Autonomous Surface Vehicle Senior Research Project

University of New Hampshire
TECH 797 Undergraduate Ocean Research Projects 2015-2016
May 4, 2016

Team Members

Ryan Bachman – Mechanical Engineering
Liam Collins – Mechanical Engineering
Aaron Connolly – Mechanical Engineering
Isaac Gagnon – Computer Science
Tim Panella – Mechanical Engineering
John Perrella – Mechanical Engineering
Hunter Rowley – Mechanical Engineering

Project Advisors

Dr. May-Win Thein
Andrew D'Amore
Damian Manda

TECH 797 Acknowledgment

This work is the result of research sponsored in part by the New Hampshire Sea Grant College Program through NOAA grant #NA10OAR4170082, and the UNH Marine Program.

Table of Figures

Introduction

Autonomous surface vessels (ASVs) can be simply defined as self-driving boats; vessels that can function independently of human control and make decisions on their own to accomplish specific missions. The goal of the UNH ASV team is to develop a control module that is low cost, robust, and modular. This module must be purchasable as a component off the shelf and be able to be installed on any existing vessel allowing that vessel to obtain autonomous capabilities. For the UNH ASV team, basic autonomy can be defined in three stages: point-to-point navigation, obstacle avoidance (adaptively changing path to avoid collisions), and target recognition, tracking and trailing. In order for a boat to have basic autonomy it must satisfy all three of the stages. Long term, the UNH ASV team is concerned with building a fleet of ASVs and having them all communicating with one another, as well as with robots beneath and above the water's surface (UUVs and UAVs).

This technology is important because the autonomous maritime robotics industry is rapidly growing and is projected to be worth billions of dollars globally in the next few years. ASVs alone are a multi-million dollar and rapidly growing part of that market. By removing the human element from maritime operations, ASVs have unlimited potential for growth and applications. They serve to reduce cost, improve efficiency, and remove human lives from danger in a plethora of applications. A few example applications include ocean mapping, swarm defense for the military, oil pipeline monitoring, aquatic research, search and rescue, and commercial shipping.

This year is the third iteration of the ASV project at UNH. The previous two years helped to pave the way for the success of the team this year. ASV1 was custom built by the first year's team, but many problems arose which hindered their ability to develop a working vessel. Although ASV1 was never operational, the first team accomplished much on the autonomy coding and immeasurably helped the success of ASV3. The second year's team went a different route and purchased a 4-foot remote control (RC) boat. Like ASV1, ASV2 was not autonomous by the end of the year, however they left a solid platform to build on. During the summer of 2015 members of this year's ASV team worked on ASV2 and achieved basic first stage autonomous functions (waypoint navigation). This was a successful proof of concept and helpful launching pad for the entire ASV team in the '15-'16 academic year.

This year's team decided to follow the same route as the previous year's team and purchase a boat. However, instead of a small RC boat, this year's team purchased a 7-foot bass fishing boat. Drawing from the experiences of the previous two teams, a larger boat was deemed necessary in order to accommodate all the necessary sensors as well as increase the functionality and ease of working on the vessel. In order to achieve autonomy on a vessel of this size, a custom steering system was constructed that enabled remote and autonomous interfacing and control of the vessel's trolling motor. This year's team achieved first stage

autonomy on ASV3 and tested the mapping capabilities of a sonar device with the vessel. They also worked to improve its mission duration by experimenting with solar charging solutions during test operations, and to advance the UNH ASV program into second and third stages of autonomy. The team this year also focused on developing multi-vehicle communication and coordination between multiple ASVs and between ASV3 and the UNH remotely operated vehicle (ROV) team. Lastly, the team this year focused on refining the autonomy package on ASV3 to meet the robust, low cost, and modular design conditions.

Physical Platform

Because ASV3 was the vessel developed by this year's team, it will remain the main focus of this report. Minor discussion of work done on ASV2 will follow later as a supplement to the work done on ASV3.

Steering System

A. Physical Platform and Mechanism

Keeping with the spirit of a low-cost open-source ASV, a simple steering system was needed that could reuse as many original motor components as possible. As such, a commercial self-steering motor (such as a Minn Kota Riptide), was out of the question. The first ideas (include scans of the earliest drawings we still have) paid homage to the steering systems of commercial inboard motors.

Hydraulic pistons, cable pulleys, and gearing mechanisms were considered, before being dismissed as overly complex, heavy, and expensive. The small amount of power needed to turn the 30 lb. thrust trolling motor did not justify the high mechanical advantage of such a system.

Our first manufactured system was a modification of online plans (Simpson) (Figure 1).

Figure 1: Original Plan for friction drive steering system (Simpson)

This system uses a high-torque, low speed 12-volt automotive window motor to operate a friction drive system. Due to relative difficulty in making accurate curved wood cuts, as well as concerns regarding humidity swelling, it was decided to use the UNH mechanical engineering department's stereolithographic 3D printer. The dimensions of the semicircular arc were slightly modified (Figure 2), while the drive wheel underwent a more radical redesign (Figure 3).

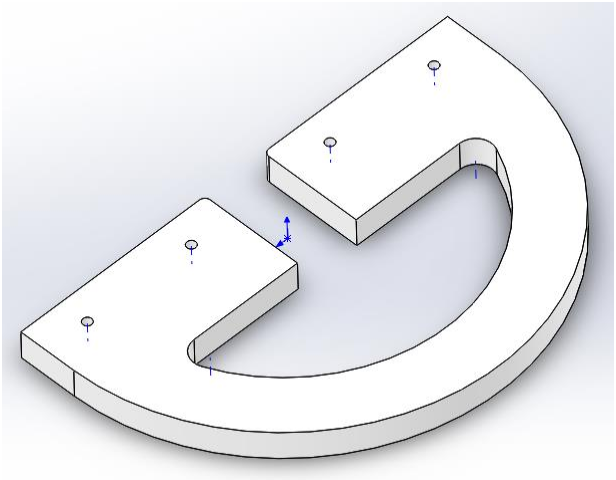


Figure 2: SolidWorks model of semicircle for friction drive steering system

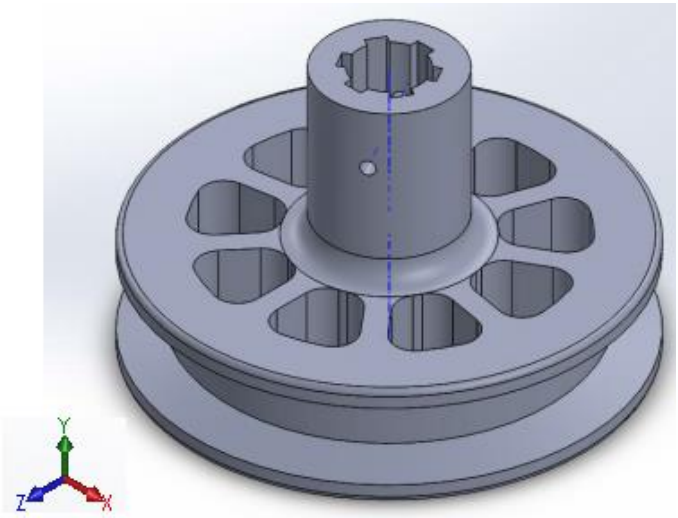
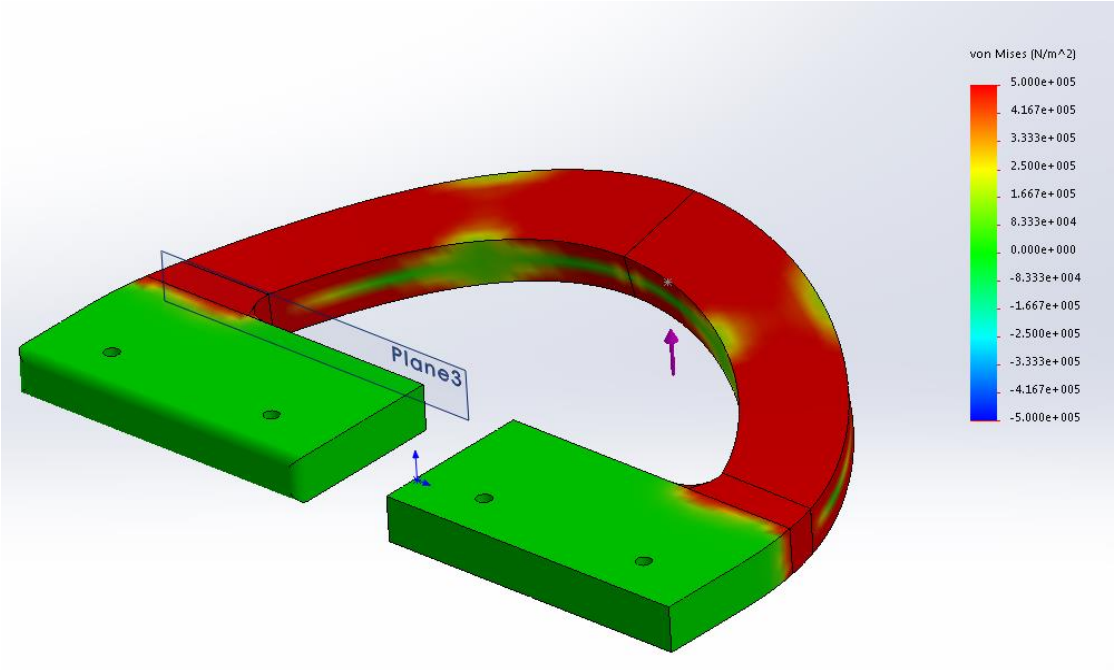


Figure 3: SolidWorks model of drive wheel for friction drive steering system

The center was hollowed out and replaced with spokes to lighten weight and reduce printing time. Along the circumference, a V-groove was formed to accept a serpentine V-belt to act as the means of providing friction contact with the semi-circle. Along the shaft, a spline matching that of the window motor was selected to allow for a tight fit. In SolidWorks, a simple stress analysis was performed to ensure the drive wheel and semicircle could withstand a point loading far in excess of the trolling motor weight. Figure 6 demonstrates that the semicircle did not fail due to excessive von Mises (resultant) stress. Shear and bending stress calculations were also made in all dimensions. Particular attention was paid toward bending normal stress in the z direction at the junction between the curved and rectangular portions of the part. It was felt that the risk of failure was greatest at this location and in the z dimension (Figure 4).



Figu

Figure 4: SolidWorks simulation showing von Mises stress under the influence of point loading (purple arrow)

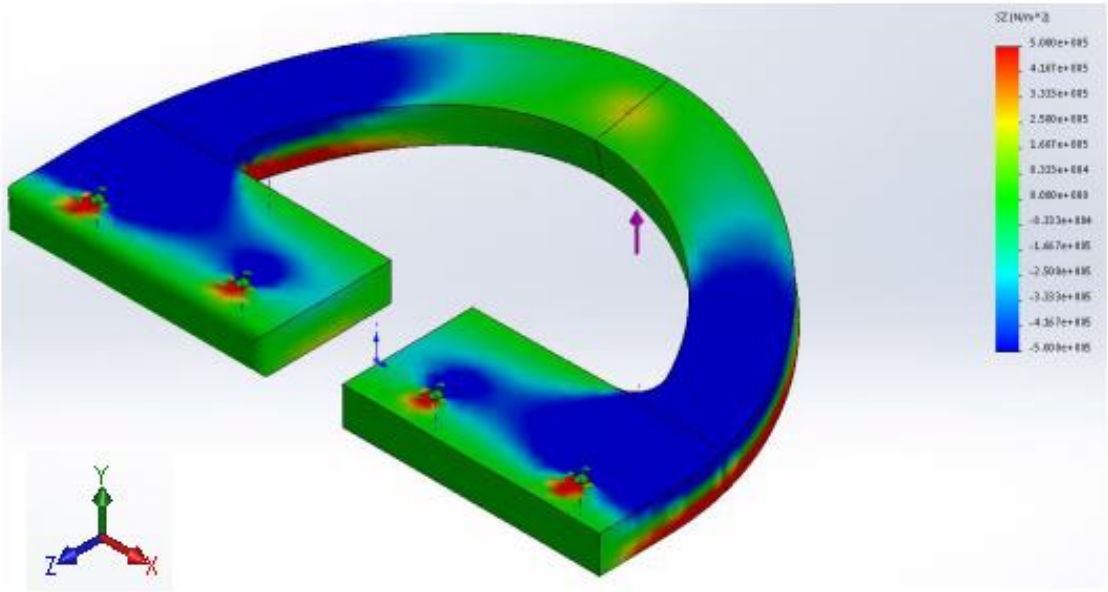


Figure 5: SolidWorks simulation showing bending normal stress in z direction under the influence of point loading (purple arrow).

Figure 6: Assembled steering system being tested on ASV in UNH Chase Ocean Engineering Laboratory pool

A crude plywood mount was fashioned and screwed directly into the boat hull. This mount held the window motor and drive wheel (Figure 6).

In practice, the system had several major shortcomings. Obtaining a drive belt of correct dimensions and significant tack was difficult. We opted for a wide belt to provide ample contact area. However, belts of such dimensions are rarely small in terms of length, which would have required the ordering of a custom made-belt. In the interim, a sheet of pliable rubber was cut and taped around the wheel. This proved unreliable, with the rubber often coming off during operation. Furthermore, when the rubber stayed on, the coefficient of friction was often not sufficient to prevent slippage between the drive wheel and the arc. The rubber-coated pipe clamps used to hold the arc to the trolling motor were difficult to attach and could slip after continuous operation. Several holes had to be modified after printing due to mistakes in measurement. The plastic of the hollow boat hull did not provide sufficient anchoring capability, leading to excessive vibration of the plywood mount. Finally, if one was not careful in remotely steering the boat, it was possible to completely push the arc off of the drive wheel, leading to a loss of control.

It was soon decided that the problems of the friction drive system outweighed the benefits of light weight, simple construction and low cost. However, the concept of using a 3D printed system on the trolling motor was judged to be sound, and development continued along this track. As opposed to friction, the revised system was to use a toothed gear mechanism. Considerations were made to use spur, bevel, and miter gears. The unique

challenges of our system precluded the purchase of commercially available gears. To install a gear on the trolling motor would have involved disassembly of the switching mechanism at the head of the motor, as well as possible drilling into the shaft to install a set-screw. Furthermore, this would have necessitated the re-routing of the trolling motor's wires. There was considerable difficulty in finding commercial gears with the large inner diameter to fit the trolling motor shaft, but without an excessively large outer diameter. On the drive wheel, a standard gear would have only relied on a setscrew to hold it to the spline of the window motor shaft, creating excessive play. Lastly, the high strength of metal gears was not needed, nor could it justify the high cost.

Originally, the drive gear was modified to include equation driven involute teeth, using the procedure given by Cao. Additionally, the semi-circular arc was given teeth to mimic a spur gear. As opposed to the perpendicular axes of the drive wheel and trolling motor originally used, the new system was to use parallel axes (Figure 7).

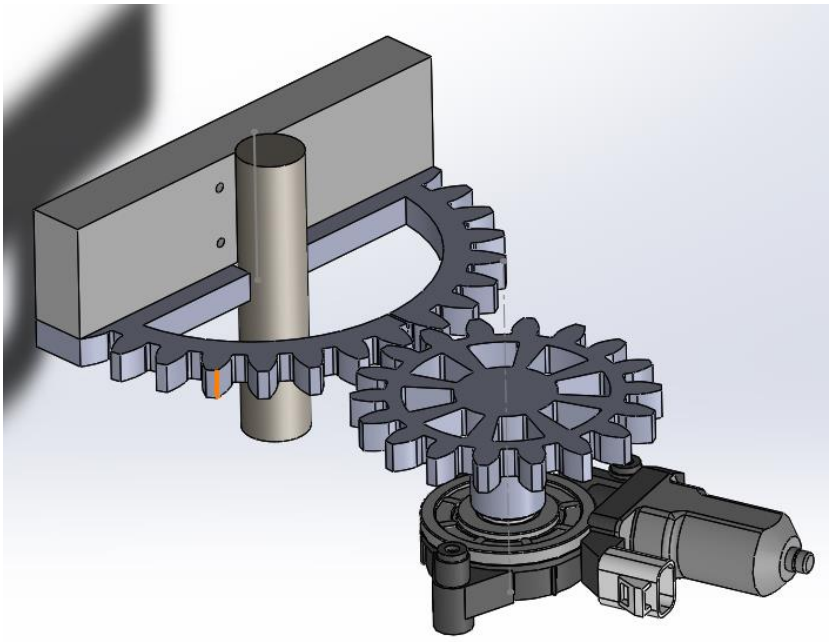


Figure 7: Proposed revised steering system using a gear drive

However, this still did not eliminate the possibility of the drive wheel separating from the arc. It was decided that the new system had to be physically incapable of separating, no matter what angle the motor was turned to. To this end, two spur gears were formed with a diameter similar to that of the original friction drive wheel. The splines and spokes of the original drive wheel were retained, and a small shaft bore was added to the top of the drive gear for the provision of a positional encoder to allow precise steering. The gear to be installed on the trolling motor was made in two parts to allow easy installation and removal (Figure 8).

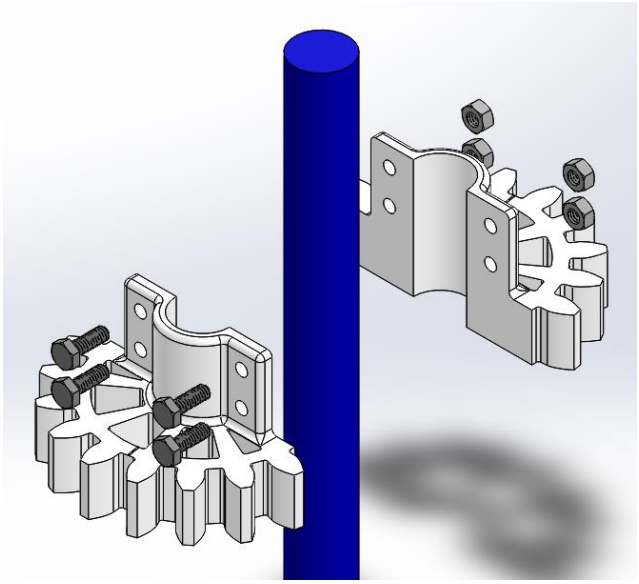


Figure 8: Two-part gear on trolling motor shaft, used for the revised drive system

Another difficulty that needed to be addressed was mounting the drive system to the boat. It was decided that screwing directly into the boat was not acceptable, due to the inability to secure fasteners from the other side of the hull, as well as the lack of flexibility that this gave in using the steering system on another boat. The final mount design takes advantage of the vise-like grip of the trolling motor mount. Brackets and wood are used to hold the window motor as well as the encoder above. This "shelf" design (Figure 9) allows easy access to components, and allows easy movement and stowing of the trolling motor.

Figure 9: Revised steering system design. Left: Motor disengaged and stored. Right: Motor engaged.

Finally, a precise encoder was needed to accurately find the rudder position. It was quickly determined that the new encoder should be absolute, rather than incremental. Such an encoder does not require counting the number of pulses to determine position. Likewise, the encoder can immediately read the rudder position on startup, whereas an incremental encoder would have to be reset to the zero position. Additionally, the encoder should have an interface that is easy to connect and read with an Arduino, and that the physical construction of the encoder should be rugged enough to withstand light pressure, shocking, and water. A comparison of encoder communication interface busses such as CAN bus, BiSS, serial-synchronous interface (SSI), and a straight binary interface. In terms of cost and ease of programming, a binary interface using Grey code was selected.

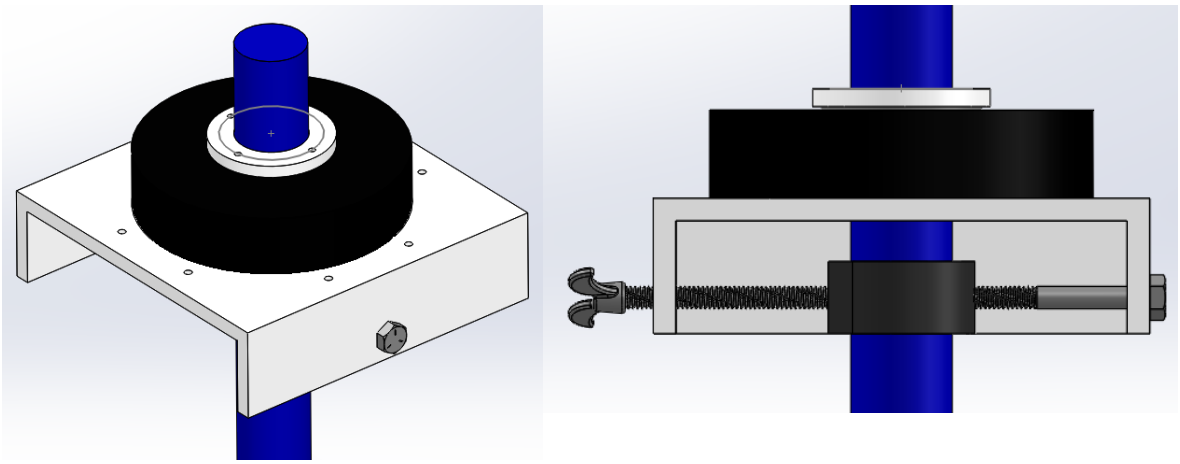


Figure 10: Originally proposed encoder mount with hollow shaft encoder.

On the advice of one of our professors, it was recommended that the encoder be sized and positioned to directly measure the motion of the trolling motor. This necessitated an encoder with a large hollow shaft and a means to mount the stationary portion of the encoder. One such mounting scheme is shown in Figure 10. The mounting bracket of the trolling motor is used to secure the white mounting plate that holds the outer ring of the encoder stationary. Normally, this portion of the mount holds the thumbscrew used to adjust and hold the rotational position of the trolling motor. Use of a longer thumbscrew as well as other supporting screws was proposed.

Unfortunately, this this was quickly realized to be impractical. The encoder had to be large enough to accommodate the large shaft of the trolling motor, while being thin enough and having a small enough outer diameter to fit into the confined space at the back of the boat. Likewise, quotes obtained from various vendors showed that such an encoder would be hundreds of dollars.

From this point, it was decided to use a small, solid shaft absolute encoder. An Ebay search led to the purchase of a Koyo TRD-NA series absolute rotary encoder from a buyer in China for a cost of approximately $40 (Figure 11).

Figure 11: Koyo TRD-NA series encoder

This encoder featured a 10-bit resolution Grey code output through a simple interface of 12 binary pins. The shaft of the encoder was inserted into a slot bore on the top of the drive gear. A 3D printed plate was custom made to hold the encoder stationary to the wooden "shelf" (Figure 12).

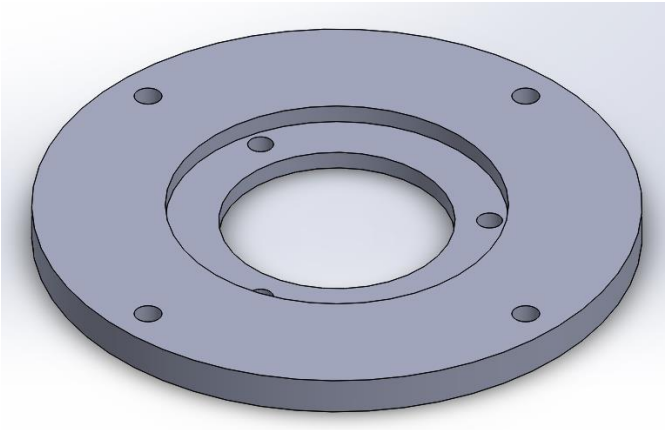


Figure 12: 3D printed ring used to hold and mount encoder.

B. Software Control

The window motor requires a 12 volt input and draws a few amps. An Arduino is only capable of a 5-volt output with a small amount of current. As such, the steering motor uses the same H-bridge to step the voltage up and handle the higher current. When the boat is operating in RC mode, the user turns and holds the steering wheel, turning the rudder. After the entire boat is pointed in the desired direction, the user releases the wheel and the rudder automatically returns to the zero position. One of the issues involving the active feedback control of the encoder involves error correction. Many electro-mechanical systems experience overshoot. Electric signals respond far faster than an inertial mass. After the current to the motor is turned off, the rotor will continue to rotate due to inertia. In a feedback loop, the

controller will actively fight this discrepancy between actual and desired position. This can lead to oscillation as the controller "hunts" for the desired position.

Citations for steering section:

Cao, Yang. "Modeling an Equation Driven Involute Spur Gear in Solidworks." Online video clip. *Youtube*, 23 Feb 2014. Web. 7 Feb 2015.

Simpson, Ken. "Remote Electric Steering for Trolling Motors." *Duckworks Boatbuilder's Supply.* N.p, 1 November 2010. Web. 21 October 2015.

Sonar System

One of the proposed applications for the UNH ASV is as an inexpensive, lightweight, and unmanned sonar mapping system. A simple depth finding sonar allows for both mapping and collision avoidance with submerged hazards. One of our graduate advisors, Damian Manda, graciously let us borrow his CEE Hydrosystems CEE Pulse 100 sonar set. This device is an active sonar; it can send out and receive pings. The set is capable of measuring depths from .3 to 100

Figure 13: CEE Pulse 100 standard 200 kHz transducer (User's Manual for CEEPulse 100).

Figure 14: CEE Pulse 100 control and power pack (User's Manual for CEEPulse 100)

meters. The transducer (Figure 13) is connected to the main controller and power pack (Figure 14). Data is transmitted through a serial port (through Bluetooth communication is possible).

As outlined in [1], simple submerged pipe was employed to mount the sonar on the boat. An extra Minn Kota trolling motor mount, identical to the model used to hold the ASV's trolling motor, was graciously loaned by Professor Foster. This mount allowed for a lightweight yet strong means to mount the sonar transducer to the front of the boat. A proprietary metal mounting device was obtained from the sonar manufacturer (Figure 15).

This device is designed to connect to a 2-inch diameter pipe. However, the Minn Kota mount can only handle a pipe of 1.25-inch diameter. Two plastic pipes of these disparate

Figure 15 CEE Hydrosystems sonar mounting bracket. Right: Cut away view

Figure 16 Sonar system with mount in storage position.

diameters were bolted together. Additionally, a slot cut was made through the 2-inch pipe to allow easy placement of the transducer cable. The mounted sonar is shown in its storage position in Figure 16.

A test of this sonar system was conducted at Swain's Lake near Durham. The ASV was placed into autonomy mode, where it made several loops at low speed. During this time, the sonar took depth measurements and the data was logged. Computer code used by our graduate advisor, Damian Manda, was used to plot these depth readings at their exact location in space. A color coded system was used to show the depth reading in meters. These results can be seen in Figure 17.

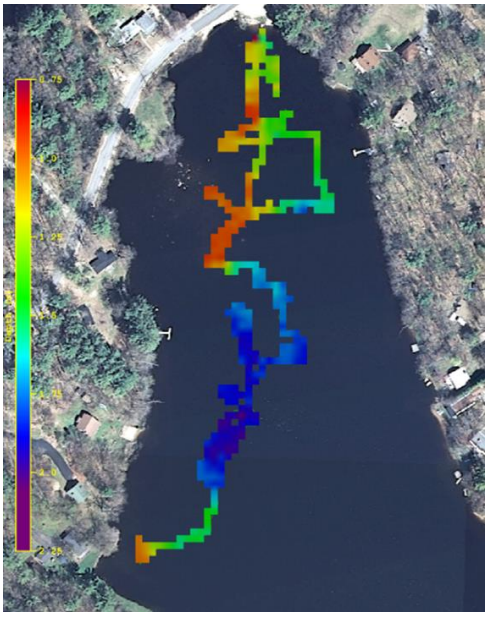


Figure 17 Sonar depth readings performed at Swain's Lake. Courtesy of Damian Manda.

Citations: [1]: *User's Manual for CEEPulse 100, Version 1.0.5.* Sydney, Australia: CEE Hydrosystems, n.d. Web.
[2]: "200 kHz 8 Degree Transducer Section." Photograph. CEE Hydrosystems, n.d. Web. 29 April 2016.

"200 kHz 8 Degree Transducer Mount." Photograph. CEE Hydrosystems, n.d. Web. 29 April 2016.

Sensor Mast

The mast on the ASV vessel is meant to hold the sensors which give feedback to the autonomous system. By placing the sensors elevated above the vessel, the sensors have unobstructed vision around the boat and can effectively transmit data. The original ASV mast was a set of PVC pipes that were adhered into the existing cup holders on the vessel with a bar of 80/20 aluminum reaching across the bow of the boat and connecting to the pipes. The

horizontal bar would hold the sensors and provide sufficient space for all the desired equipment and any more space for future sensors. With the mast securely fastened there were very little wind oscillations during testing, however, the adhesive used to secure the pipes had cracked when the ASV was brought back to the Chase building. This was an indication that the adhesive would not suffice as a long term means of securing the mast. Other options were explored in securing the mast to the vessel, with the goal to use what parts were available.

Two seats were provided with the boat and they were secured by aluminum frames that slid into slots along the center of the boat. The seats were held to the aluminum frames with J-bolts; these bolts could be used to secure a platform to the aluminum frame which would then be secured to the boat. The same pipe mast could then be secured to the new platform in a way that prevented permanent fixtures to the boat and would utilize existing materials. The drawbacks of this installment were the previous mast assembly was attached to cup holders that were positioned on the gunwales of the boat and had a more elevated position; with the new assembly the mast would have to sit on the deck for stability which would lower the position of the mast. As an alternative, the mast could be raised higher on the aluminum frame platform, at the cost of weaker construction and lowered resistance to wind oscillations.

During the production of the mast, some 80/20 fixtures became available and were then used in producing the mast. The 80/20 is an aluminum bar that have slots on each face and along the length that provide easier and sturdier assembly of a structure. With two bars of equal length and a third that would span the beam of the boat, brackets were used to secure the pieces together that can be dismantled for ease of transportation. To secure this fixture to the boat, holes were drilled into the boat such that brackets could attach the new mast to the boat. The holes were drilled near the center of the length of the vessel which provided a better clearance for the sensors over the boats hardware. Using toggle bolts, the 80/20 mast was secured to the gunwales of the boat where it has a flat stable position. Given the size and rigidity of the brackets, the new mast is stable considering it is heavier than the previous PVC mast.

The toggle bolts also provide a secure hold between the brackets and the vessel where the inside of the vessel is covered in foam to provide the vessel with some buoyancy if it were to take on water. The area of a regular bolt would not be sufficient in holding the foam material; the foam easily compresses under a compressive load, resulting in a weak connection. The toggle bolts have a much larger contact surface area, distributing the load more evenly. Near the same position of the mast are portholes on both sides of the boat that provided some access to the inside, allowing for the bolts to be securely tightened. With the new mast in place, the sensors can easily be attached and are given a stable platform for collecting data.

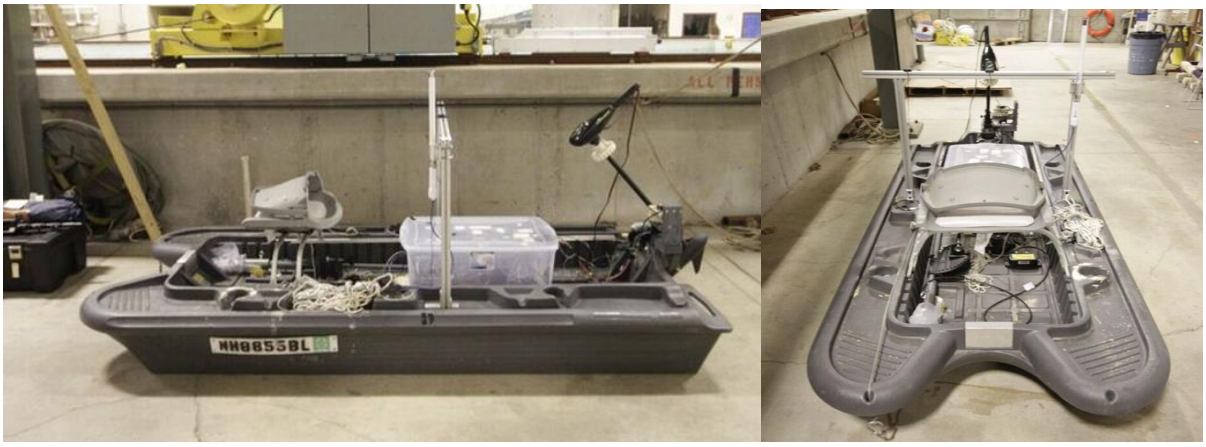


Figure 18: Finished ASV3 with Sensor Mast

ASV - ROV Communications

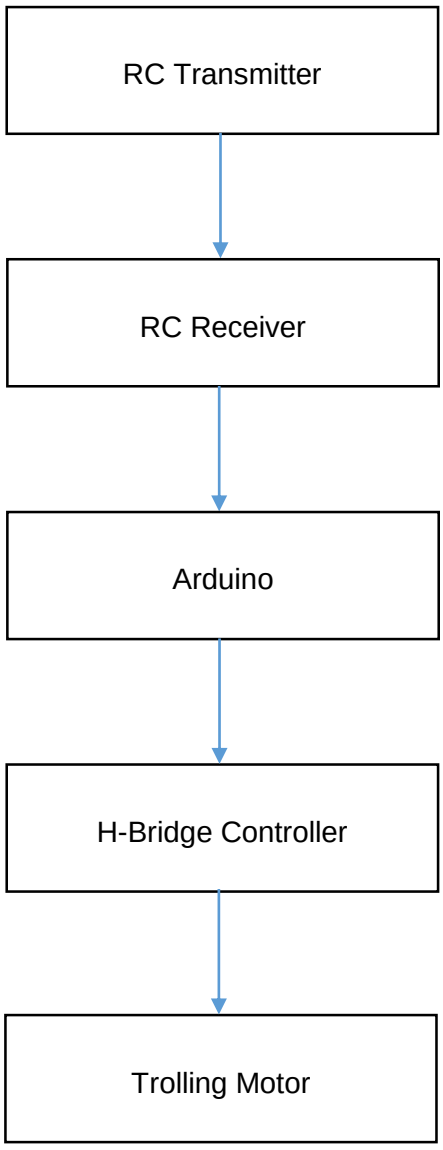
Work has begun on establishing basic communication between the ASVs and the ROV. Our plan is to have the ROV flash a light underwater while the ASV has a GoPro camera looking underwater and attempting to recognize the light. If the GoPro successfully identifies the light, the ASV will flash a light on top of its mast that mirrors the flashing of the ROV's light, therefore proving that basic communication has been achieved. In order for the GoPro to recognize the light and flash the ASV's light, the programs OpenCV and Matlab are being used in conjunction. OpenCV is useful for object identification and in this case the object would be a flashing light. Once the program successfully identities the light it will transmit data to Matlab which will in turn send a command to flash the light on top of the ASV. If everything works correctly we can begin to experiment with different intensities or colors of light and possibly transmit data in the form of Morse code.

Solar Panel

In order to charge the battery on longer voyages a trickle charge solar panel was purchased. Because it is just a trickle charger it does not provide enough power to keep the batteries running 24/7, instead it lengthens the battery life by an hour or two. Mainly the solar panel was purchased to test the idea of keeping the battery charged for longer voyages. Eventually a code can be programmed that tells the ASV to hold position for a certain amount of time when the battery is low so the solar panel can work to recharge the battery efficiently. Future years may look into this concept further and purchase larger and more powerful solar panels that could provide a large part of the boat's required power.

Wiring Diagram

Signal Flow Chart



Handheld RC transmitter sends signal based on user input. Signal is used to send information about desired throttle speed and direction.

Receiver interprets signal as a PWM signal with a pulse length from 1000 to 2000 microseconds. Receiver is wired to an input pin on the Arduino.

Arduino reads signal through interrupt routines. Translates signal length to numerical value. Uses numerical value to determine proper action and sends signal to H-bridge DC motor

H-bridge controls motor by directing power from battery to positive or negative leads of motor. 5-volt signal from Arduino is sufficient to control MOSFET transistors directing up to 30

Brushed DC trolling motor able to run in two directions and responds to PWM input.

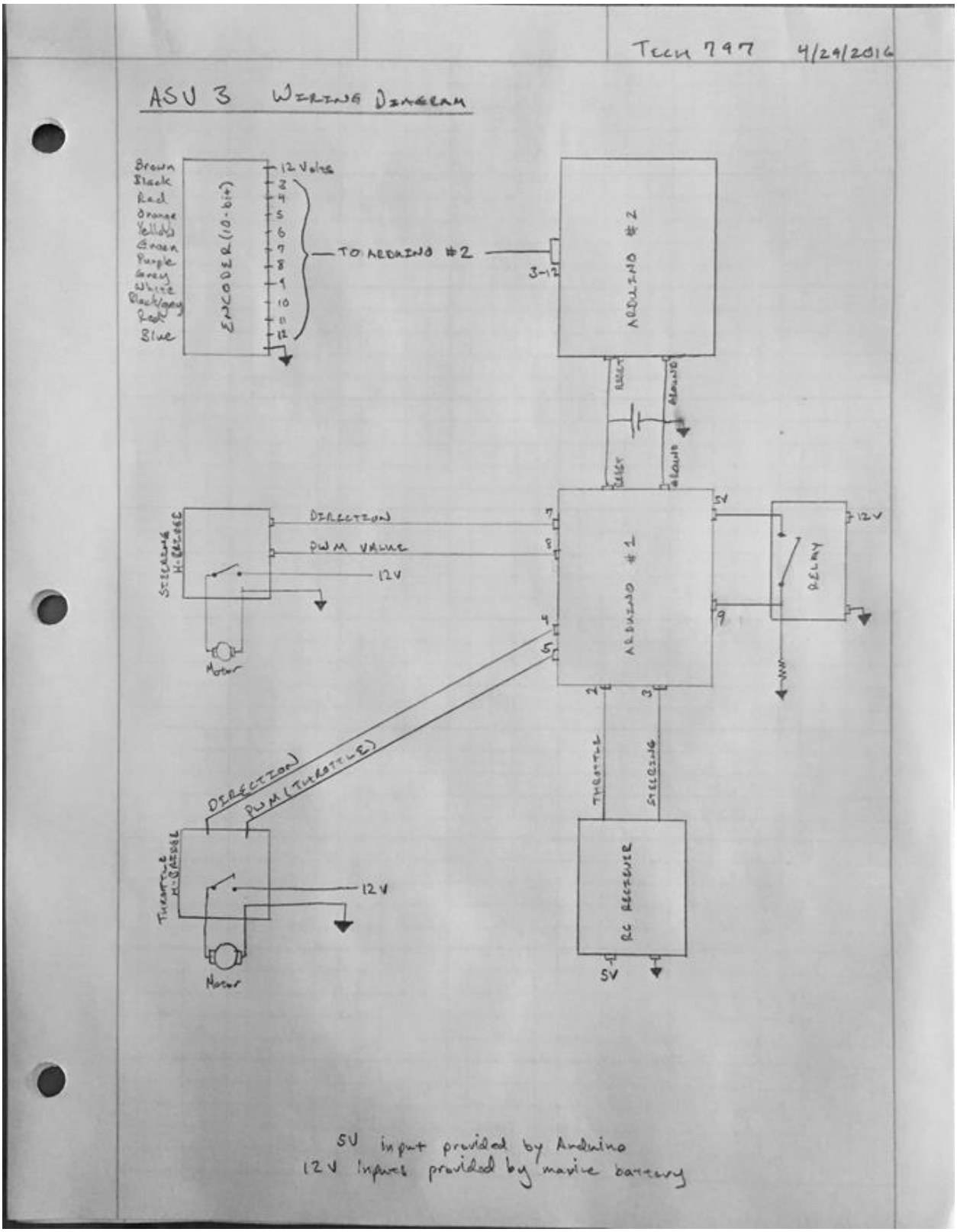


Figure 19: Wiring Diagram

Clockwise, from top left:

Encoder:

A 10-bit absolute encoder was chosen to provide position feedback to the control code. The encoder is mounted to the window motor and operates on the assumption of no slip between the gear attached to the window motor and the gear attached to the shaft of the trolling motor.

Arduino # 2:

The second of the two Arduinos is responsible for reading the input from the absolute encoder and translating the 10-bits being read into a usable integer value. This value is shared with Arduino # 1, where is used to calculate the error signal in the control code.

Arduino # 1:

The first of the two Arduinos is the hub of all activity on the ASV. In addition to communicating with Arduino # 2, Arduino # 1 controls the relay, receives RC input, and sends PWM signals to each of the h-bridge motor drivers.

Relay:

The relay acts as a toggle between RC and Autonomous modes. Depending on if the switch is open or not, pin nine can be at 5V or pulled to ground. This pin is read in the main code as an interrupt.

RC receiver:

When in RC mode this receiver sends a PWM signal to input pins on the Arduino, this input is translated into a direction and voltage and sent to the h-bridges.

H-bridge motor drivers:

Use of h-bridges allows a low power signal from the Arduino to control the high current flow between the marine batteries and the electric motors aboard the ASV.

Software

The software behind the UNH ASV team's autonomy can be broken down into three main components of discussion. First is the autonomy software itself, which allows for the boat to make sense of all it's different sensor values and make decisions using those values. Second is the boat control code, which allows the boat to take commands from the autonomy system and actually translate those commands into physical rudder and throttle behaviors, as well as allows the boat to have autonomous and remote control modes. Thirdly is the sensor suite itself, which involves all of the ins and outs of the different sensor hardware, the roles they play in the bigger picture of autonomy, and the formats they output data in and that this data is then interpreted by the autonomy system.

-Autonomy

Firstly, to understand the software side of the project, one must have a basic understanding of the autonomy software itself. The UNH ASV team achieves autonomy using an open source package that was developed jointly by MIT and Oxford: MOOS-IvP. MOOS stands for Mission Oriented Operating Suite and IvP stands for Interval Programming. If the ASV is considered as an organism, the brain of this organism is the raspberry pi 2 microcontroller (or laptop computer depending on system size) and the MOOS is what allows the brain to work. Imagine MOOS to be all of the neurons within the brain of the ASV. MOOS can be called a "publish and subscribe" software package, because it essentially is organized as a tree of numerous different applications that publish information to one another and subscribe for information from other applications to allow themselves to run. Graphically, this is shown in Figure 20 below:

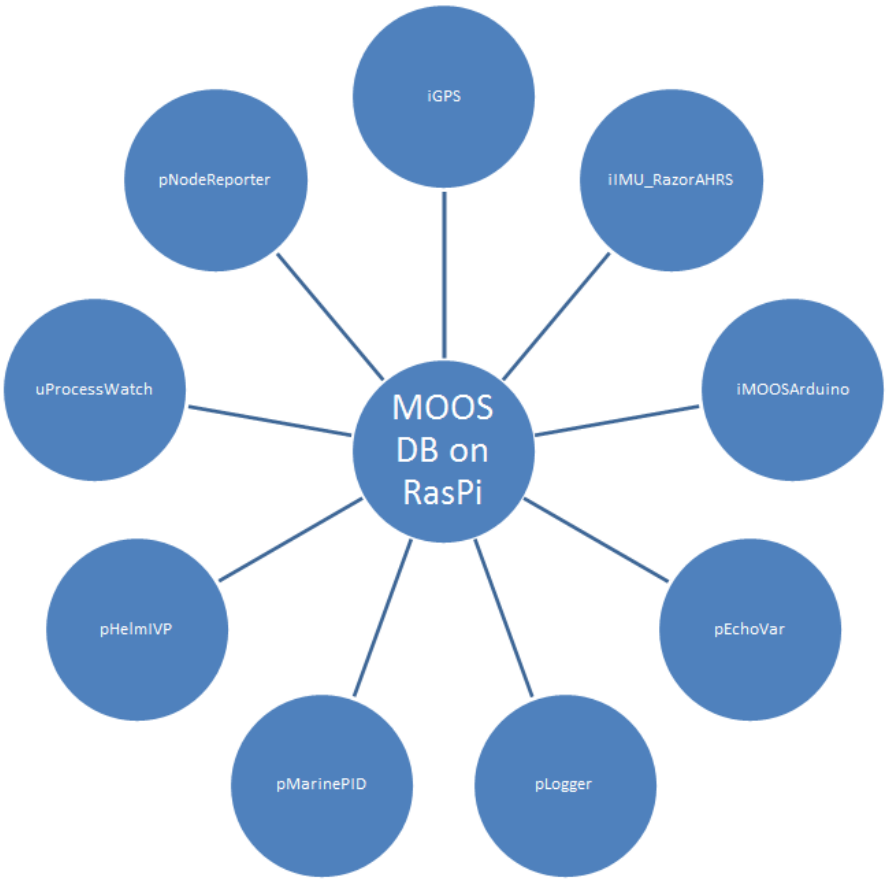


Figure 20: MOOSDB Sensor Graphic

The central application to all of this is the MOOS database (MOOSDB). This application is simply a place for all other applications to dump their information to, and it is where all applications look for the needed input values they subscribe for. The MOOSDB is the "central neuron" (continuing with the brain analogy) that connects all others. The other applications ringing out around the MOOSDB are theoretically infinite in number and in their purposes. There are a host of applications that come with the initial download of MOOS from online, and MOOS users can

write their own MOOS applications to accomplish whatever tasks they desire. The essential MOOS applications that are used when running basic missions on ASV 3 and their purposes for ASV3, at the time of this writing, are listed below. Note, applications marked with a star were developed at UNH. All others come with the initial download of MOOS-IvP from online. Also note, purposes/ functions of all the MOOS apps below are not fully understood by the ASV team at this juncture.

- MOOSDB-- MOOS database where all moos apps publish and subscribe to
- uProcessWatch-- Monitors processes that the autonomy system is conducting; either says the processes are present and functioning, or flags the processes for running slowly, encountering errors, or not being present at all despite being expected
- pNodeReporter-- " Captures vehicle state information and publishes a summary string" -oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php?n=Site.Modules
- iGPS-- interfaces with GPS (global positioning system) sensor and sends GPS_X, GPS_Y, GPS_Lat, GPS_Long, and GPS_Speed values to the MOOSDB
- iIMU_RazorAHRS-- interfaces with IMU (inertial measurement unit) sensor and sends YAW, PITCH, and ROLL values measured by IMU to the MOOSDB
- iMOOSArduino**-- MOOS app developed by ASV1 team. Takes "Desired_thrust" and "Desired_rudder" values from MOOSDB and sends them to control code on Arduino.
- pEchoVar--takes variables specified by user (in our case, GPS_X, GPS_Y, GPS_Speed, YAW) and renames them as specified by the user (in our case, to NAV_X, NAV_Y, NAV_Speed, and NAV_Heading respectively). This is used by ASV3 to rename values input by sensors to names that are expected by the stock MOOS applications.
- pLogger-- MOOS app that logs the values and results of an entire mission along with timestamps of when events occur so that the mission has an automatically generated and easily analyzed record file
- pMarinePID-- built in PID controller that is used by the MOOS decision making helm to determine, based on its known position, speed, and heading, what it's desired speed and heading values are
- pHelmIvP-- this is the decision making helm for the MOOS-IvP software. Amongst other things, this App looks at a mission file and the behaviors desired for that given mission, as well as all relevant sensor data, and determines what actions to take and commands to write to the rest of the system. Two key variables it publishes are "Desired_Speed" and "Desired_Heading" which are then taken by iMOOSArduino to the boat control code
- pShare-- MOOS application for sharing information about the running MOOSDB with other MOOSDBs. This is critical for having multiple boats that are running MOOS talking to one another.

- pHostInfo-- MOOS application for sharing with other MOOSDBs more general information about the host MOOSDB it is running on.
- uFldNodeBroker-- MOOS application for sharing information with other MOOSDBs about the host MOOSDB it is running on
- pBasicContactMgr-- MOOS app responsible for monitoring other MOOSDBs sharing information with its host and working to ensure there are no collisions between its host and these other vessels.

As can be seen from this list, even running basic MOOS missions requires coordination of numerous MOOS applications all of which are accomplishing a wide variety of functions. Some of these functions are internal to the ASV's processing, such as pLogger, uProcessWatch, or pHelmIvP. Others are essential for interfacing with the boat's sensor suite and reading in sensor data, such as iGPS and iIMU_RazorAHRS. Others still are essential for interacting with other vessels that are running MOOSDBs.

This is the general architecture of the MOOS-IvP software stored on the Raspberry Pi 2-B microcontroller of ASV3. In the future, the team would like to use a laptop rather than the Raspberry Pi, as higher levels of autonomy will quickly demand more processing power and speed than the Raspberry Pi can easily produce. How does the ASV team actually configure what MOOS applications they want to be using in a given mission, or what tasks they want the boat to accomplish during said mission? That is done using a number of essential files. The raspberry pi on ASV3 runs a linux/ raspbian debian wheezy operating system. In a central file location which the ASV team has specified for its MOOS missions, four essential files are used to configure a mission. Step one in mission configuration is to determine the area the team wishes to operate the vessel in. By going onto google maps, a snap image can be taken of a satellite image of a large area (such as a lake). This image is then saved in a .tif file format in the given file location. Still using the google image online corresponding to the .tif image just taken, the GPS latitude and longitude of the extreme northern, southern, eastern, and western boundaries of the photo are recorded in a .info file with the same name as the .tif file.These values are used by MOOS to automatically divide the .tif file into both a global lat/long grid and into a local XY coordinate grid.

These files are needed to run the MOOS GUI for observing the boat on a shore side computer during mission operations, and are not used for any other purpose other than giving the shore side team the GUI for monitoring and prompting the boat during operations. These files are therefore not stored on the microcontroller on the boat (as the boat is not running the GUI; there is no-one on the boat to look at the GUI!). The last two files configure the actual MOOS mission itself, and they are the .MOOS and .BHV files. The .MOOS file, amongst

numerous purposes, names the mission, specifies the initial latitude and longitude location of the mission origin (immediately updated by the GPS upon mission launch), specifies what MOOS apps are running during the given mission, specifies where the mission behavior files are to be found, and configures essential parameters for every MOOS application. These parameters vary from application to application (for example specifying the PID gains for pMarinePID, specifying the ports that the GPS, IMU, and Arduino are located at etc). Also specified for each application are the number of times per second applications are run and that applications publish to the MOOSDB and subscribe for information. This aspect is of critical importance, because if the applications are not all timed correctly a mission can quickly breakdown for numerous reasons, from applications looking for information prior to applications they depend on publishing that information, to running so many times per second the rest of the mission is starved of processing power. The .BHV file specifies the different behaviors/ tasks the boat will be running during the mission. These behaviors, similar to with MOOS apps, are theoretically infinite in the number that can be run and their functionality. Stock behaviors exist with the download of MOOS, such as the avoid collision behavior for avoiding crashes with other MOOS-running-vessels or the waypoint behavior for doing basic waypoint navigation. Custom behaviors can also be written, such as an ocean mapping behavior, an aggressive swarm and defend behavior, or a search and rescue behavior. Also similar to the MOOS apps, the meat and potatoes of the behavior/ application functions are not contained within the .BHV/.MOOS files. These files contain essential configuration information, but the behaviors and apps themselves are defined elsewhere in the system. In the .BHV file specifically, the configuration information for each behavior varies, for example the waypoint behavior allows for specifying what points the boat should visit, how close to a point the boat must get to consider itself “arrived”, etc while the avoid collision behavior requires specifying how far away to keep other vessels, etc. All behaviors require inputting a priority weight, however, which tells the MOOS decision making helm which behavior is the most important one if faced with a decision where multiple behaviors it is running conflict with one another. For example, a boat running the collision avoidance and waypoint behaviors during its mission is running both behaviors constantly, but if collision avoidance is weighted higher and another vessel comes dangerously close, the boat will engage in avoiding that collision over continuing along its planned waypoint path, and will not resume the waypoint path until the collision is avoided. This concludes the discussion of the Autonomy helm as a whole.

-Arduino Control Code

Continuing the analogy of the ASV as an organism, if the Raspberry Pi or Laptop containing MOOS and the sensor drivers is the brain, then the Arduino Mega 2560 microcontroller is the nervous system. This microcontroller receives input of desired heading and desired thrust values from either the autonomy helm or the user via remote control,

converts those values to a pulse width modulated (PWM) signal, and then sends those values where they need to go in order to influence the steering and throttle motors. The throttle aspect of the code is relatively straightforward, initial calibration of the RC system was done early in the year to determine what PWM values received from the RC receiver on the boat corresponded to what degree of trigger pull by the human captain on the RC controller. These PWM values are then mapped to produce a given speed output by the trolling motor, and are sent to an h-bridge controller. The h-bridge acts as a switch within the motor's circuit, allowing or denying current based on a low power control signal. Whenever this signal from the Arduino is set high (5V) the h-bridge will allow the current to flow from the battery to the motor. Two h-bridges were used in this iteration of the ASV, a 30 Amp h-bridge for the trolling motor (throttle) and a 2 Amp version for the window motor (steering).

-Sensor Functionality

The last major subject to discuss regarding the software side of ASV3 is the raw sensor output signals and the work done with forwarding higher level autonomy utilizing experimentation with higher level sensors.

The GPS and IMU sensors on ASV 3 function in what is essentially a plug and play fashion. The only calibration necessary for these sensors involved calibrating the accelerometers, gyroscopes, and magnetometers on the 9DOF SparkFun razor IMU. This IMU has a built in Arduino pro mini microcontroller that internally processes its raw data values. In MOOS, in the iIMU_RazorAHRS app, the sensor output format and specific data desired can be specified, and for our case this desired data is simply the YAW about the z-axis of the vessel (used to determine heading). The GPS_X,GPS_Y, and GPS_Speed values are piped directly in using the iGPS moos app without any calibration or processing, and these are used to determine the position and speed of the ASV.

These two sensors alone are sufficient for obtaining point to point navigation and for running the collision avoidance behavior in MOOS. Remember, collision avoidance is avoiding collision with another vessel running MOOS/ a vessel that tells the ASV where it is and where it is going. This is very different than obstacle avoidance (second stage autonomy) which involves the ASV detecting an obstacle on its own (without the obstacle saying it is there) and then adapting its waypoint planned path to avoid said obstacle. The sensor used to accomplish obstacle avoidance is a Light Detection and Ranging system (LiDAR). LiDAR works by mounting a laser beam atop a spinning, encoded platform. As the laser spins, the beam goes out and bounces off of obstacles, and based on the phase of the laser when it returns to the sensor the sensor determines the distance to the particle. Based on the moment that the laser beam is received back in by the lidar, the system also knows at what angle the beam was fired out, and therefore also knows the angle off of the boat to the given obstacle. Currently, the ASV team has worked with a low cost Neato-XV11 LiDAR system which will be implemented on ASV2

(purchasing a system for ASV3 was not within this year's existing budget). The first major step to implementing obstacle avoidance has been attempting to understand the format that the LiDAR uses to report data, and from there working to find or code a script that will essentially create an occupancy grid about the boat and use LiDAR data to constantly update this grid with the presence of obstacles. Up till the time of this writing, the ASV3 team has only read data from ASV2's LiDAR on an Ubuntu laptop computer. At first, a python script for reading in LiDAR data and producing a GUI of the LiDAR was used to help conceptually understand the values being read. Quickly, however, the team found that python was far too slow computationally to handle the massive amount of data the LiDAR was generating. The team this year found a ROS (Robot Operating System) script that had been developed for the same LiDAR as ASV2. ROS is a software platform that is far more widely used than MOOS-IvP, and while MOOS is stronger for developing autonomy, ROS is more widely used therefore more support and question forums exist making it more user friendly. It was also quickly found when running the ROS LiDAR script that ROS was far more computationally efficient at processing the LiDAR data and running the LiDAR GUI than python. An image of the GUI produced by the ROS script can be seen in Figure 21 below:

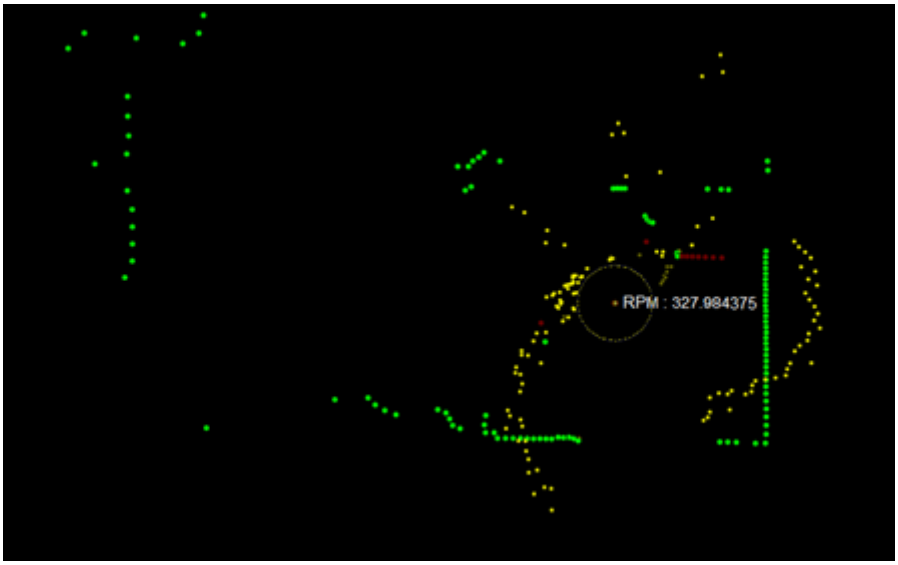


Figure 21: LiDAR GUI

The current work being done by the ASV team at the time of this writing is working to figure out the following: how can LiDAR values be piped from ROS, which can handle the large data volume in a computationally efficient manner, into MOOS and made available to the autonomy system? If LiDAR data is piped directly into MOOS, what existing drivers are there for obstacle avoidance, and how could they be adapted to utilize LiDAR data? If no existing stock MOOS applications work for using our specific LiDAR, our CS student has been considering an obstacle avoidance code that could be written either within MOOS as a MOOS application, or outside of MOOS that could pipe final decisions regarding obstacle locations and needed autonomy helm

actions into the MOOSDB. This specific area of work, despite being focused on ASV2's LiDAR and developing second stage autonomy on ASV2, is directly applicable to the future of working with any LiDAR to do obstacle avoidance on any MOOS vessel in UNH's ASV fleet. More specific details about the logic behind the potential obstacle avoidance code can be obtained upon request.

The other area of major development regarding software for the sensor suite is on the image processing front. The team this year began delving into work on to enable the target track and trail behavior. MOOS has a stock track and trail behavior built in for recognizing another vessel running MOOS as a target, however the team has no experience running this behavior and is in the process of working to understand what applications will enable its functionality and how so. As this research has continued, the team is greatly interested in developing a track and trail behavior for recognizing targets that may not be running a MOOS database and reporting their information to a vessel in the fleet. To do this, the first step by the team has been looking into the way in which a computer loads in images from a GoPro camera (our camera of choice for ASV3) various programs for image processing. Among these programs: two open source software have been considered (OpenCV and Predator) as has Matlab. Currently, the team is developing an image processing script to use in providing proof of concept of ASV-ROV communication. Using Matlab imaging processing software and a GoPro camera, ASV3 will monitor the ocean beneath it. The ROV will drive beneath the ASV and flash an incredibly bright light in a pattern, which the ASV will observe and then duplicate using a light on the surface. This project goal serves a twofold purpose. First of all, it is to prove that an ASV and ROV can communicate with one another/ pass information in some basic form. Long term, acoustics are the ideal and practical form of communication, however due to budgeting issues the team this year could not afford an expensive acoustic transducer, and therefore light is an inexpensive means for achieving proof of concept in the current academic year. Secondly, using light allows the team to conduct research and gain experience in doing image processing to detect a "target" and make decisions based on the actions of that target. This experience will prove invaluable for applying image processing to recognizing and reacting to targets on the surface. At this juncture, a Matlab code has been developed that is capable of picking a target out from a known backdrop, and interpreting information regarding target size and velocity from that information. The key now is strengthening this code so that it will be more applicable when some characteristics are not known about the background (such as the characteristic length information of pixel size to physical dimension needed to interpret target size and velocity), dealing with target motion not just perpendicular to the vessel, but also at angles to the vessel, and getting Matlab to make decisions based on the observation of the target (such as turning a surface light on or off for a given time period). This concludes discussion of the current software aspects surrounding the sensor suite.

Applications/ Importance

ASVs have almost unlimited applications in the maritime theater. They can be used for a variety of applications ranging from ocean mapping to search and rescue to swarm defense. For example, if the National Oceanic and Atmospheric Administration (NOAA) had a fleet of ASVs equipped with ocean mapping equipment, they would be able to send each ASV to a different location and cut down on their mapping time and costs significantly. The Navy is very interested in autonomous vessels for both search and rescue and swarm defense. When they need to rescue someone in the water, they can send in an ASV instead of endangering other sailors lives by sending them to rescue the person. A swarm of ASVs could also be programmed with a swarm defense behavior to deter any threats to Navy ships or any other ship that runs through dangerous waters. ASVs are ideal for potentially dangerous missions, as they remove the human element from the whole operation.

Holloway Competition

This year our team entered the Holloway Prize Competition, an idea to market competition within the University System of New Hampshire. The goal of our company, Aquatic Autonomous Solutions, was to provide customers with low cost autonomy for their vessels in the form of a “black box” control module. This control module would be able to interface with any vessel, from a small fishing boat to a full size ocean liner, and allow it to have full autonomous capabilities. The aspect of our product that set us apart from other companies is that they don’t have to purchase a new autonomous boat. Our control module would interface with their existing vessels and would save the customer tens of thousands of dollars.

Our team placed second in the Nelson Poster Competition and third in the Bud Albin semifinal round. Although we did not make it to the final round, many judges expressed interest in our product and encouraged us to continue to working on our business plan.

Conclusion

This year’s team has been the most successful out of the previous 2 years. Full first stage autonomy was achieved on ASV2 as well as ASV3, and lots of progress was made with the other sensors such as the LiDAR and the sonar. We created a large proof of concept in the form of ASV3 and proved that the modular and component of the shelf philosophy of our team can be applied to larger vessels. We began to delve into future applications such as ocean mapping with the sonar tests, and laid the groundwork for future teams to continue developing obstacle avoidance and object tracking and trailing.

Appendices

Works Cited

Cao, Yang. "Modeling an Equation Driven Involute Spur Gear in Solidworks." Online video clip. *Youtube*, 23 Feb 2014. Web. 7 Feb 2015.

Simpson, Ken. "Remote Electric Steering for Trolling Motors." *Duckworks Boatbuilder's Supply.* N.p, 1 November 2010. Web. 21 October 2015.

User's Manual for CEEPulse 100, Version 1.0.5. Sydney, Australia: CEE Hydrosystems, n.d. Web.

"200 kHz 8 Degree Transducer Section." Photograph. CEE Hydrosystems, n.d. Web. 29 April 2016.

List of Figures

Figure 1

Figure 2

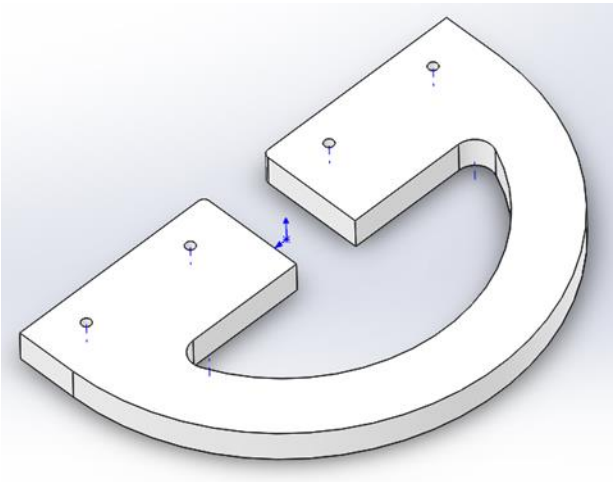


Figure 3

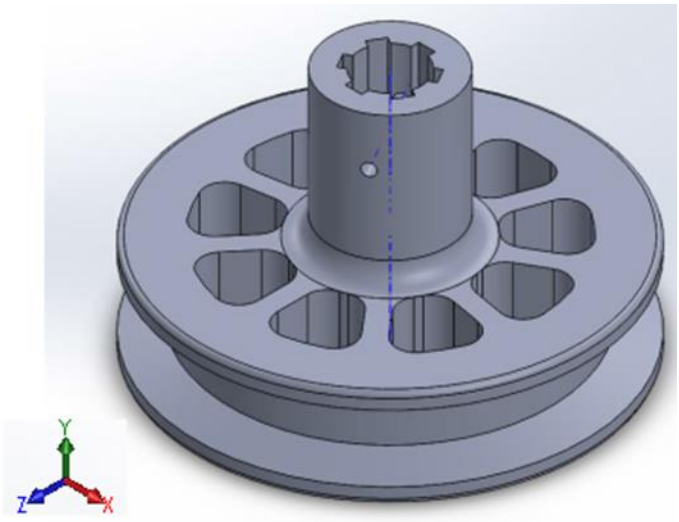


Figure 4

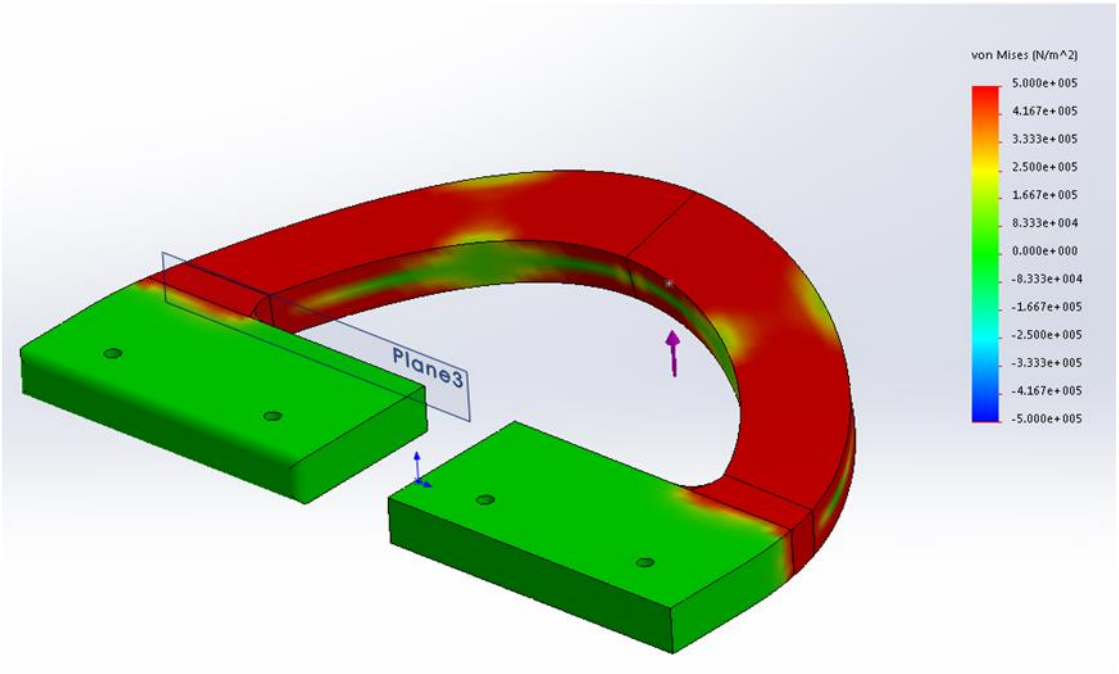


Figure 5

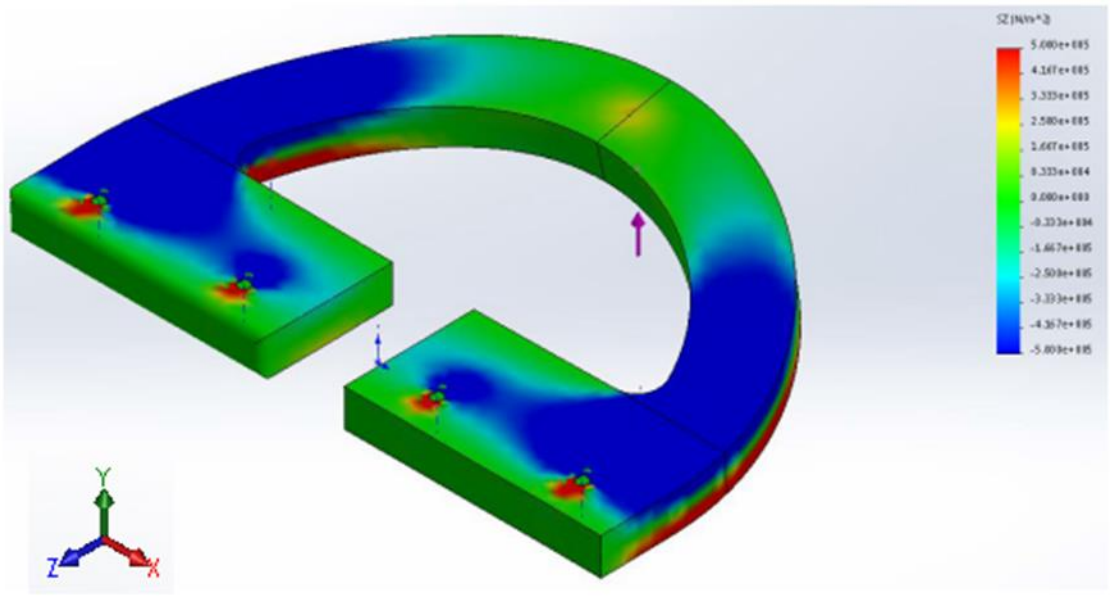


Figure 6

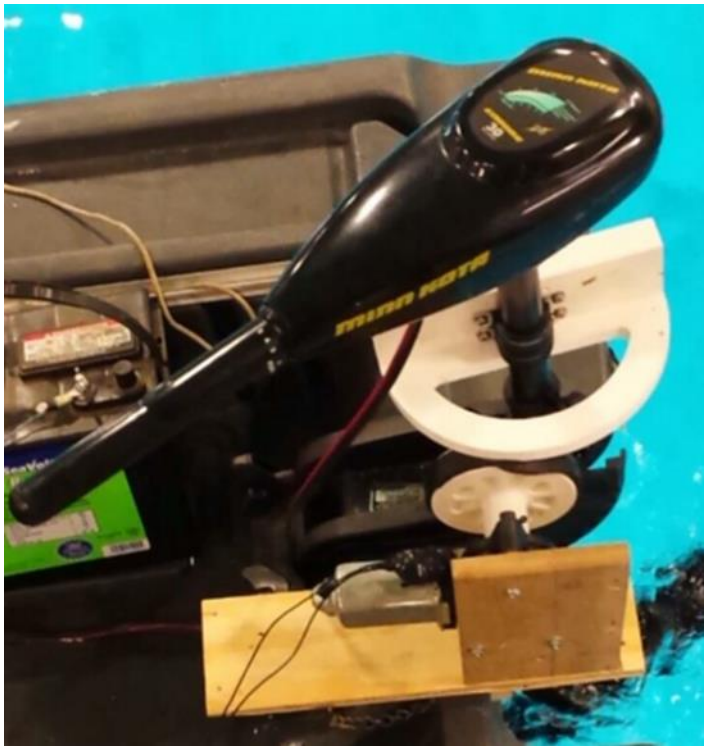


Figure 7

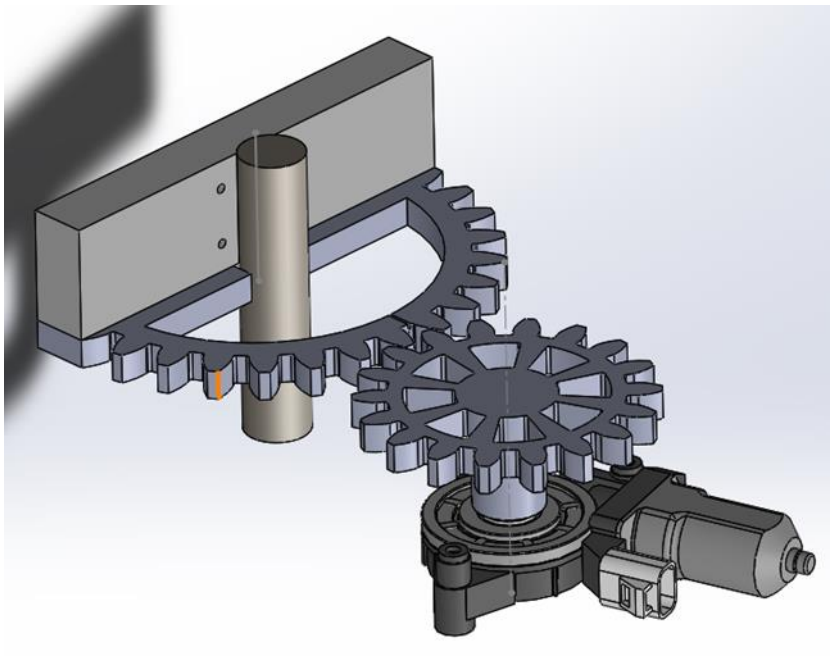


Figure 8

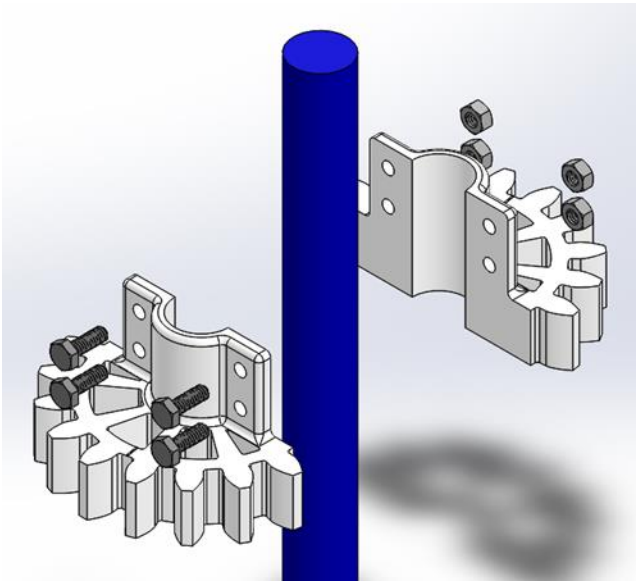


Figure 9

Figure 10

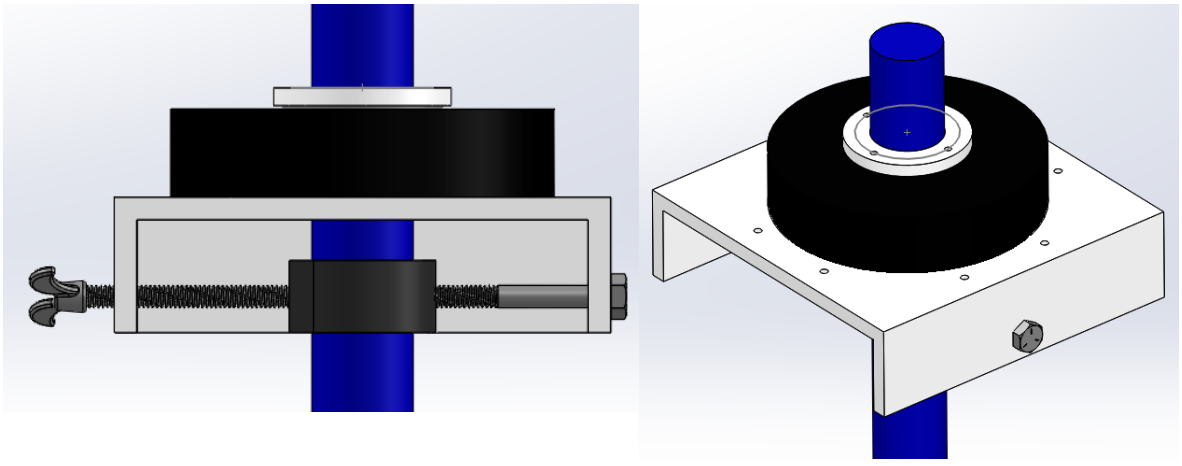


Figure 11

Figure 12

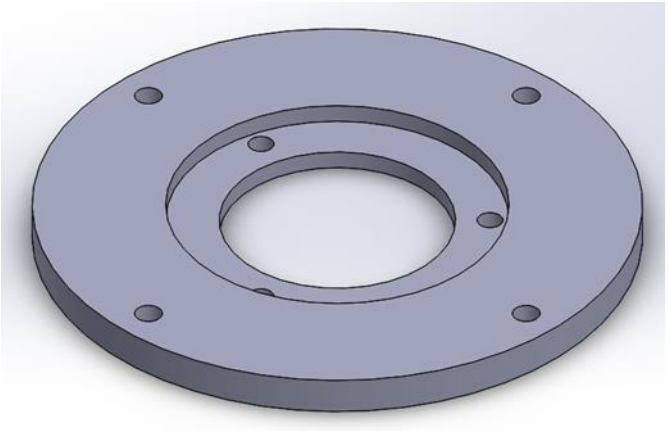


Figure 13

Figure 14

Figure 15

Figure 16

Figure 17

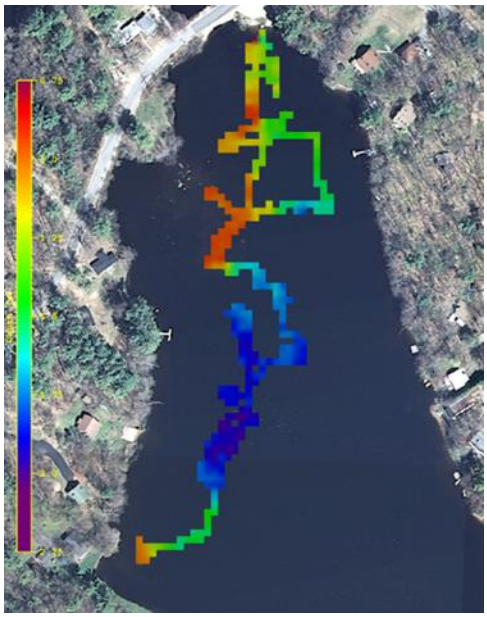


Figure 18

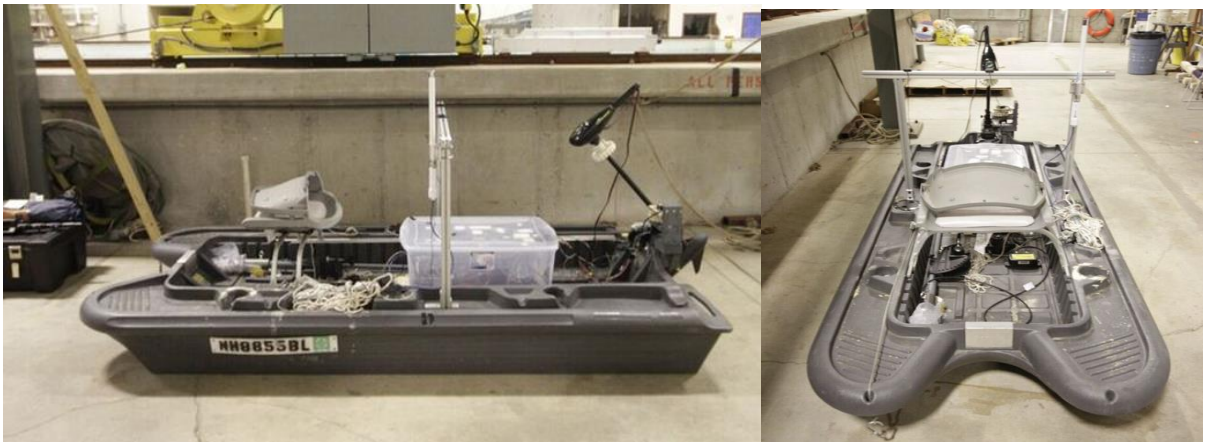


Figure 19

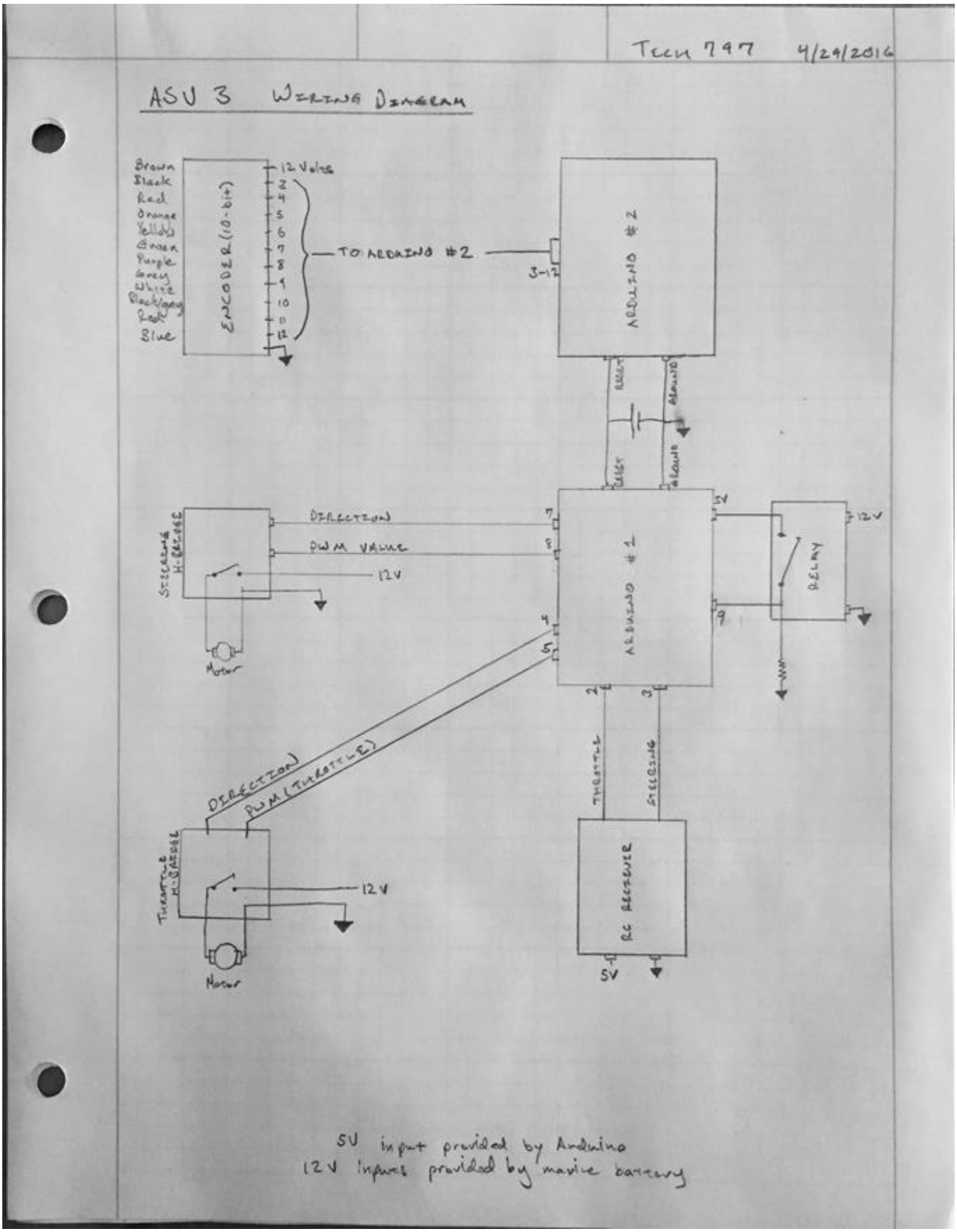

Tech 797 4/29/2016
ASV 3 Wiring Diagram
Brown
Black
Red
Orange
Yellow
Green
Purple
Grey
White
Black/gray
Red
Blue
ENCODER (10-bit)
12 Volts
TO ARDUINO #2
ARDUINO #2
DIRECTION
PWM VALUE
12V
STEERING MOTOR
Motor
ARDUINO #1
5V
RELAY
12V
DIRECTION
PWM (THROTTLE)
THROTTLE MOTOR
12V
Motor
THROTTLE
STEERING
RC RECEIVER
5V
5V input provided by Arduino
12V Inputs provided by marine battery

Figure 20

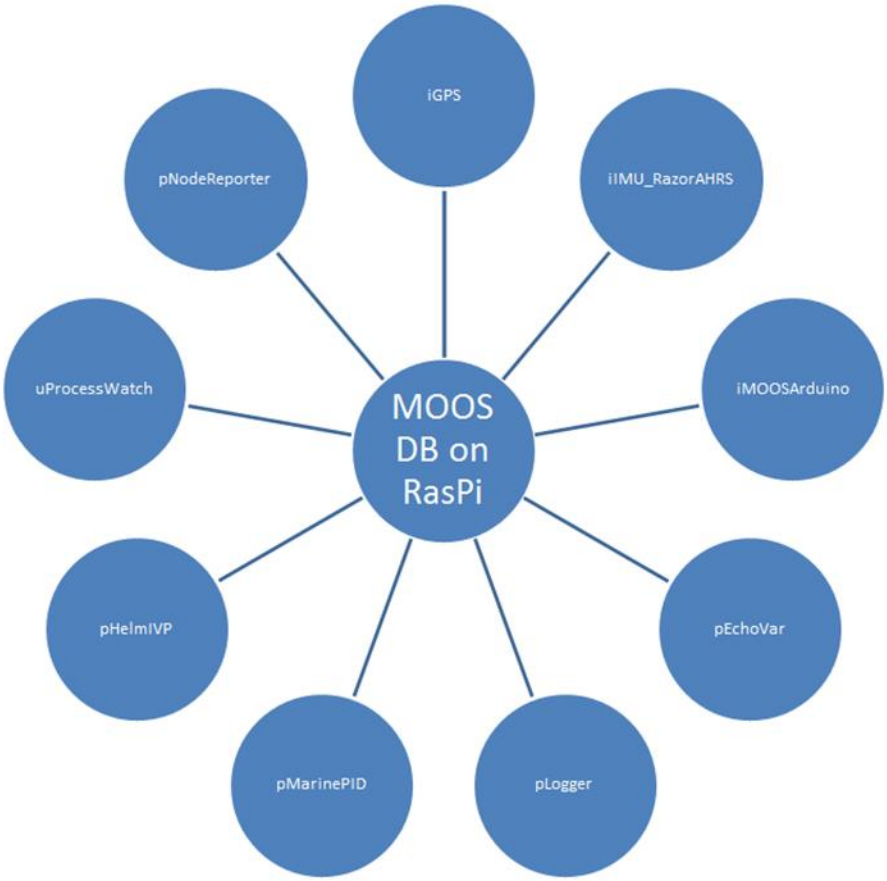
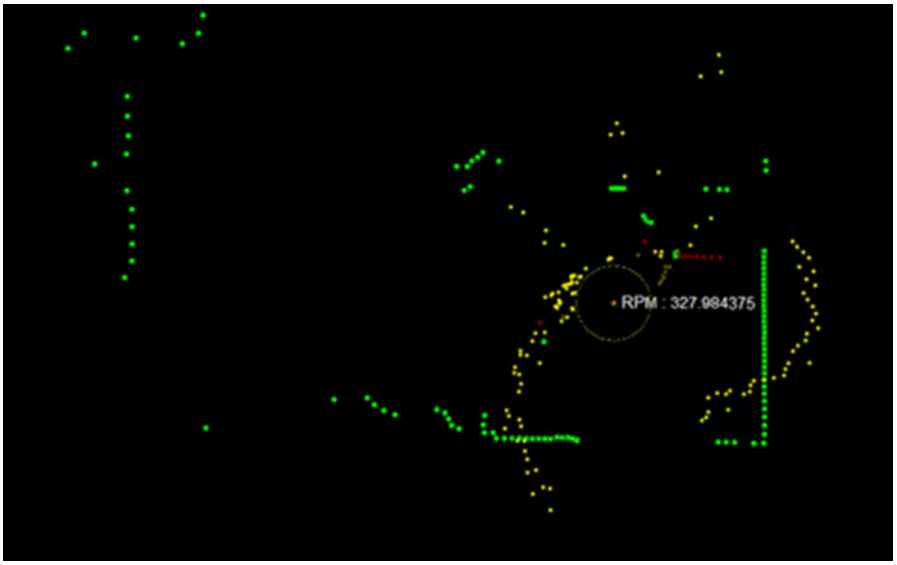

iGPS
pNodeReporter
iIMU_RazorAHRS
uProcessWatch
MOOS DB on RasPi
iMOOSArduino
pHelmIVP
pEchoVar
pMarinePID
pLogger

Figure 21


RPM : 327.984375