMEPAD_A);
		Controller[15].value = (wButtons & XINPUT_GAMEPAD_B);
		Controller[16].value = (wButtons & XINPUT_GAMEPAD_Y);
		Controller[17].value = (wButtons & XINPUT_GAMEPAD_X);
		Controller[18].value = (wButtons & XINPUT_GAMEPAD_START);
		Controller[19].value = (wButtons & XINPUT_GAMEPAD_BACK);

		CheckDeadZone();

		if (controller.isConnected() == true)
```

```
		{
			sum = 0;
			frame = controller.frame();
			hands = frame.hands();
			h1 = hands[0];
			fingers = frame.fingers();
			thumb = fingers[0];
			index = fingers[1];
			pointables = frame.pointables();

			Leapvalues[0].value = h1.palmVelocity().x;
			Leapvalues[1].value = h1.palmVelocity().y;
			Leapvalues[2].value = h1.palmVelocity().z;

			Leapvalues[3].value = h1.direction().pitch()*Leap::RAD_TO_DEG;
			Leapvalues[4].value = h1.direction().yaw()*Leap::RAD_TO_DEG;
			Leapvalues[5].value = h1.direction().roll()*Leap::RAD_TO_DEG;

			indexX = index.tipPosition().x;
			indexY = index.tipPosition().y;
			indexZ = index.tipPosition().z;

			thumbX = thumb.tipPosition().x;
			thumbY = thumb.tipPosition().y;
			thumbZ = thumb.tipPosition().z;

			Leapvalues[6].value = sqrt(pow((indexX - thumbX), 2) + pow((indexY -
thumbY), 2) + pow((indexZ - thumbZ), 2));

			leapConnected = true;
			CheckLeapDeadZone();

			std::cout << Leapvalues[5].value << '\t' << Leapvalues[6].value <<
std::endl;
		}

		for (i = 6; i < NUMBER_OF_BUTTONS; i++)  //DO NOT SET TO <=
NUMBER_OF_BUTTONS, NOT A MISTAKE. Verification bit should always keep its value
		{
			{
				Controller[i].value = AnalogToDigital(Controller[i].value);
	//converts all of the button presses on the controller to a binary value
			}
		}

		//turns all of the numerical values into buffers that can be passed to the
arduino
		for (i = 0; i <= NUMBER_OF_BUTTONS; i++)
		{
			_itoa_s(Controller[i].value, Controller[i].passedValue, 10);
		}

		if (leapConnected = true)
		{
			for (i = 0; i < NUMBER_OF_LEAP_INPUTS; i++)
			{
				_itoa_s(Leapvalues[i].value, Leapvalues[i].passedValue, 10);
			}
```

```
			}

			if (SendData() == 1)
			{
				//SerialPort.ReadData(recieved[0], 3);
			}

			//std::cout << Controller[8].value << std::endl;
			printf("%d", cycles);
			printf("\n");
		}
		return 0;
}

//--------------------------------------------------------------------------------
HRESULT UpdateControllerState()
{
		DWORD dwResult;
		for (DWORD w = 0; w < MAX_CONTROLLERS; w++)
		{
			// Simply get the state of the controller from XInput.
			dwResult = XInputGetState(w, &g_Controllers[w].state);

			if (dwResult == ERROR_SUCCESS)
				g_Controllers[w].bConnected = true;
			else
				g_Controllers[w].bConnected = false;
		}

		return S_OK;
}

//Defaulty, each controller button has its own respective value for high when pressed.
//Analog to digital converts them all to a binary value so less data needs to be sent to
the arduino
int AnalogToDigital(int ControllerVal)
{
		if (ControllerVal != 0)
		{
			return 1;
		}

		else
			return 0;
}

//Tells the structure what the packet size should be for each piece of data
//analog sticks need 7 bytes, triggers take 4, buttons take 2
void FillByteSize()
{
		int i = 0;
		for (i = 0; i <= NUMBER_OF_BUTTONS; i++)
		{
			if (i <= 3)
				Controller[i].packetSize = 7;

			if (i >= 4 && i <= 5)
				Controller[i].packetSize = 4;
```

```
		if (i >= 6 && i <= NUMBER_OF_BUTTONS)
		{
			Controller[i].packetSize = 2;
		}
	}

	for (i = 0; i < NUMBER_OF_LEAP_INPUTS; i++)
	{
		Leapvalues[i].packetSize = 10;
	}
}


//Checks the value to the analog sticks and converts them to zero if they are under the
threshold
void CheckDeadZone()
{
	int i = 0;
	for (i = 0; i <= 3; i++)
	{
		if (abs(Controller[i].value) < CONTROLLER_DEADZONE)
		{
			Controller[i].value = 0;
		}
	}
}

void CheckLeapDeadZone()
{
	int i = 0;
	for (i = 0; i < 3; i++)
	{
		if (abs(Leapvalues[i].value) < LEAP_DEADZONE)
			Leapvalues[i].value = 0;
	}
}

int SendData()
{
	int i = 0;
	int count = 0;
	char ioControlBuffer[3];
	char ioControl[3];
	int bytesRecieved = 0;
	SerialPort.SendData("hg1", 3);

	while (true)
	{
		//Sleep(20);
		bytesRecieved = SerialPort.ReadData(ioControlBuffer, 1);

		ioControl[0] = ioControlBuffer[0];
		printf("%c\t%d\t", ioControlBuffer[0], bytesRecieved);
		if (ioControl[0] == 'r')
		{
			for (i = 0; i <= NUMBER_OF_BUTTONS; i++)
```

```
                        SerialPort.SendData(Controller[i].passedValue,
sizeof(Controller[i].passedValue));

                    if (leapConnected == true)
                    {
                        for (i = 0; i < NUMBER_OF_LEAP_INPUTS; i++)
                            SerialPort.SendData(Leapvalues[i].passedValue,
sizeof(Leapvalues[i].passedValue));
                    }

                    else
                    {
                        for (i = 0; i < NUMBER_OF_LEAP_INPUTS; i++)
                            SerialPort.SendData("0", 1);
                    }

                    count++;
                    if (count > 5)
                    {
                        _sleep(750);
                        return 0;
                    }
                }

                else if (ioControl[0] == 'q')
                {
                    return 1;
                }

                else
                {
                    printf("Error: Unknown codeword passed from arduino [%c], aborting
send\n", ioControl[0]);
                }
            }
}
```

Arduino Code on ROV

```
#include<Wire.h>
#include <Servo.h>

//Gyro and accelerometer:
/************************************************************
WIRING OF MPU-6050
VCC: +3.3V From Arduino (attach 0.1 microFarad capacitor to ground)
GND: GND
SCL: SCL
SDA: SDA
XDA: Not connected
XCL: Not connected
ADO: GND
INT: Digital Pin 2 on Arduino
************************************************************/


//Arduino global variables:
Servo m0,m1,m2,m3,m4,m5;
long minMotorValue = -400;
long maxMotorValue = 400;
long avgMotorValue = 0;

//Raw values passed from the C++ program. Kept as seperate variables mainly for intuitiveness
int RT, LT, R3, L3, RB, LB, UpD, DownD, LeftD, RightD, A, B, X, Y, Start, Back, Verification;
int RBPrevState = 0;
long RX, RY, LX, LY;

//Leap motion controller values passed from the C++ program
int palmVX = 0, palmVY = 0, palmVZ = 0, palmPitch = 0, palmYaw = 0, palmRoll = 0, clawDistance = 0;
boolean usingLeap = false;

void setup()
{
  InitializeESCs();

  Serial.begin(115200, SERIAL_8N1);  //Baud here must match baud on c++ side
  Serial.setTimeout(100);
}

void loop()
{
  if(Serial.available() > 0)
  {
    if(ReadInValues()!= -1)
```

```
   {
     if (DriveControl() == 1)
     {
       Move();
     }

//     else if (DriveControl() == -1)
//     {
//       LeapMove();
//     }
   }

   else
   {
     StopROV();
   }
  }
}

int ReadInValues()
{
  Serial.print("r");
  RX = Serial.parseInt();
  RY = Serial.parseInt();
  LX = Serial.parseInt();
  LY = Serial.parseInt();
  RT = Serial.parseInt();
  LT = Serial.parseInt();
  R3 = Serial.parseInt();
  L3 = Serial.parseInt();
  RB = Serial.parseInt();
  LB = Serial.parseInt();
  UpD = Serial.parseInt();
  DownD = Serial.parseInt();
  LeftD = Serial.parseInt();
  RightD = Serial.parseInt();
  A = Serial.parseInt();
  B = Serial.parseInt();
  Y = Serial.parseInt();
  X = Serial.parseInt();
  Start = Serial.parseInt();
  Back = Serial.parseInt();
  Verification = Serial.parseInt();
  palmVX = Serial.parseInt();
  palmVY = Serial.parseInt();
  palmVZ = Serial.parseInt();
  palmPitch = Serial.parseInt();
  palmYaw = Serial.parseInt();
```

```
  palmRoll = Serial.parseInt();
  clawDistance = Serial.parseInt();

  if (Verification != 9)
  {
    return -1;
  }

  else
  {
    Serial.print("q");
  }

  RX = map(RX, -32767, 32767, -250, 250);
  RY = map(RY, -32767, 32767, -250, 250);
  LX = map(LX, -32767, 32767, -250, 250);
  LY = map(LY, -32767, 32767, -250, 250);
  LT = map(LT, 0, 255, 0, 250);
  RT = map(RT, 0, 255, 0, -250);
  return 0;
}

int DriveControl()
{
  if (usingLeap == true && RBPrevState == 1 &&camera RB == 0)
  {
    usingLeap = false;
  }

  else if (usingLeap == false && RBPrevState == 1 && RB == 0)
  {
    usingLeap = true;
  }

  RBPrevState = RB;

  if (usingLeap == false)
    return 1;

  else if (usingLeap == true)
    return -1;
}

void Move()  //digital write pushes water out, high sucks water in
{
  //Control for motors 0 and 1, thrusters on back of ROV
   if (LY != 0 && RX == 0)
  {
```

```
  writeMotor(m0, LY);
  writeMotor(m1, LY);
 }

 else if (LY == 0 && RX != 0)
 {
  writeMotor(m0, RX);
  writeMotor(m1, -(RX));
 }

 else if (LY != 0 && RX != 0)
{
  writeMotor(m0, DirectionCorrection(LY, RX, '+'));
  writeMotor(m1, DirectionCorrection(LY, RX, '-'));
}
 // ROV isn't moving back thrusters
 else
 {
  writeMotor(m0, 0);
  writeMotor(m1, 0);
 }

 //motor 2 and 3 control, RT is decend, LT is ascend
 if (RY == 0 && (RT != 0  || LT != 0))
 {
  if (RT != 0 && LT == 0)
  {
   writeMotor(m2, RT);
   writeMotor(m3, RT);
  }

  else if (RT == 0 && LT != 0)
  {
   writeMotor(m2, LT);
   writeMotor(m3, LT);
  }

  else
  {
   writeMotor(m2, 0);
   writeMotor(m3, 0);
  }
 }

 else if (RY != 0 && RT == 0 && LT == 0)
 {
  writeMotor(m2, -(RY));
  writeMotor(m3, RY);
```

```
  }

  else if (RY != 0 && (RT != 0  || LT != 0))
  {
   if (RT != 0 && LT == 0)
   {
    writeMotor(m2, DirectionCorrection(RT, RY, '+'));
    writeMotor(m3, DirectionCorrection(RT, RY, '-'));
   }

   else if (RT == 0 && LT != 0)
   {
    writeMotor(m2,  DirectionCorrection(LT, RY, '-'));
    writeMotor(m3,  DirectionCorrection(LT, RY, '+'));
   }

   else
   {
    writeMotor(m2, 0);
    writeMotor(m3, 0);
   }
  }

  else
  {
   writeMotor(m2, 0);
   writeMotor(m3, 0);
  }
  //motor 4 and 5 control
  writeMotor(m4, -(LX));
  writeMotor(m5, LX);

}

void StopROV()
{
 m0.writeMicroseconds(1500);
 m1.writeMicroseconds(1500);
 m2.writeMicroseconds(1500);
 m3.writeMicroseconds(1500);
 m4.writeMicroseconds(1500);
 m5.writeMicroseconds(1500);
}

void InitializeESCs()
{
  m0.attach(3);
  m1.attach(4);
```

```
 m2.attach(5);
 m3.attach(6);
 m4.attach(7);
 m5.attach(8);
 pinMode(3, OUTPUT);
 pinMode(4, OUTPUT);
 pinMode(5, OUTPUT);
 pinMode(6, OUTPUT);
 pinMode(7, OUTPUT);
 pinMode(8, OUTPUT);
 m0.writeMicroseconds(1500);
 m1.writeMicroseconds(1500);
 m2.writeMicroseconds(1500);
 m3.writeMicroseconds(1500);
 m4.writeMicroseconds(1500);
 m5.writeMicroseconds(1500);

 delay(1500);
}

void writeMotor(Servo motor, int value)
{
 motor.writeMicroseconds((1500 + value));
}

int DirectionCorrection(long primaryValue, long secondaryValue, char side)
{
 int correctedValue = 0;

 if (side == '+')
 {
  correctedValue = primaryValue/2 + primaryValue*secondaryValue/800;
 }

 else if (side == '-')
 {
  correctedValue = primaryValue/2 - primaryValue*secondaryValue/800;
 }

 return correctedValue;
}
```

GUI Code on ROV

```
/*****************************************************************************
 * GUI.java
 *
 * Creates an instance of WindowManager and starts the GUI. GUI.java is the
```

```
 * main file for the ROV GUI.
 *
 * @author Jerry Rosati
 * @since 2/2/15
 *
 ********************************************************************************/

package rov.gui;

import javax.swing.SwingUtilities;
import javax.swing.*;

public class GUI {

	public static void main(String[] args) {

		// Start the GUI on the initial thread
		SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				WindowManager wm = new WindowManager();
			}
		});
	}
}

/********************************************************************************
 * WindowManager.java
 *
 * Creates the different components for the GUI and adds them to guiFrame,
 * which is the main GUI frame. The GUI will have the following components:
 *
 * - A camera stream from the webcam on the ROV
 * - A table that displays the current sensor values
 * - A compass to show orientation of the robot
 * - A 3D representation of the ROV to also show orientation
 *
 * @author Jerry Rosati
 * @since 11/13/14
 *
 ********************************************************************************/

package rov.gui;

import rov.gui.elements.CameraPanel;
import rov.gui.elements.CameraStream;
import rov.gui.elements.GUIMenu;
import rov.gui.elements.GlassPane;
import rov.gui.elements.SensorPanel;

import java.awt.Dimension;
import java.awt.Toolkit;
```

```java
import javax.swing.*;

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.highgui.VideoCapture;

public class WindowManager //extends SwingWorker<Integer, Integer>
{
	// GUI components
	private JFrame		guiFrame;
	private GUIMenu		menu;
	private GlassPane	glassPane;
	private CameraPanel	camPanel;
	private CameraStream stream;
	//private SensorPanel  sensorPanel;

	private Server			tcpServer;

	// Constant values
	private final int	portNumber = 5555; // TCP Port
	private final int	camChannel = 1;	// Webcam channel

	/**************************************************************************
	 * Constructor
	 *
	 * Creates all of the components for the GUI and adds them to guiFrame.
	 * After adding the components, the GUI is displayed and the camera stream
	 * is started.
	 *
	 **************************************************************************/

	public WindowManager()
	{
		// Load OpenCV libraries into the program
		System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

		// Create the frames and panels
		guiFrame	= new JFrame("ROV3");
		camPanel	= new CameraPanel();
		menu		= new GUIMenu();
		glassPane	= new GlassPane();
		//sensorPanel = new SensorPanel();

		// Add components to the main GUI frame and display the GUI
		initializeGUI();

		// Start the camera stream
		stream = new CameraStream(camPanel, camChannel);
		stream.execute();
```

```
		resizeGUI();

		// Start the TCP Server
		tcpServer = new Server(portNumber);
		tcpServer.execute();
	}

	/**************************************************************************
	 * initializeGUI()
	 *
	 * Adds all of the components to guiFrame.
	 *
	 *************************************************************************/

	private void initializeGUI()
	{
		Dimension screenSize;
		screenSize = Toolkit.getDefaultToolkit().getScreenSize();

		// Initialize functionality for the main GUI frame.
		guiFrame.setSize(screenSize.width - 200, screenSize.height - 200);
		guiFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		//guiFrame.setContentPane(camPanel);
		guiFrame.setGlassPane(glassPane);
		guiFrame.setVisible(true);
		guiFrame.add(camPanel);

		glassPane.setVisible(true);
		glassPane.setOpaque(false);
	}

	/**************************************************************************
	 * resizeGUI()
	 *
	 * Resize the GUI dimensions to be the same as the video resolution
	 *
	 *************************************************************************/

	public void resizeGUI()
	{
		Dimension camRes;
		camRes = stream.getCameraSize();

		// Hard-code the size rather than getting the camera resolution
		// Change this after because the other one doesn't work

		guiFrame.setSize(1320, 760);
	}
}

/**************************************************************************
```

```
 * FrameException.java
 *
 * Custom Exception that is thrown when the webcam can't retrieve frames
 * when streaming.
 *
 * @author Jerry Rosati
 * @since 2/10/15
 *
 ********************************************************************************/

package rov.gui.exceptions;

import java.lang.Exception;
import java.lang.String;

public class FrameException extends Exception {

	public FrameException()
	{
		super();
	}

	public FrameException(String message)
	{
		super(message);
	}

	public FrameException(String message, Throwable cause)
	{
		super(message, cause);
	}

	public FrameException(Throwable cause)
	{
		super(cause);
	}
}

/********************************************************************************
 * Listener.java
 *
 * Custom MouseListener class
 *
 * @author Jerry Rosati
 * @since 3/23/15
 *
 ********************************************************************************/

package rov.gui.events;

import rov.gui.elements.GlassPane;
```

```java
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.JComponent;
import javax.swing.event.MouseInputAdapter;

public class Listener extends MouseInputAdapter implements MouseListener
{
	private final JComponent component;
	private GlassPane glassPane;

	public Listener(JComponent component, GlassPane glassPane)
	{
		this.component = component;
		this.glassPane = glassPane;
	}

	/**************************************************************************
	 * mousePressed(MouseEvent me)
	 *
	 *
	 * Get the point that was clicked. This will act as the origin for the
	 * rectangle that will be created if the mouse is dragged. Set the
	 * destination coordinates to (0, 0) to ensure that a rectangle won't
	 * be created when the mouse is pressed.
	 *
	 * @param me The mouse event
	 *
	 *************************************************************************/

	@Override
	public void mousePressed(MouseEvent me)
	{
		System.out.println("Pressed");
		Point origin = new Point(me.getX(), me.getY());
		Point destination = new Point(0, 0);

		glassPane.setOrigin(origin);
		glassPane.setDestination(destination);
		glassPane.repaint();
	}

	/**************************************************************************
	 * mouseReleased
	 *
	 *
	 *
	 *************************************************************************/
```

```
@Override
public void mouseReleased(MouseEvent me)
{
	System.out.println("Released");

}

/**************************************************************************
 * mouseEntered
 *
 *
 *
 *************************************************************************/

@Override
public void mouseEntered(MouseEvent me)
{
	System.out.println("Entered");
}

/**************************************************************************
 * mouseExited
 *
 *
 *
 *************************************************************************/

@Override
public void mouseExited(MouseEvent me)
{
	System.out.println("Exited");
}

/**************************************************************************
 * mouseClicked
 *
 *
 *
 *************************************************************************/

@Override
public void mouseClicked(MouseEvent me)
{
	System.out.println("Clicked");
}

/**************************************************************************
 * mouseDragged(MouseEvent me)
 *
 * Set the destination coordinates to whatever the current coordinates are
 * and repaint the panel. This creates a rectangle with a dynamic size.
```

```
         *
         * @param me The mouse event
         *
         ************************************************************************/

        @Override
        public void mouseDragged(MouseEvent me)
        {
                System.out.println("Dragged");
                Point destination = new Point(me.getX(), me.getY());
                glassPane.setDestination(destination);
                glassPane.repaint();
    }
}

/*************************************************************************
 * MeasurementTableModel.java
 *
 * Table model for the tables on the GUI
 *
 ************************************************************************/


package rov.gui.elements;

import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;

public class MeasurementTableModel extends AbstractTableModel
{
        private String[] columnNames;
        private Object[][] data = {{"0", "0", "0"}};
        private int numRows;
        private int numCols;

        /************************************************************************
         * MeasurementTableModel(String[] _columnNames, int row, int col)
         *
         * Initializes the table
         *
         ************************************************************************/

        public MeasurementTableModel(String[] _columnNames, int row, int col)
        {
                super();

                data = new Object[row][col];
                this.columnNames = new String[_columnNames.length];
                columnNames = _columnNames;
```

```
        numRows = row;
        numCols = col;

        initializeValues();
}

/**************************************************************************
 * initializeValues
 *
 * Initialize the values in the table by setting the column titles and
 * setting all measurement values to 0
 *
 **************************************************************************/

public void initializeValues()
{
        // Set the column titles for the columns
        for (int j = 0; j < getColumnCount(); j++) {
                setValueAt(columnNames[j], 0, j);
        }

        // Initialize all measurements to 0
        for (int i = 0; i < getColumnCount(); i++) {
                setValueAt("0", 1, i);
        }
}

/**************************************************************************
 * updateValues(Object value)
 *
 * Updates the measurement values in the table.
 *
 * @param values The new table values
 *
 **************************************************************************/

public void updateValues(String[] values)
{
        for (int i = 0; i < getColumnCount(); i++) {
                setValueAt(values[i], 1, i);
        }
}

/**************************************************************************
 * setValueAt(Object value, int row, int col)
 *
 * Sets the value in the table in a specific location
 *
 * @param value The new table value
 * @param row The row to change
```

```
     * @param col The column to change
     *
     *************************************************************************/

    @Override
    public void setValueAt(Object value, int row, int col)
    {
           data[row][col] = value;
           fireTableCellUpdated(row, col);
    }

    /*************************************************************************
     * getColumnCount
     *
     * Gets the number of columns in the table
     *
     * @return The number of columns in the table
     *
     *************************************************************************/

    @Override
    public int getColumnCount()
    {
           return columnNames.length;
    }

    /*************************************************************************
     * getRowCount
     *
     * Gets the number of rows in the table
     *
     * @return The number of rows in the table
     *
     *************************************************************************/

    @Override
    public int getRowCount()
    {
           return data.length;
    }

    /*************************************************************************
     * getValueAt(int row, int col)
     *
     * Gets the value at a location in the table
     *
     * @param row The row to look in
     * @param col The column to look in
     *
     * @return The value at (row, col)
     *
```

```
	 ***************************************************************************/

	@Override
	public Object getValueAt(int row, int col)
	{
		return data[row][col];
	}

	/*****************************************************************************
	 * isCellEditable(int row, int col)
	 *
	 * Allows a cell to be editable.
	 *
	 * @param row The row of the cell
	 * @param col The column of the cell
	 *
	 * @return true The cell can be edited
	 * @return false The cell can't be edited
	 *
	 ****************************************************************************/

	@Override
	public boolean isCellEditable(int row, int col)
	{
		return true;
	}
}

/*******************************************************************************
 * SensorPanel.java
 *
 * The SensorPanel will display information from the different sensors on
 * the ROV. The data will be output to a JTable, where it will be updating
 * in real time. The data will come from the C++ program that acts as a
 * TCP client, which will retrieve the data from the Arduino.
 *
 * @author Jerry Rosati
 * @since 3/23/15
 *
 ******************************************************************************/

package rov.gui.elements;

import rov.gui.elements.MeasurementTableModel;

import java.awt.AlphaComposite;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
```

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.RenderingHints;

import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.border.MatteBorder;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;

public class SensorPanel extends JPanel
{
	// Table components
	private JTable                measurements;
	private MeasurementTableModel  tableModel;
	private JScrollPane           scrollPane;
	private TableColumn           col;
	private String[]              colLabels = {"Depth",

	"Height of Object",
                                                                    "Width of
Object",
                                                                   "Distance to
Target"};

	private GridBagLayout gbl;
	private GridBagConstraints gbc;

	// Table dimensions
	private final int columnWidth = 120;
	private final int rowHeight = 20;
	private final int numColumns = 4;
	private final int numRows = 2;

	// Constants for the measurement type for the table.
	private final static int DEPTH    = 0;
	private final static int HEIGHT   = 1;
	private final static int WIDTH    = 2;
	private final static int DISTANCE = 3;

	/**************************************************************************
	 * Constructor
	 *
	 *
	 *
```

```
 ***************************************************************************/

public SensorPanel()
{
	super();
	gbl = new GridBagLayout();
	gbc = new GridBagConstraints();

	this.setLayout(gbl);
	this.setOpaque(true);

	initializeTable();
	this.addComponents();
}

/***************************************************************************
 * initializeTable()
 *
 * Initializes the table functionality and aesthetics.
 *
 ***************************************************************************/

public void initializeTable()
{
	tableModel = new MeasurementTableModel(colLabels, numRows, numColumns);
	measurements = new JTable(tableModel);
	scrollPane = new JScrollPane(measurements);
	col = null;

	measurements.setBorder(new MatteBorder(1, 1, 0, 0, Color.GRAY));
	measurements.setFocusable(false);
	setTableDimensions(rowHeight, columnWidth);
}

/***************************************************************************
 * setTableDimensions(int rowSize, int colSize)
 *
 * Set the size of the table by setting the size of the rows and the
 * size of the columns
 *
 * @param rowSize The size of each of the rows to change the height
 * @param colSize The size of each of the columns to change the width
 *
 ***************************************************************************/

public void setTableDimensions(int rowSize, int colSize)
{
	// Set column width of all of the columns
	for (int i = 0; i < tableModel.getColumnCount(); i++) {
	    col = measurements.getColumnModel().getColumn(i);
	    col.setPreferredWidth(colSize);
```

```
        }

        // Set row height of all of the rows
        for (int i = 0; i < tableModel.getRowCount(); i++) {
                measurements.setRowHeight(i, rowHeight);
        }
}

/*****************************************************************************
 * getTableDimensions()
 *
 * Get the table dimensions. The table dimensions will be equal to
 * the size of all of the columns by the size of all of the rows.
 *
 * @return The size of the table as a Dimension
 *
 *****************************************************************************/

public Dimension getTableDimensions()
{
        int width;
        int height;

        width = tableModel.getColumnCount() * columnWidth;
        height = tableModel.getRowCount() * rowHeight;

        return new Dimension(width + 5, height + 5);
}

/*****************************************************************************
 * setData(Object values)
 *
 * Sets the data that is displayed in the table. The table will display
 * - the Depth of the ROV
 * - the height of the object
 * - the width of the object
 * - the Distance to the target
 *
 * @param type - The type of measurement coming in. The location in the
 * table is determined by the type.
 *
 *****************************************************************************/

public void setData(Object value, int type)
{
        switch(type)
        {
                // Set the depth
                case DEPTH:
                        tableModel.setValueAt(value, 1, DEPTH);
                        break;
```

```
                // Set the height of the object
                case HEIGHT:
                        tableModel.setValueAt(value, 1, HEIGHT);
                        break;

                // Set the width of the object
                case WIDTH:
                        tableModel.setValueAt(value, 1, WIDTH);
                        break;

                // Set the distance to the object
                case DISTANCE:
                        tableModel.setValueAt(value, 1, DISTANCE);
                        break;
        }
}

/**************************************************************************
 * addComponents()
 *
 * Add components to the SensorPanel using a GridBag Layout.
 *
 **************************************************************************/

public void addComponents()
{
        // Add measurement table to the panel
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.weightx = 1;
        gbc.weighty = 1;
        this.add(measurements, gbc);
}

/**************************************************************************
 * getPreferredSize()
 *
 *
 *
 **************************************************************************/

@Override
public Dimension getPreferredSize()
{
        return getTableDimensions();
}

/**************************************************************************
 * getMinimumSize()
```

```
         *
         *
         *
         ***************************************************************************/

        @Override
        public Dimension getMinimumSize()
        {
                return getTableDimensions();
        }

        /***************************************************************************
         * getMaximumSize()
         *
         *
         *
         ***************************************************************************/

        @Override
        public Dimension getMaximumSize()
        {
                return getTableDimensions();
        }

        /***************************************************************************
         * paintComponent(Graphics g)
         *
         *
         *
         ***************************************************************************/

        @Override
        public void paintComponent(Graphics g)
        {

        }
}

/*****************************************************************************
 * CameraStream.java
 *
 * The CameraStream class will stream from the connected webcam. The streaming
 * is done in its own thread so that the GUI continues to be responsive and
 * doesn't freeze.
 *
 * @author Jerry Rosati
 * @since 2/15/15
 *
 *****************************************************************************/

package rov.gui.elements;
```

```java
import rov.gui.exceptions.FrameException;

import java.lang.Thread;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.highgui.VideoCapture;
import org.opencv.highgui.Highgui;
import org.opencv.core.CvType;


public class CameraStream extends SwingWorker<Integer, Integer>
{
	private CameraPanel camPanel;
	private Mat webcamImage;
	private VideoCapture capture;

	private final int cameraWidth = 1280;
	private final int cameraHeight = 720;

	/**************************************************************************
	 * CameraStream()
	 *
	 * Begins camera capture on Channel 1. The cameras that are on the
	 * the different channels differ depending on what the computer is.
	 *
	 **************************************************************************/

	public CameraStream()
	{
		super();
		webcamImage = new Mat();
		capture = new VideoCapture(1);
	}

	/**************************************************************************
	 * CameraStream(CameraPanel _camPanel, int channel)
	 *
	 * Begins camera capture on the specified channel
	 *
	 * @param _camPanel The panel that the camera stream will display on
	 * @param channel The input channel to stream from
	 *
	 **************************************************************************/
```

```
public CameraStream(CameraPanel _camPanel, int channel)
{
	super();
	webcamImage = new Mat();
	capture = new VideoCapture(channel);

    setCameraSize(new Dimension(cameraWidth, cameraHeight));
	camPanel = _camPanel;
}

/**************************************************************************
 * doInBackGround()
 *
 * Streams the camera on a new thread. If the camera has an error when
 * trying to stream, then it will throw an exception.
 *
 * @return 0 for Success
 *
 *************************************************************************/

@Override
protected Integer doInBackground() throws Exception
{
	try {
		this.stream();

	} catch (FrameException fe) {
		System.err.println(fe.getMessage());
		throw new Exception("Error with webcam!");
	}

	return 0;
}

/**************************************************************************
 * setCameraPanel(CameraPanel _camPanel)
 *
 * Set the panel for the camera stream to display on.
 *
 *************************************************************************/

public void setCameraPanel(CameraPanel _camPanel)
{
	camPanel = _camPanel;
}

/**************************************************************************
 * getCameraPanel()
 *
 * @return The camera panel that the camera is currently streaming on.
 *
```

```
 ***************************************************************************/

public CameraPanel getCameraPanel()
{
	return camPanel;
}

/***************************************************************************
 * stream()
 *
 * Gets frames from the webcam (as Matrices) and converts them to
 * a BufferedImage before displaying them by repainting the camera panel.
 *
 ***************************************************************************/

public void stream() throws FrameException
{
	if (capture.isOpened()) {
		while(true) {
			capture.read(webcamImage);

			if (!webcamImage.empty()) {
				camPanel.matToBufferedImage(webcamImage);
				camPanel.repaint();
			}
		}

	} else {
		throw new FrameException("Can't capture webcam frame!\n");
	}
}


/***************************************************************************
 * getCameraSize()
 *
 * Gets the current resolution of the video stream.
 *
 * @return The resolution of the video stream as a Dimension.
 *
 ***************************************************************************/

public Dimension getCameraSize()
{
	return new Dimension(webcamImage.width(), webcamImage.height());
}

/***************************************************************************
 * setCameraSize(Dimension res)
 *
 * Sets the resolution of the camera stream
```

```
         *
         * @param res The new resolution of the video stream
         *
         * @return 1 Resolution change was successful
         * @return 0 Resolution change failed
         *
         ************************************************************************/

        public boolean setCameraSize(Dimension res)
        {
                boolean wMod; // If the width has been changed
                boolean hMod; // If the height has been changed

                wMod = capture.set(Highgui.CV_CAP_PROP_FRAME_WIDTH, res.getWidth());
            hMod = capture.set(Highgui.CV_CAP_PROP_FRAME_HEIGHT, res.getHeight());

            return (wMod & hMod);
        }
}

/*****************************************************************************
 * GlassPane.java
 *
 * The GlassPane will be a transparent panel on the GUI. It will cover the
 * camera stream so that it's possible to see the camera stream, but the
 * different components will actually be on the GlassPane. All mouse
 * interactions will also be done on the GlassPane.
 *
 * The GlassPane will have a BorderLayout
 *
 * @author Jerry Rosati
 * @since 3/21/15
 *
 ****************************************************************************/

package rov.gui.elements;

import rov.gui.events.Listener;
import rov.gui.elements.SensorPanel;

import java.awt.AlphaComposite;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.Insets;
```

```java
import java.util.ArrayList;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JPanel;

public class GlassPane extends JPanel
{
	private Listener mouseListener;
	private SensorPanel sensorPanel;

	private Point origin;
	private Point destination;

	private GridBagLayout gbl;
	private GridBagConstraints gbc;

	/**************************************************************************
	 * Constructor
	 *
	 * Create the glass pane and add mouse listeners to it
	 *
	 *************************************************************************/

	public GlassPane()
	{
		super();
		mouseListener = new Listener(this, this);
		sensorPanel   = new SensorPanel();

		origin        = new Point(0, 0);
		destination   = new Point(0, 0);

		gbc                = new GridBagConstraints();

		this.addMouseListener(mouseListener);
		this.addMouseMotionListener(mouseListener);

		this.setLayout(new GridBagLayout());
		this.addComponents();
	}

	/**************************************************************************
	 * paintComponent
	 *
	 * Draw a rectangle whose width is (destination.x - origin.x) and whose
	 * height is (destination.y - origin.y), where origin is the coordinates
	 * of the pixel where the mouse was pressed, and destination is the
	 * coordinates of the pixel the mouse is currently at.
	 *
	 * @param g Used to draw on the glass pane.
```

```
 *
 ************************************************************************/

@Override
public void paintComponent(Graphics g)
{
       int width;
       int height;
       Graphics2D g2d;

       width  = destination.x - origin.x;
       height = destination.y - origin.y;
       g2d = (Graphics2D) g;

       g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);

       // Make components on the glasspane semi-transparent
    g2d.setComposite(AlphaComposite.getInstance(
                             AlphaComposite.SRC_OVER, 1f));


    // Color the rectangle and draw it
       g2d.setColor(Color.blue);
       g2d.fillRect(origin.x, origin.y, width, height);
}

/************************************************************************
 * setOrigin(Point origin)
 *
 * Set the coordinates for the origin of the rectangle that will be drawn
 * when the mouse is dragged.
 *
 * @param origin
 *
 ************************************************************************/

public void setOrigin(Point origin)
{
       this.origin.x = origin.x;
       this.origin.y = origin.y;
}

/************************************************************************
 * setDestination(Point destination)
 *
 * Set the coordinates for the endpoint of the rectangle that will be
 * drawn when the mouse is dragged.
 *
 * @param destination
 *
```

```
         *********************************************************************/

        public void setDestination(Point destination)
        {
                this.destination.x = destination.x;
                this.destination.y = destination.y;
        }

        /**********************************************************************
         * addComponents()
         *
         * Add components to the GlassPane.
         *
         *********************************************************************/

        public void addComponents()
        {
                gbc.fill    = GridBagConstraints.NONE;  // Don't fill the line
                gbc.gridx   = 0;                                    // Start at
leftmost cell
                gbc.gridy   = 2;                                    // Start at bottom
cell
                gbc.weighty = 1;                                    // Grid takes up
entire space
                gbc.insets  = new Insets(10,20,40,0);   // Set padding
                gbc.anchor  = GridBagConstraints.LAST_LINE_START;
                this.add(sensorPanel, gbc);

                gbc.gridx = 0;
                gbc.gridy = 0;
                gbc.anchor = GridBagConstraints.WEST;
                gbc.weighty = 0;
                gbc.weightx = 1;
                this.add(new JButton("Test1"), gbc);
        }

        /**********************************************************************
         * getSensorPanel()
         *
         * Get the SensorPanel object that's on the glass pane
         *
         * @return A SensorPanel object
         *
         *********************************************************************/
        public SensorPanel getSensorPanel()
        {
                return this.sensorPanel;
        }
}

/**************************************************************************
```

```
 * CameraPanel.java
 *
 * Streams from the webcam to the GUI through the camera panel. The image
 * is read in as a matrix and converted to BufferedImage type, which is then
 * added to the camera panel.
 *
 * @author Jerry Rosati
 * @since 2/10/15
 *
 ********************************************************************************/

package rov.gui.elements;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
import javax.swing.*;
import org.opencv.core.Mat;

public class CameraPanel extends JPanel
{
	private BufferedImage image;

	/********************************************************************************
	 * Constructor
	 *
	 * Initializes the CameraPanel by creating a JPanel.
	 *
	 ********************************************************************************/

	public CameraPanel()
	{
		super();
	}

	/********************************************************************************
	 * matToBufferedImage(Mat matBGR)
	 *
	 * Converts a matrix representation of an image to a BufferedImage
	 * representation so that it can be displayed on the CameraPanel.
	 *
	 * @param matBGR The matrix representation of the image.
	 *
	 ********************************************************************************/

	public void matToBufferedImage(Mat matBGR)
	{
		//long startTime = System.nanoTime();
		//long endTime = 0;
		int width            = matBGR.width();
		int height        = matBGR.height();
```

```
            int channels      = matBGR.channels();
            byte[] srcPixels  = new byte[width * height * channels];
            byte[] destPixels = null;
            DataBufferByte dataBuffer;

            matBGR.get(0, 0, srcPixels);
            image = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);

            dataBuffer = ((DataBufferByte)image.getRaster().getDataBuffer());
            destPixels = dataBuffer.getData();

            System.arraycopy(srcPixels, 0, destPixels, 0, srcPixels.length);
            //endTime = System.nanoTime();
            /*System.out.println(String.format("Elapsed time: %.2f ms",
                                        (float)(endTime - startTime)));*/
      }

      /************************************************************************
       * paintComponent(Graphics g)
       *
       * Draws the image on the CameraPanel after it's converted from a matrix
       * to a BufferedImage.
       *
       * @param g The graphics object that will be used to draw
       *
       ************************************************************************/

      @Override
      public void paintComponent(Graphics g)
      {
            super.paintComponent(g);

            if (image == null) {
                  return;
            }

            g.drawImage(image, 10, 10, image.getWidth(), image.getHeight(), null);
      }
}
```