Autonomous Surface Vehicles

Authors: Forrest Walen, Justyn Sterritt, Paul Willis, Andrea Dargie, Philip Goff, Lucas Davies, Ben Novak

Project Advisors: Professor May-Win Thein, Damian Manda, Andrew D'Amore

Sea Grant
New Hampshire

Table of Contents

Abstract

Autonomous Surface Vehicles (ASVs) continue to be in high demand especially in situations where human involvement is not desired. These situations include hazardous environments, general surveillance, as well as search and rescue missions. This project aimed to utilize the necessary hardware and software components for multiple levels of autonomy to construct a proof of concept autonomous surface platform for use in future research.

There are three levels of autonomy that the team aimed to achieve. The first level is waypoint navigation: moving the boat from an initial position to a specified point without an operator. This mission uses MOOS-IvP to process this information and direct the boat. MOOS-IvP is an open source computer software platform designed specifically to work with autonomous feedback control for marine vessels. The second level of autonomy is obstacle avoidance: rerouting the boat to move around an object while still reaching the final specified point. This level of autonomy uses a LiDAR object detection system to work with the navigation system and guide the boat safely around obstacles. The third level of autonomy track and trail, involves tracking a physical object, such as another vehicle, or following a predetermined path. This level of autonomy uses both navigation and object detection. Once these levels of autonomy have been achieved the future goal of this project is to build multiple ASVs that can communicate both with each other and Unmanned Underwater Vehicles (UUVs). The current ASV uses a BeagleBone Black to communicate instructions to an Arduino Mega 2560, which relays the commands to the motor and rudder. Simulations were used to test the logic and processes of the MOOS-IvP behaviors. They also helped verify that the boat dynamics and sensor readings stored within the MOOS database (MOOSDB) were correct. These tests, simulations, and models provided the foundation for building the stages of autonomy. The team achieved basic simulated waypoint navigation in a pool test and developed the fundamental elements for obstacle avoidance.

Purpose

The project to construct an autonomous surface vehicle was guided by two principle purposes. The first of these two principles involves the importance of autonomy to society in general while the second of these principles is more focused on research that is being done at the University of New Hampshire.

By taking advantage of newer and more powerful embedded systems platforms such as an autonomous surface vehicle can perform tasks in place of humans. There are two main reasons that replacing human effort is important. The first reason for using autonomous vehicles to complete objectives is that certain tasks have a cost of human lives: most notably search and rescue operations in dangerous waters. By deploying an autonomous unit instead of a human one the risk to human rescuers is eliminated. An autonomous vehicle can be replaced, human life cannot. The second major concern that is addressed by autonomy is that certain tasks are simply too inefficient for humans to perform. One example of such a task that is more efficient for a machine to complete is that of ocean mapping. Building or purchasing an autonomous unit incurs a one-time cost and machines produce results with repeatability, absent human error. These two reasons form the basis of what makes autonomy such an important field in present and future research.

This project also serves as a platform for research being performed by a UNH graduate student. The graduate student will utilize the single ASV produced by this year's senior project team along with platforms produced by future teams to build a swarm system that incorporates the efforts of multiple autonomous vehicles working together towards a larger task than each individual unit would be responsible for. One such task could be a security net of autonomous vehicles: forming a protective border in the waters of a given nation to detect approaching vessels both above and below the surface of the water. Such capabilities make the research of this graduate student very important.

Goals of Project

The overarching goal for this year's ASV team was to create a marine platform to run a given task without any external assistance. This task was to be achieved within a reasonable budget. A reasonable budget in this case was $2000: given by the TECH 797 course and Mechanical Engineering Department at the University of New Hampshire. This task was also needed to be completed by an interdisciplinary team consisting of Mechanical Engineers, an Electrical Engineer, and a programmer.

The first step in achieving this goal was to choose the physical platform that will accomplish these tasks. Several parameters were considered when choosing the right platform: cost, stability, size, repeatability and ability to fit the equipment needed to achieve autonomy.

The first autonomy goal that was set to be achieved was waypoint navigation. This was chosen as the first goal because it is the most basic form of control. This goal involved being able to tell the platform to go from a current position to a set waypoint. To achieve this goal the necessary sensors and microprocessors needed to be chosen based off of the compatibility with the autonomy platform.

The second autonomy goal was to achieve obstacle avoidance. This goal involved the platform being able to detect an obstacle and simultaneously adjusting its course in order for it to avoid the obstacle while maintaining waypoint navigation. In terms of hardware a sensor that acted as an "eye" for the platform needed to be chosen based on its cost and compatibility with the system. This goal was chosen to be second because when coupled with waypoint navigation the platform can navigate itself freely.

The third autonomy goal that was set to be achieved was track and trail. This goal involved the platform identifying an object and then following it using a certain distance. The platform needed to identify the speed and position of the object in relation to its own position. This stage also needed an "eye" that would be able to identify an object and its position relative to the platform. This "eye" however needed to be different that the one that was needed in stage two.

Components

Boat

The autonomy platform used for this project was the Atomik ARC 58" remote control boat. It was chosen as the best option based on the goals of this team. It has one motor, one propeller, and one rudder. This made it easy to solve for the dynamic parameters of the boat. This also make it easier to program; one rudder and propeller simplifies the controls as the multiple rudders and motors would need to be synchronized correctly. The boat's style is a large catamaran which allowed the team to mount all the sensor units without the worry of the boat becoming unstable. The last requirement for the team was that this boat is battery operated and does not have a combustion engine. Most tests were run indoors and the sound level and exhaust fumes would have been undesirable. Having an electric motor also made it easier to integrate microcontrollers as the electrically controlled mechanical systems were already in place.

Figure 1-Boat used as ASV platform.

GPS/IMU

A Global Positioning System (GPS) module was needed in order to help the boat locate itself and navigate. The GPS module that was chosen was the Adafruit Ultimate GPS Breakout. This GPS module was chosen because it had a decent amount of accuracy at a low price while also being compatible with a BeagleBone Black. It was necessary to couple the GPS module with an Inertial Measurement Unit (IMU). This was necessary because the GPS serves as a best guess to the location and movement of the boat. The IMU is able to measure the velocity of the boat more precisely but is inaccurate over large distances. Combining both sensors with a Kalman filter allows the control system to have an accurate reading of its position. The IMU that was chosen was a Razor IMU.

Lidar

To achieve obstacle avoidance it was necessary to choose a sensor that will serve be able to detect an objects location. A Lidar was chosen because it was the most precise above water sensor available. The LIDAR unit the team chose to implement into the ASV was taken from a Neato Robotics XV-11. This Lidar was inexpensive compared to others which were out of the team's available budget. This Lidar was also researched by last year's team and has an acceptable sight range of about 15 feet. Any Lidar with a further sight range and a third spherical dimension of object detection would be priced several times higher than the total available budget.

Arduino Mega 2560

A low power microcontroller was needed to provide the controls for the servo connected to the rudder and the motor of the boat. The microcontroller the team chose for controlling the ASV's movement was the Arduino Mega 2560. The reason the team chose this microcontroller was due to its compatibility with the Electronic Speed Controller (ESC) and the servo motor for the rudder. It also had enough processing power and output pins to perform these tasks however not enough processing power to run MOOS which is why a second microcontroller was added to the ASV.

Figure 2-Arduino used for motor and rudder control.

BeagleBone Black

A second microcontroller was needed to actually run the autonomy processes of the boat. The microcontroller the team chose for running MOOS and interfacing with sensors was the BeagleBone Black. At the time the BeagleBone Black had the most processing power and could perform these calculations quickly for a reasonable price. The BeagleBone Black is able to make its calculations and decisions in MOOS and then pass this information to the ASV's Arduino Mega 2560, which would then translate the decision made into the actual signaling sent for the servo controller and ESC.

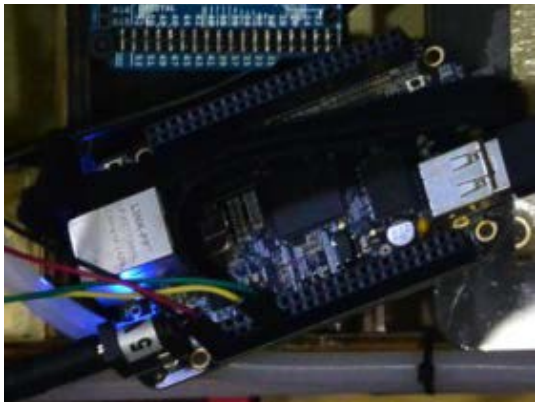


Figure 3-Beagelbone used for general autonomy.

Lidar Explanation

Lidar is a technology used for sensing objects near the sensor. It is conceptually the same as radar but uses the visible light spectrum (400 – 700 nm) rather than microwave and radio spectrums (3 mm – 100 m). This higher frequency spectrum allows for accurate detection of smaller objects over a shorter range. As a result Lidar is a better choice for an ASV obstacle detection because maneuvering around unexpected objects requires fine detail over a few boat lengths.

There were other alternatives that were considered for obstacle detection: sonar and video. Ultrasonic sonar does not offer fine enough resolution to build a map of nearby obstacles because one module only offers a narrow range of angles. Covering all angles would require a large number of sonar modules. Underwater sonar is not fit for avoidance of surface obstacles because it would not reliably detect shallow draft hulls. Using a combination of video and digital image processing would add a layer of unpredictability to the obstacle detection system. A video system's obstacle detection reliability is dependent on the efficiency of the coding, robustness of the algorithms, and speed of the hardware. Conversely the Lidar system used has reliably provided 360 degree (1 data point per degree) with little processing overheard to decode the serial stream from the Lidar module.

The LiDAR module used was taken from a Neato Robotics XV-11: a COTS robotic vacuum cleaner. This module is popular among electronic hobbyists for the ease of interfacing and low cost. The module outputs a stream of serial data that contains data points made up of distance, angle, and confidence. A python script was written so the BeagleBone could decode the serial stream and output obstacle locations to MOOS. Any data points past a certain distance (15 feet, chosen from empirical accuracy testing), in angle ranges that the boat is in relative to the module, or not accurate were thrown out. The last step was to get the data in to MOOS from python. A python-moos library was necessary to send the messages to MOOS but the implementation was not finished in time for the URC. ROS (MOOS-IvP alternative) was considered for importing data from sensors because it had drivers for a wide array of sensors already. One of the grad students working on the project was working on a ROS-MOOS bridge so that ROS could be used to collect sensor data. This would allow for future expansion of the ASV fleet without having to use the exact same hardware or spend a large amount of time creating custom drivers to import data.

The module's housing rotates at 300 (±30) rpm: allowing for a maximum refresh rate of five times a second. This refresh rate combined with a minimum safe distance between the boat and an obstacle determines maximum safe speed of the boat.

$$speed_{max} = f_{lidar} * \left(d_{range} - d_{safe}\right) \qquad 1.$$

$$d_{range} > d_{safe}$$

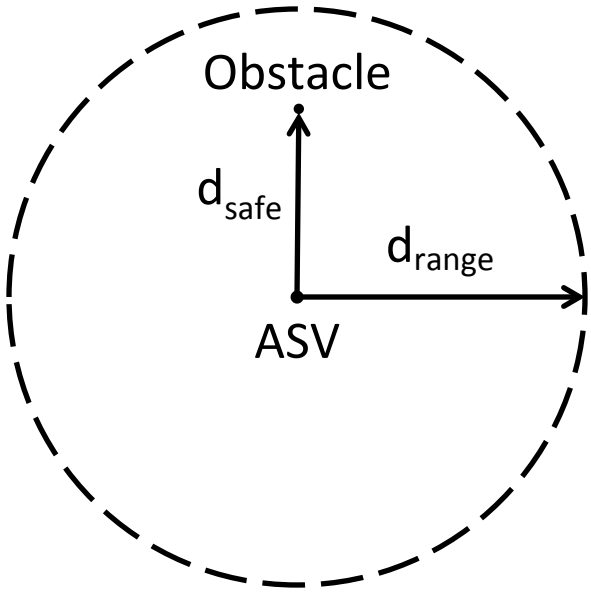


Figure 4-Effect of Lidar range and refresh rate on max speed.

In the case of this year's implementation it was prudent to reduce the refresh rate in software to account for potential slowdowns in the Lidar housing. The Lidar's range was 15 feet and a minimum safe distance of 10 feet was chosen.

$$speed_{max} = 4Hz * (15ft - 10ft) = 20\ {}^{ft}/_{s} \approx 11.85\ knots \qquad 2.$$

In practice a max speed of 4 knots was chosen for safe testing. The Lidar provides a sufficient amount of data. The refresh rate could have also been reduced to 2 Hz in order to lower the processing overhead without violating the Lidar's max speed criteria if the BeagleBone was not fast enough.

The python script has provisions to display a rendering of the lidar data in real time for debug purposes. This demonstrates the Lidar's accuracy and that the script works correctly in real time. The figure below is a snapshot of the debug display inside of a rectangular room. The green points are the obstacles found by the Lidar at a given angle. The yellow points denote the confidence for a given angle. The further away they are from the origin, the more confident the data point is.

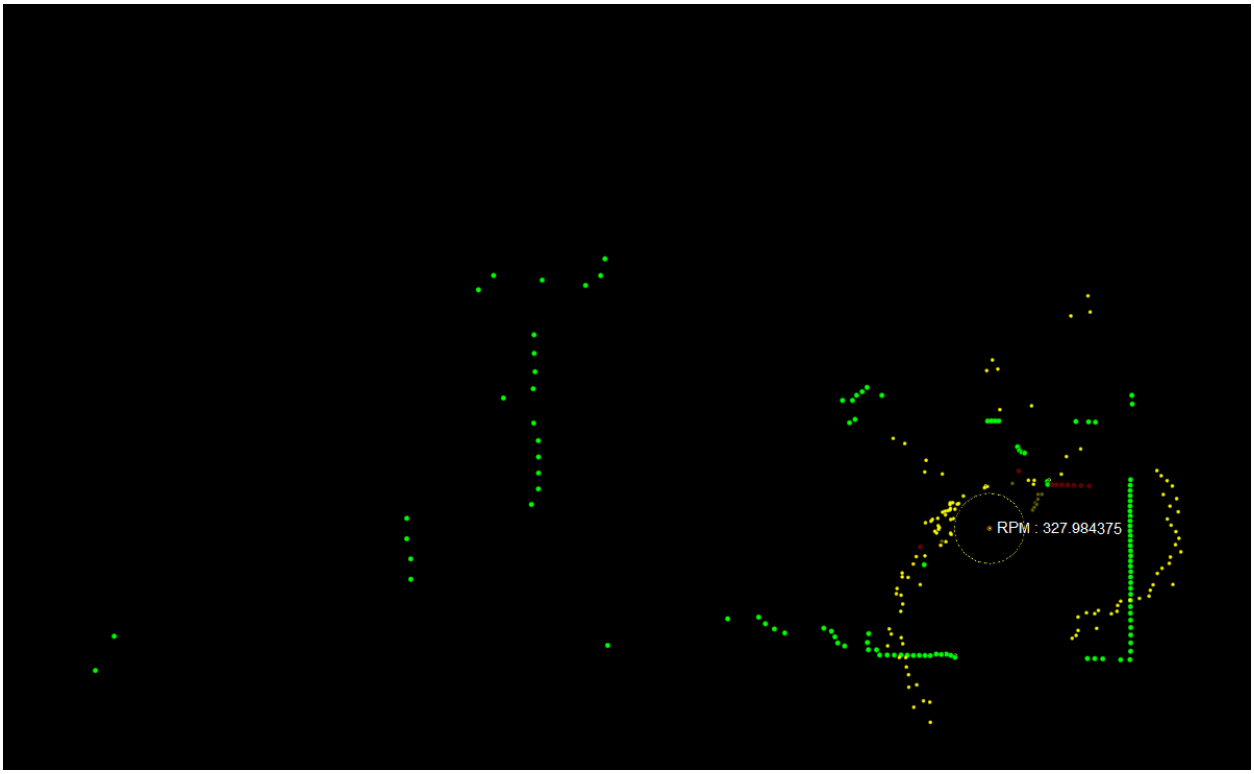


Figure 5-Lidar debug display outside of Plexiglas shielding

MOOS-IvP

MOOS-IvP is a two part open source software suite designed to control marine vehicles autonomously. It was originally created by MIT and Oxford respectively. The two main components are MOOS (Mission Oriented Operating Suite) and IvP (Internal Programming) Helm. MOOS maintains a database that can be accessed by multiple programs, known as MOOSApps, which contribute to autonomy in various ways, by accessing data in a collective database known as MOOSDB. The primary MOOSApp is IvP Helm which makes the autonomous decisions based on several autonomous behaviors, known as IvP Behaviors.

MOOS-IvP can be downloaded from moos-ivp.org

MOOSApps

MOOSApps (MOOS Applications) are programs which connect to a single MOOSDB. MOOSApps do not interact directly with one another, and therefore act independently of one another, each sending or receiving data from the MOOSDB as part of its behavior. This also means any combination of existing MOOSApps can be used on any given mission. Several MOOSApps come with MOOS-IvP and custom MOOSApps can be written to perform additional functions, such as communicating directly with sensor input. Figure 6 is a diagram known as a MOOS Community, which includes the MOOSDB, IvP Helm, and all MOOS Apps associated with the MOOSDB for a given mission.

When a MOOSApp launches, it will subscribe to a set of variables. When the variable is changed, no matter the source of the change, the MOOSApp's copy of the variable will be updated to the new value. A MOOSApp can send changes it has made to a variable to MOOSDB and therefore update all other MOOSApps that are subscribed to that variable.

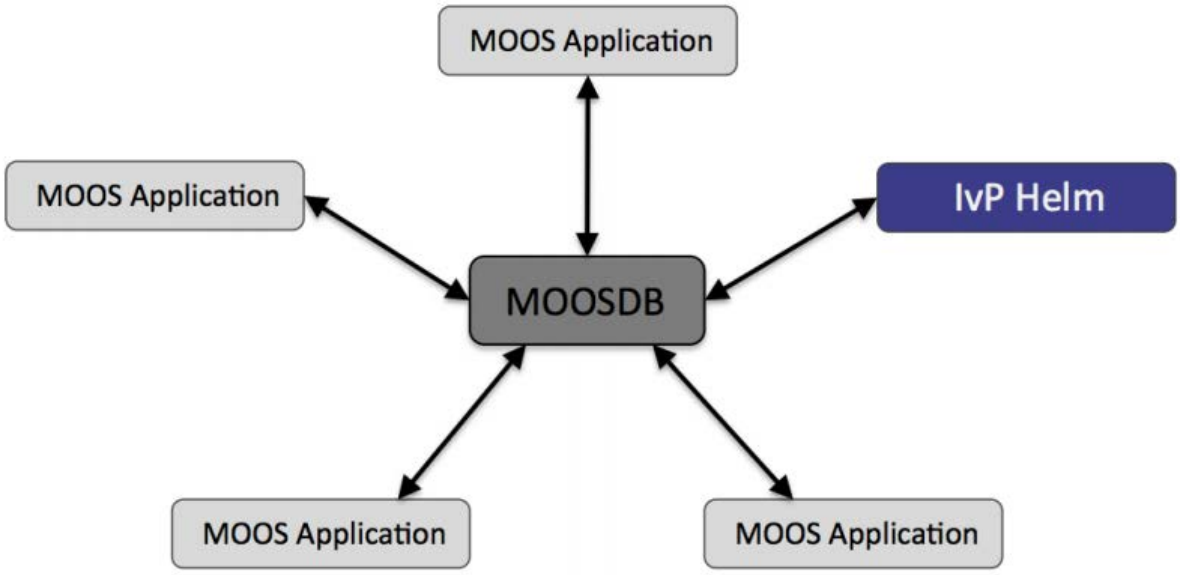


Figure 6-Example of a MOOS Community containing the IvP Helm. http://oceanai.mit.edu/moos-ivp-pdf/moosivp-helm.pdf

A MOOSApp can perform any function. MOOS-IvP comes packaged with many MOOSApps, some of which were used for creating missions. pLogger is a MOOSApp which was useful for recording what the system thought it was doing over the course of a mission by logging the values of all fields in the MOOSDB over the course of a mission.

Another MOOSApp package with MOOS-IvP is pMarineViewer. This App is a graphical user interface with displays the variables being used by the current mission. It also offers a visualization of the mission including location of obstacles, vehicles, and vehicle paths and waypoints. The pMarineViewer MOOSApp can be seen in Figure 7.

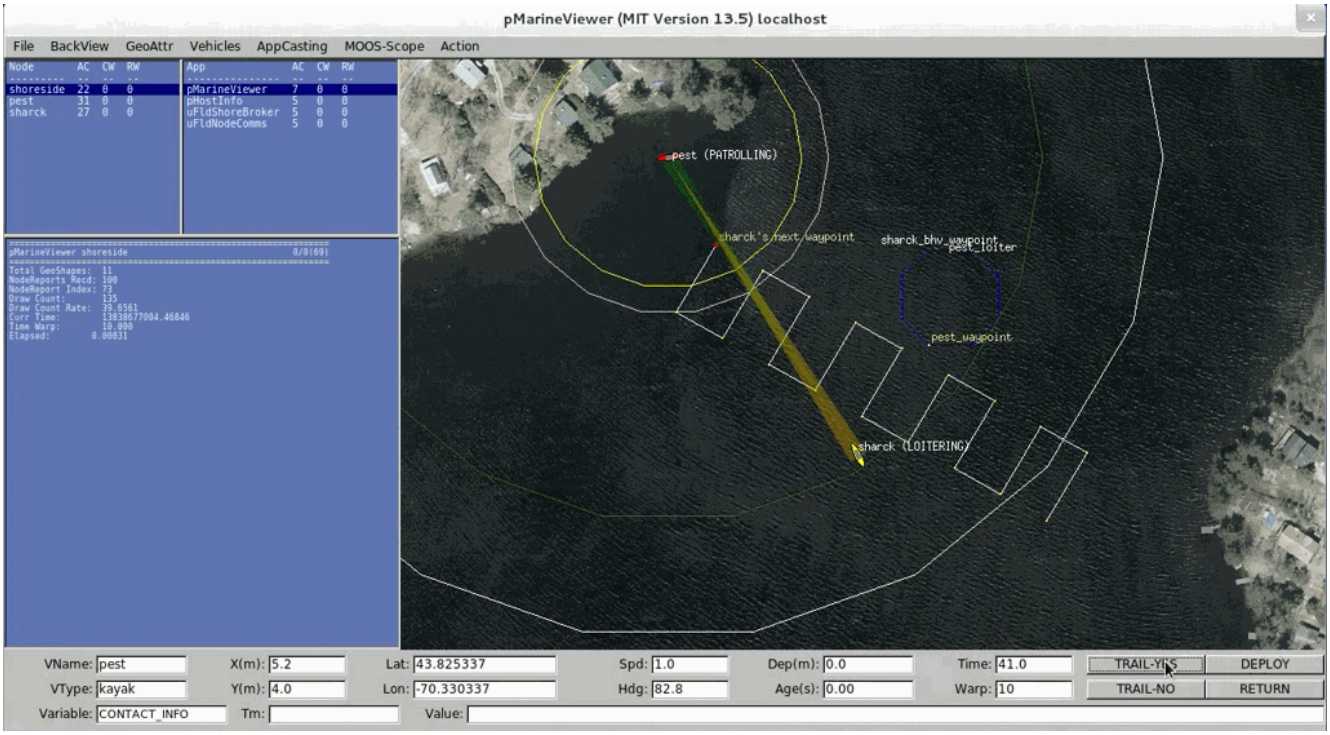


Figure 7-The built-in MOOS-IvP graphical mission viewer (pMarineViewer) running a simulation of a track and trail mission.

MOOSApp Architecture

MOOSApps all have the structure shown in Figure 8. The OnStartUp method is called when the process starts and registers variables from the MOOSDB that the MOOSApp will use. The MOOSApp then enters an infinite loop calling OnNewMail, followed by Iterate. OnNewMail checks the registered variables for changes and updates the MOOSApp as appropriate. Iterate is the main algorithm for the MOOSApp, executing the desired functionality for the MOOSApp.

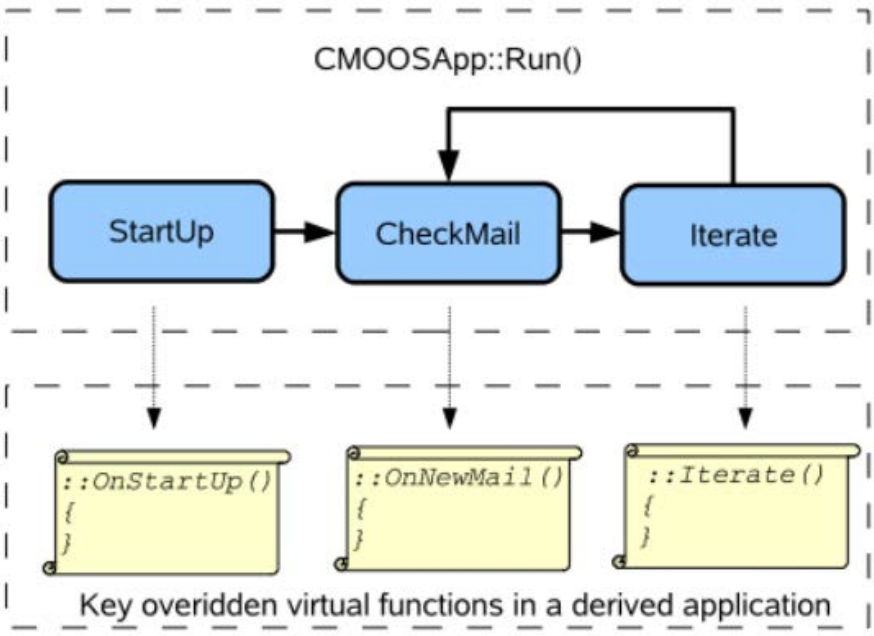


Figure 8-The main MOOSApp program flow and the corresponding methods which perform the program execution at each state. http://oceanai.mit.edu/moos-ivp-pdf/moosivp-helm.pdf

IvP Helm

IvP Helm is a single MOOSApp which is responsible for making the autonomous decisions. It uses a set of behaviors which are competing for control of the "helm" throughout the mission. IvP Helm decides which behavior should have control at any given time. Behavior examples can vary from simple point-to-point navigation to continuous actions such as following a moving waypoint or avoiding obstacles. Output from IvP Helm is stored in the MOOSDB just like any other MOOSApp but it is this data that is the key to autonomy.

IvP Helm manages all the behaviors which are used in a mission. It determines which behaviors will be active, and when. It executes all active behaviors, each behavior examines the current state of the mission, and produces an IvP Function. IvP Fucntions are mathematical functions used by the IvP Helm to make the final decision for the current iterate loop.

The IvP Helm structure and functionality can be seen in Figure 9.

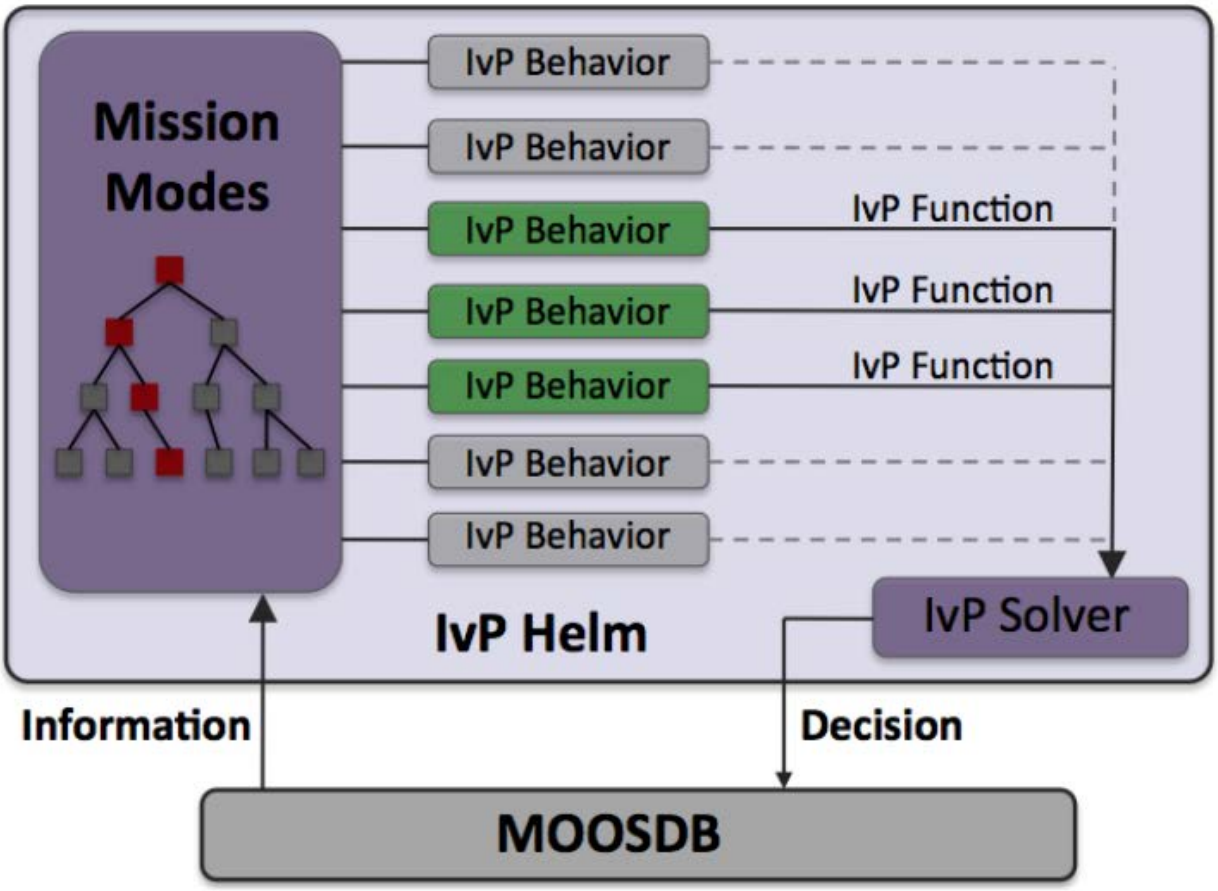


Figure 9-The IvP Helm internal structure. http://oceanai.mit.edu/moos-ivp-pdf/moosivp-helm.pdf

IvP Behaviors Example

If IvP Helm is running two active behaviors, for example waypoint navigation and obstacle avoidance, it has to determine how to execute both behaviors without compromising either one. The waypoint behavior always tries to go the shortest possible distance to reach a waypoint. The obstacle avoidance behavior with case the vehicle's planned path to be altered if the vehicle would collide with an obstacle. The IvP Helm will determine a compromise between the two behaviors such that the vehicle will go the shortest possible path, while still avoiding the obstacle.

MOOS-IvP Missions

A MOOS-IvP mission is simply a set of configuration files. There are two types of files with MOOS-IvP uses, one for MOOSApps and one for behaviors. These files tell MOOS-IvP which MOOSApps and behaviors should be used for the mission and how they should function (by setting variable values). The mission is launched by using pAntler, an application that comes with MOOS-IvP. pAntler simply takes all configuration files for the mission as arguments and launches them with a new MOOSDB. When IvP Helm is launched by pAntler it is fed the configuration file for the behaviors so it can load the needed details.

Programming MOOS-IvP

MOOS-IvP is an open source tool, so it can easily be modified and/or expanded to meet the user's needs. MOOS is written in C++, so new MOOSApps and behaviors can just extend their respective classes and overwrite methods as needed to produce the desired functionality. For this project several MOOSApps were written to communicate with systems outside of the MOOS-IvP. Behaviors can also be written for MOOS-IvP, but none were needed with for the current scope of the project.

The MOOSApps developed for this project are known collectively as the Payload Autonomy Interface. These MOOSApps were used for communication with the sensory information and to transfer autonomy information to the controls of the vehicle through the Arduino. A sketch was also written to convert autonomy output from MOOS-IvP to the ASV controls (servo and rudder) from the Arduino.

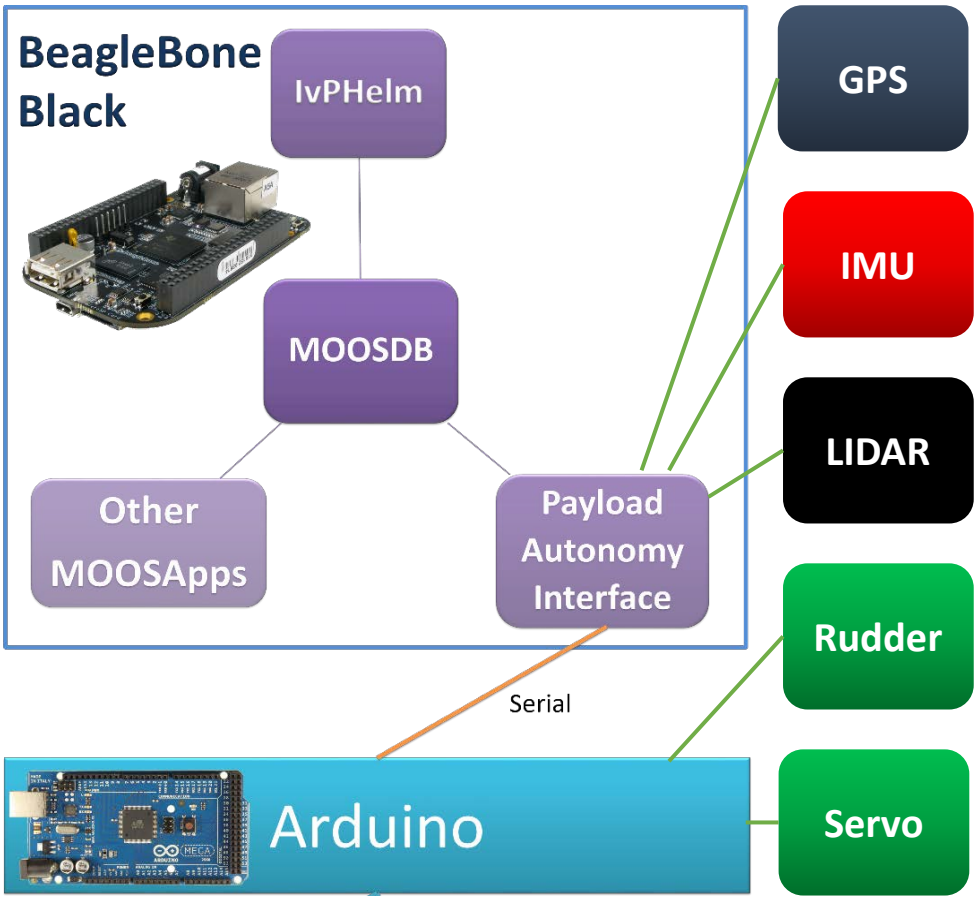


Figure 10-Overview of the main software components and their relation to the hardware they are contained in and interface with.

Payload Autonomy Interface

The Payload Autonomy Interface is a collection of MOOSApps which register MOOSDB variables that are produced by the IvP Helm. It sent data to the Arduino over a USB serial connection (see Figure 10). The MOOSApps also read in sensor data from the GPS, Lidar, and

IMU to updated the MOOSDB with the actual ASV position, heading, and speed, as well as detected obstacle data. This data flow can be seen in Figure 12.

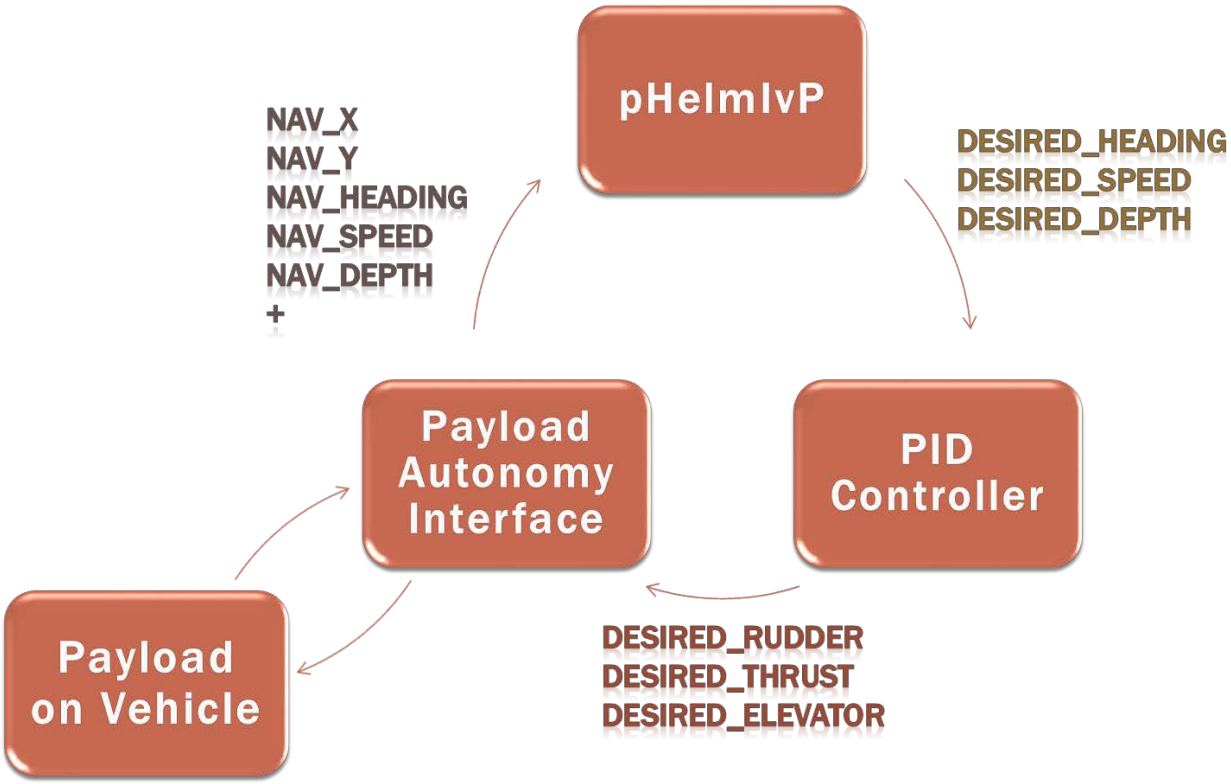


Figure 11-The data flow to and from the payload autonomy interface. The actual data flow within MOOS-IvP goes through the MOOSDB.

Messages sent to the Arduino are in a *key=value* format, where *key* is the name of the variable being sent and the *value* is the actual data value. This format works well because it is human readable (for debugging), but isn't too slow because the set of key needed is very small. Key-value pairs are separated by a delimiter to distinguish between different messages as they are buffered through the serial connection. The default delimiter is a comma, but the system is set up so the delimiter can easily be changed.

The values were then processed by the Arduino sketch. The string was parsed and value extracted for the angle and speed desired. Both values were scaled to match appropriate values to be sent to the servo and rudder (such as the desired rudder value is converted from [-90,90] to [0,180]). The values were then written out to the servo and rudder hardware controls to execute and move the ASV.

With the payload autonomy interface setup, data was seamlessly passed to and from the frontend on the vehicle thus creating a contained loop of information necessary for the autonomous operations of the vehicle. There is architecture in place to support the passing of NAV_X, NAV_Y, NAV_SPEED, NAV_HEADING, DESIRED_SPEED, DESIRED_HEADING, DESIRED_THRUST, and DESIRED_RUDDER. NAV_* variable are the actual ASV location data obtained via sensor input, while DESIRED_* variables are the output of IvP Helm to control the ASV autonomously.

Results and Discussion

The first step that the group took was to decide how to approach the task of constructing an autonomous boat. The first part of this step was to define exactly what the team would consider autonomous behavior which it broke into the three previously mentioned stages. The next step was to discuss what parameters would be used by the boat to act out these three stages. It was decided that for point to point navigation the destination should be approached by the ASV and then once a radial distance between the ASV and the point has been achieved, the boat would then enter a circular holding pattern about the point as seen in Figure 12 while the boat decelerates. For obstacle avoidance it was decided that the boat should simply maintain a minimum distance between itself and each detected obstacle: moving around each object at this distance to avoid collision. This behavior can be seen graphically in Figure 13. For track and trail the boat would behave similarly to obstacle avoidance and maintain a set distance behind the vessel being followed by the ASV.

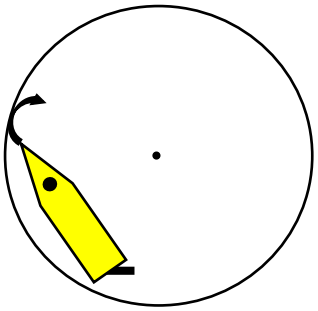


Figure 12-This demonstrates the ASV entering an acceptable radial distance from its destination and entering an orbital pattern about its destination while decelerating.

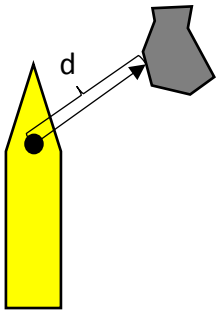


Figure 13-This figure demonstrates the intended behavior of MOOS-IvP keeping a minimum distance between the ASV and a detected object.

The second step that the group took in this project was to separate into three distinct teams to work on the project in parallel components so that the most work could be accomplished at once. The first team the group designated was the coding team. This team was responsible for all autonomy behaviors and signal processing on the main microprocessor (BeagleBone). This team had the first semester set aside to write and troubleshoot the code that was intended to eventually dictate the behavior of the boat in the processes of navigation, obstacle avoidance, and track and trail missions. The second team that the group broke into was the team that was responsible for the physical construction and configuration of the autonomy platform for which the first team was writing code. The third team was tasked with choosing and configuring the sensors and Arduino that would allow the BeagleBone to interface with the boat and its surroundings: producing a platform that could read in situational data and respond physically to accomplish a designated task such as navigating from one location to the next.

Once the group was properly broken into teams and the project had been divided into manageable pieces the next step was to purchase the components necessary to fit a boat with an autonomous module. Following the purchase of these components the boat was tested for functionality to ensure that any future issues could be troubleshot on the installed components with the knowledge that the boat and its preinstalled motor and servo functioned properly out of the box.

While the components of the boat were coming together, the programming team had its own task to complete before getting into the detailed coding for the project. Its assignment was to research ROS, another environment that different organizations are using to program behaviors into different platforms, and MOOS, the platform that the previous year's team used. It was decided that the group should stay with MOOS for this year's iteration of the project due to the existence of legacy code from the previous year and the fact that the team led by the point of contact to the navy is also using MOOS for its surface vehicle research.

The next step was to install the physical control system into the boat. The first part of this step was to install the Arduino Mega 2560. The Arduino was installed so that it acted as a bridge between the RC receiver and the motor and rudder. To accomplish this the Arduino was connected to the RC receiver so that it receives an input from the receiver. The Arduino was then connected to the motor and the servo that is connected to the rudder so that it outputs to these components of the boat. In setting up the Arduino in this way the boat was able to be controlled in a manual, or RC, mode and an autonomy mode. When switched into the autonomy mode the boat would perform whatever mission it was tasked with autonomously. The manual mode was put in as a failsafe in case the boat did not act as it should. The manual mode can override the autonomy mode so that the boat can be returned safely.

This step was an important one for establishing the control system of the boat. Achieving RC control through the Arduino allowed for verification that the Arduino was doing what it was told to do by the RC remote controller. From here the inputs into the Arduino could be read so that for certain speeds and rudder angles; an Arduino input value could be assigned to each. This was an important step for the autonomy as the autonomy program could communicate to the Arduino what it wants accomplished.

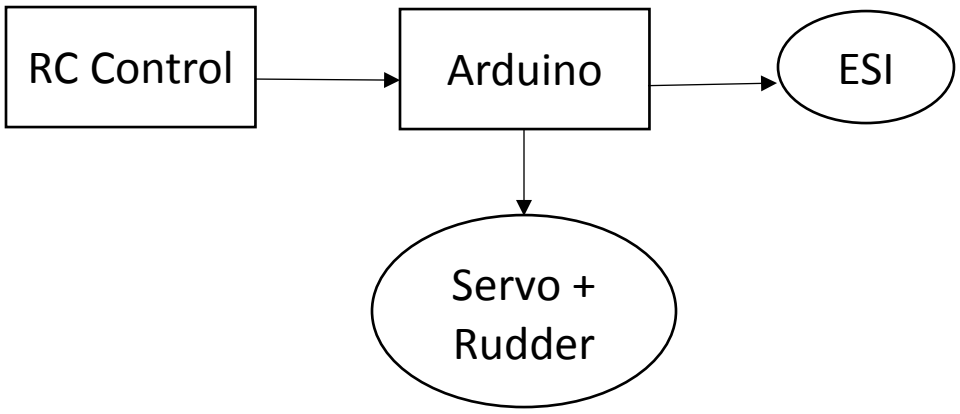


Figure 14-This figure display a simplified diagram of how RC control is passed through the Arduino.

Another important step in completing the control system was to understand how the boat physically responds in water to certain inputs. To do this an analytical and experimental model of the boat were created. The goal of these models were to take a certain thrust input from the boat and model the velocity that the boat will produce. This was important because it gave the control system of the boat a fairly good initial guess of where and how the boat will start moving.

To compare against the velocity data produced by OPIE the team also constructed a Simulink model of the boat based off of equations listed in Thor I. Fossen's *Guidance and Control of Ocean Vehicles.* The author details three kinematic equations of motion to describe the movement of the boat across the surface of the water in terms of a body coordinate system:

Surge: $$m(\dot{u} - vr - x_G r^2) = X$$

Sway: $$m(\dot{v} + ur + x_G \dot{r}) = Y$$ 3.

Yaw: $$I_z \dot{r} + m x_G (\dot{v} + ur) = N$$

Where m is the mass of the boat, x_G is the distance between the origin and the x-axis center of gravity of the ship, I_z is the moment of inertia about the vertical axis at the origin of the body coordinate system for the boat, u is the forward velocity of the boat, v is the horizontal velocity of the boat and r is the rate of rotation about the vertical axis of the boat.

These equations are simplified from an original six that are meant for submersible vehicle, but roll, pitch and vertical movement are all forms of motion that would be undesirable for this vehicle so they were not included in the model. The model was simplified by anchoring the coordinate axis at the center of gravity for the boat: driving x_G to equal zero.

The next step of the analysis was to convert the body anchored coordinate system results into a more usable Earth centered coordinate system. The process necessary to complete this step was also included in Fossen's publication and involves performing systematic rotations of the body coordinate system for the boat until it aligns with the fixed coordinate system of the Earth. Conveniently, the fact that the boat is a surface vehicle prohibits it from performing rotation about its body x or y axes, as detailed in Figure 15.

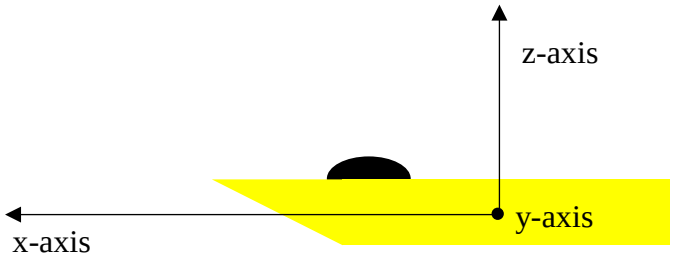


Figure 15-This figure displays the body coordinate system used for the boat.

Figure 16 displays the yaw rotation that the boat is capable of on the surface that it is traversing:

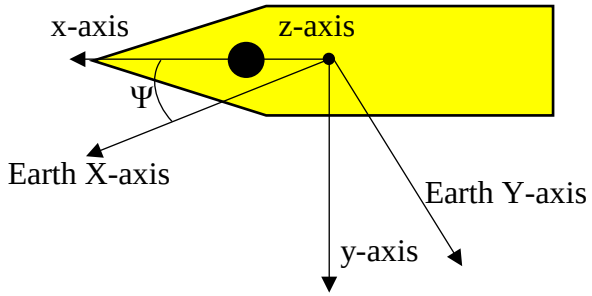


Figure 16-This figure shows the rotation between the body coordinate system of the boat and the fixed coordinate system of the earth.

This rotation was accomplished through the following transformation matrix:

$$C_z = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad 4.$$

Where C_z denotes that this rotation occurs about the z-axis.

Note that the angle denoted as Ψ in Equation 4 is the angular displacement between the body coordinate system of the boat and the fixed coordinate system of the Earth and r, found in equation 3 is also its derivative in the body coordinate system.

The third main step of the analytical model that needed to be accounted for was the skin friction drag on the surface of the boat. This contribution to thrust was modeled using a flat plate analysis assuming that the flow under the boat transitioned from laminar flow to turbulent flow in the boundary layer. The area of the plate that was modeled was dimensioned with the same width and length of the boat. The coefficient of drag was calculated from equations 5 and 6.

$$Re_L = \frac{uL}{\nu} \quad 5.$$

Where u= surge velocity of flow, L=length of boat, υ=kinematic viscosity, and Re_L=Reynold's number at the back of the boat.

$$C_D = \frac{.0455}{(\log(Re_L))^{2.58}} - \frac{1610}{Re_L} \quad 6.$$

The velocity u that was used to calculate the Reynold's number was 1.2 m/s, which was found from the experimental results as a reasonable value for all three tests. These calculations resulted in a coefficient of drag of .003. These equations were taken from *Fox and Mcdonald's Introduction to Fluid Mechanics 8th edition* by Philip J. Pritchard. The resultant drag force, was then calculated for a component contributing to surge, sway and yaw.

The experimental portion of this model involved obtaining a velocity versus time profile for the boat when it is initially at rest. The goal of doing this was to compare the experimental

profile to the one that is generated from the analytical model. This allows the analytical model to be verified for its accuracy in predicting the dynamics of the boat.

The experimental procedure of obtaining this profile had two steps. The first step was to obtain thrust values for the boat corresponding to certain Arduino input values. The second step was to actually obtain a velocity profile for the boat at a given Arduino input value. The first step allows for comparison with of the experimental model to the analytical model. Obtaining Arduino input values corresponding to thrust values allowed that a given thrust value could be assigned to an experimental velocity profile.

To perform these tests, the input from the Arduino to the motor system for the boat had to be tested and calibrated. It was found from this investigation that the input signals ranging from 84 to 180 from the Arduino would produce motor movement, with the spectrum from 0-84 corresponding to reverse thrust from the motor which was ignored by the electronic speed controller of the boat, providing the thrust necessary to drive the boat between 0 and 40 mph. To keep well below the 40 mph threshold, the three steps of 95, 100 and 105 were chosen for the experimental portion of this project.

The first experiment performed was to obtain velocity data versus time for the boat in response to the three step input signals to the motor. To obtain velocity data a target was mounted to the front of the boat, providing a calibration distance as well as two tracking dots for the program, OPIE, to follow and eventually calculate position versus time data from. The method used by OPIE to track objects operates off of the pixel count for each frame and the calibration distance provided on the target. First a width for the calibration dot is entered into the OPIE interface, then OPIE uses this entry as a legend for all future distance calculations. The program counts the number of pixels between the position of the primary tracking dot from one frame to the next and uses the time elapsed between frames, which at 60 fps is 1/60th of a second, to find the position of the boat as a function of time. For these tests the boat was run perpendicular to the position of the camera to ensure that all translation in front of the camera occurred along the horizontal axis of the perspective of the camera. The data from these tests was then loaded into MATLAB and numerically differentiated to find the velocity versus time of the boat. An example of this calculation method can be seen in Figure 17.

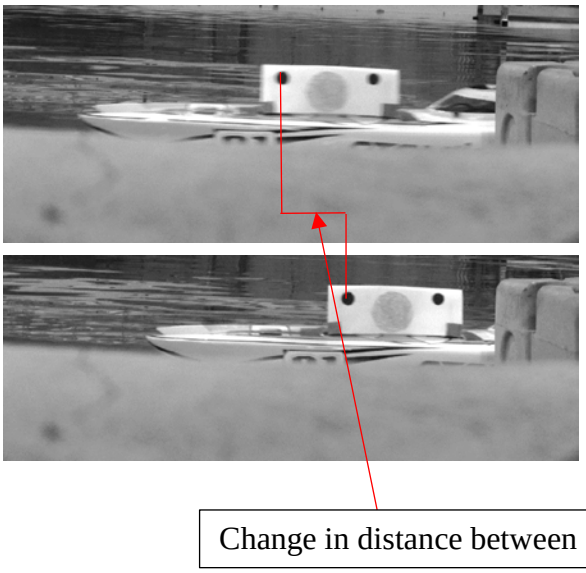


Figure 17-This figure demonstrates an exaggerrated example of the method used by OPIE, a change in position for the front dot is reqistered by the Matlab script and the time step of the frame change is also registered.

The second experiment performed was to determine the thrust output by the motor of the boat at each of the selected step input signals used in these experiments. Since the step signals were chosen without exact knowledge of their physical thrust output, it was important to find the thrusts that corresponded to each step input. To accomplish this test it was required to set up a load cell with a 100x differential amplifier powered by a +15V,-15V power supply at the side of the Chase OE tank and push the boat with the same step inputs while reading the output of the load cell powered with 5V on a handheld DMM. The load force from the thrust was calculated using the manufacturer's published sensitivity for the load cell. The premise behind this test was that with the boat tied down, the only force acting on the rope between the boat and the load cell was the thrust output from the motor, due to it being static. The damping force and the reaction force from the water being pushed out of the path of the boat did not factor into this test because the boat's change in position and rate of change of position were constrained to be zero in this experiment. Figure 18 shows a sketch of the experimental setup used for this test.

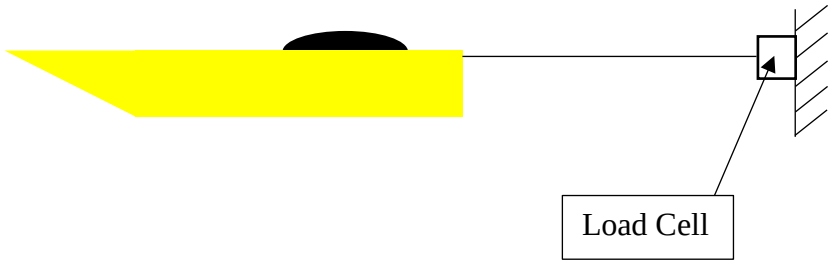


Figure 18-This figure shows a general schematic for the second experiment performed with the boat.

Once the boat was physically defined the BeagleBone and GPS could be installed. Once this step was completed, the hardware for achieving waypoint navigation was fully equipped. The GPS was mounted in the Lidar housing for ease of access and functionality. The BeagleBone was mounted inside the boat just behind the area where the Lidar housing is on the top of the boat. This was determined as the optimum spot to mount the BeagleBone because it was located close to where the sensors are located and there was an open space within the boat. To mount the BeagleBone two Velcro straps were attached in that space so that the BeagleBone would be securely fastened when the boat was in use.

Once the hardware for waypoint navigation was successfully installed the programming of this first stage if autonomy could be implemented. The first step in implementing the code was to test the GPS in testing areas to make sure that the accuracy of the GPS was sufficient enough to run the autonomy. If the accuracy of the GPS was not good enough the boat would not be able to travel to a waypoint successfully as it would not get an accurate enough signal of its location. Two testing locations were used: The Chase Ocean Engineering Laboratory engineering tank, and the Swasey Pool. It was necessary to test inside at the time due to seasonal limitations. Any outdoor body of freshwater was frozen at the time of the planned testing. It was determined that the Chase engineering tank did not get an accurate enough GPS signal however the Swasey Pool did have a sufficient signal for preliminary testing. Based on these results two dates were planned and reserved for the Swasey Pool.

It was important that the boat was tested before it was put in the pool to make sure that there were as little problems as possible once the testing dates came. To do this the boat was dry tested. To do this the boat was put on its stand and had one of its missions programmed onto it. The program was then run to see if the boat was reacting how it was expected to. Unfortunately not all of the software bugs were fixed before the first testing date arrived.

The first day of testing in the pool consisted of more bug fixing: mostly related to the GPS. The boat was first put in the pool and run through RC control. This was done to make sure that all of the components within the boat were working properly. Once this was done, the next step was locating the GPS signal. The GPS needs to initially connect to a satellite before it is fully functional. A GPS signal was soon found and the next step was actually implementing autonomy. Several unsuccessful tests were performed and autonomy was not achieved. This testing did however give us data that could be used to debug the autonomy code for future testing.

The second day of testing had similar results. The GPS was connected and waypoint navigation was tested. During this testing it was discovered the the BeagleBone was not communicating properly with the Arduino. Much of the testing time was spent trying to debug this issue and so the testing ended as another unsuccessful test in terms of achieving autonomy.

At this point the team had to prepare for the Naval Engineering Education Consortium. For this event the team had to present its findings and progress on the ASV. To prepare for this a poster was created showing the team's findings along with the graduate student advisors' findings. The team spent three days at the conference.

Conclusion

The overarching purpose of the ASV project was to provide a proof of concept for future autonomous systems by outfitting a vessel with the programs and components needed to perform waypoint navigation, obstacle avoidance, and track-and-trail. The team has contributed to this goal by achieving simulated waypoint navigation on a modified commercially purchased boat using communications between Adruino and a BeagleBone running MOOS-IvP. The boat was able to follow a programmed mission, which involved moving at a given speed for a specific amount of time before cutting its motor. GPS data extraction and transfer to the BeagleBone is currently in progress, and will enable the MOOS processes and behaviors necessary to direct the boat based on its location. The Lidar can now successfully collect data and will soon be integrated with the BeagleBone to allow for information storage in MOOSDB. With this process, the prioritized reaction to a recognized obstacle is for the boat to maintain a constant distance between itself and the object for the duration of the avoidance maneuver. The future of ASV consists of building on the current autonomy and developing the logic and processes for track-and-trail.

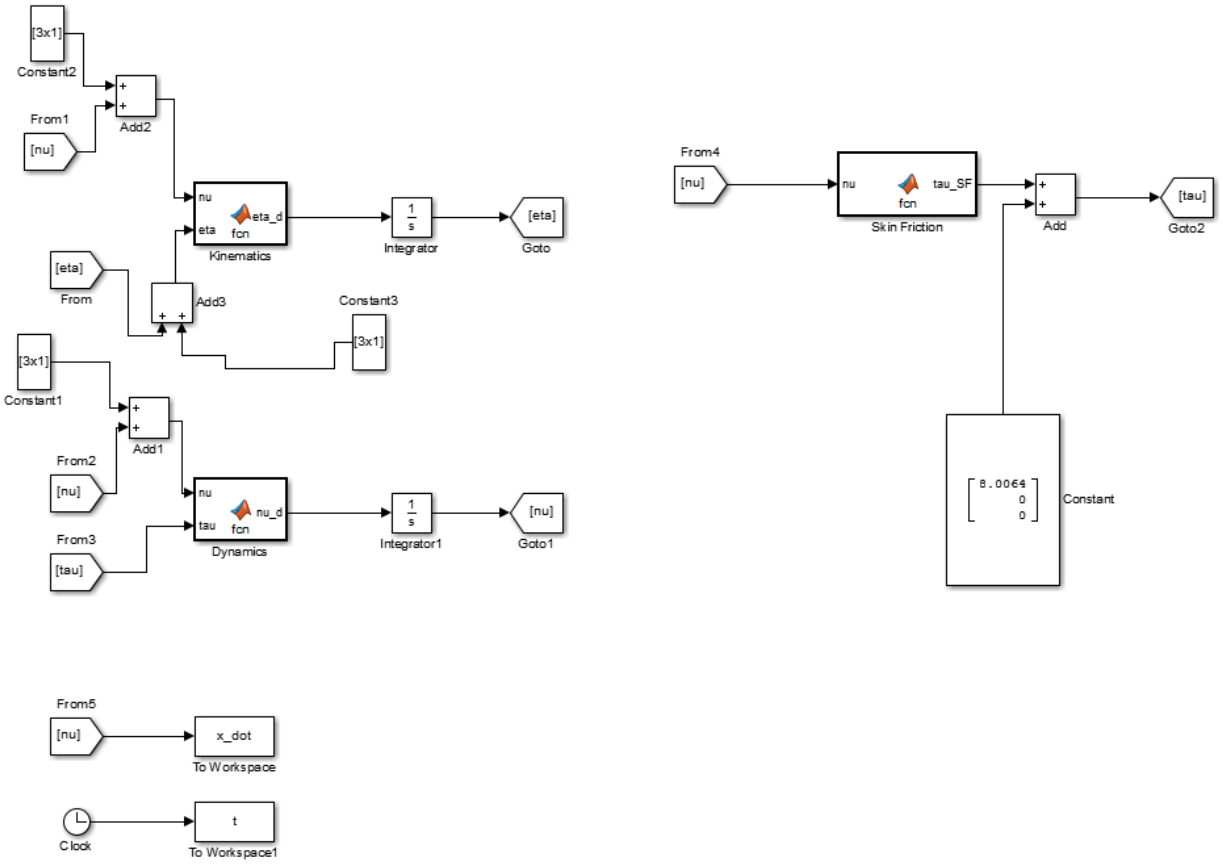
Works Cited

Fossen, Thor I. *Guidance and Control of Ocean Vehicles*. Chichester: Wiley, 1994. Print.

Pritchard, Philip. *Fox and McDonald's Introduction to Fluid Mechanics*. 8th ed. Hoboken, NJ: John Wiley & Sons, 2011. 440-448. Print.

Appendices

Simulink Diagram



Simulink Code

Kinematics

```
function eta_d = fcn(nu,eta)
%#codegen
%Kinematics

p=eta(3);

B=[cosd(p) sind(p) 0; -sind(p) cosd(p) 0; 0 0 1];

eta_d=B*nu;
```

Dynamics

```
function nu_d   = fcn(nu,tau)
%#codegen
%Dynamic Block
```

```
m=6.8737; %kg
xg=0;
Iz=.13;

X=tau(1);
Y=tau(2);
N=tau(3);

u=nu(1);
v=nu(2);
r=nu(3);

r=nu(3);

A=[m 0 0; 0 m xg; 0 m*xg Iz];

Ainv=inv(A);

b=[X+m*(v*r+xg*(r^2));Y+m*(-u*r);N-m*xg*u*r];

nu_d=Ainv*b;
```

Non-Linear Terms

```
function tau_SF = fcn(nu)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Calculates the forces/thrusts associated with Skin Friction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%

D_SF_nu = 0.003;        % Coefficient for Skin Friction term

Length_ft = 58/12;
Width_ft = 13/12;
Surface_Area_ft = Length_ft*Width_ft;    % Cross-sectional areas (ft^2)

% Define states

%psi = eta(3);           % absolute heading (deg)

vel_u = nu(1);     % velocity in u-direction (m/s)
vel_v = nu(2);     % velocity in v-direction (m/s)

omega_r = nu(3);     % angular velocity (deg/s)
psi_vel = atan2(vel_v,vel_u);    % direction of velocity vector wrt body
coordinates
```

```
%vel_u_knots = vel_u*0.447*1.15;     % velocity in u-direction (knots)
%vel_v_knots = vel_v*0.447*1.15;     % velocity in v-direction (knots)
speed_knots = sqrt(vel_u^2 + vel_v^2)*(0.447*1.15)^2;

% Calculate forces in body-fixed coordinates

%R_SF_lbf = D_SF_nu*Surface_Area_ft*(speed_knots^1.83);  % total resistive
force (lbf)
R_SF_lbf = D_SF_nu*Surface_Area_ft*(speed_knots^1.83);  % total resistive
force (lbf)
R_SF_N = R_SF_lbf*4.448;                                 % total resistive
force (N)

%Rr_SF_lbf_ft =
D_SF_nu*Surface_Area_ft*(abs(omega_r)*pi/180*Length_ft/4)^1.83*Length_ft/4;
Rr_SF_lbf_ft =
D_SF_nu*Surface_Area_ft*(abs(omega_r)*pi/180*Length_ft/4)^1.83*Length_ft/4;
                                        % resistive yaw torque (lbf-ft)

tau_SF_u = R_SF_N*cos(psi_vel);                          % resistance in u-
direction (N)
tau_SF_v = R_SF_N*sin(psi_vel);                          % resistance in v-
direction (N)
tau_SF_r = (Rr_SF_lbf_ft*4.482*0.3048)*sign(omega_r);   % resistance in yaw
direction (N-m)

tau_SF = [tau_SF_u tau_SF_v tau_SF_r]';
```

Comparison Code

```
%Final Code

close all
clear all
clc

Vel_105=[0.220721925133689;0.0735739750445629;0.147147950089126;0.33108288770
0534;0.294295900178252;0.220721925133689;0.110360962566845;0.183934937611408;
0.257508912655970;0.478230837789659;0.515017825311940;0.294295900178252;0.367
869875222815;0.404656862745096;0.515017825311941;0.515017825311941;0.40465686
2745096;0.625378787878786;0.735739750445630;0.662165775401066;0.8461007130124
74;0.882887700534756;0.735739750445630;0.919674688057037;0.956461675579318;0.
846100713012474;0.956461675579319;1.10360962566844;1.17718360071301;1.1036096
2566844;1.10360962566845;1.17718360071301;1.17718360071301;1.17718360071301;1
.17718360071301;1.17718360071301;1.17718360071301;1.17718360071301;1.17718360
071301];
t_105=[0,0.017000000000000,0.034000000000000,0.051000000000000,0.068000000
0000000,0.085000000000000,0.102000000000000,0.119000000000000,0.136000000000
000,0.153000000000000,0.170000000000000,0.187000000000000,0.204000000000000,0
.221000000000000,0.238000000000000,0.255000000000000,0.272000000000000,0.2890
00000000000,0.306000000000000,0.323000000000000,0.340000000000000,0.357000000
000000,0.374000000000000,0.391000000000000,0.408000000000000,0.42500000000000
0,0.442000000000000,0.459000000000000,0.476000000000000,0.493000000000000,0.5
10000000000000,0.527000000000000,0.544000000000000,0.561000000000000,0.578000
```

```
000000000,0.595000000000000,0.612000000000000,0.629000000000000,0.646000000000000]';
Vel_100=[0.357050173010379;0.142820069204152;0.142820069204152;0.035705017301038
0;0.0357050173010380;0.285640138408303;0.178525086505190;0.107115051903114;0.107115051903114;0.249935121107265;0.392755190311417;0.357050173010379;0.357050173010379;0.499870242214531;0.535575259515569;0.357050173010379;0.285640138408303;0.714100346020758;0.678395328719721;0.535575259515568;0.749805363321796;0.678395328719720;0.606985294117645;0.678395328719721;0.821215397923872;0.964035467128023;0.749805363321797;0.642690311418683;0.928330449826986;0.928330449826986;0.928330449826986;1.07115051903114;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.10685553633217;1.10685553633217;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321;1.14256055363321];
t_100=[0,0.0170000000000000,0.0340000000000000,0.0510000000000000,0.0680000000000000,0.0850000000000000,0.102000000000000,0.119000000000000,0.136000000000000,0.153000000000000,0.170000000000000,0.187000000000000,0.204000000000000,0.221000000000000,0.238000000000000,0.255000000000000,0.272000000000000,0.289000000000000,0.306000000000000,0.323000000000000,0.340000000000000,0.357000000000000,0.374000000000000,0.391000000000000,0.408000000000000,0.425000000000000,0.442000000000000,0.459000000000000,0.476000000000000,0.493000000000000,0.510000000000000,0.527000000000000,0.544000000000000,0.561000000000000,0.578000000000000,0.595000000000000,0.612000000000000,0.629000000000000,0.646000000000000,0.663000000000000,0.680000000000000,0.697000000000000,0.714000000000000];
Vel_95=[0.550333333333331;0.137583333333333;-0.103187499999999;0.0343958333333332;0.275166666666665;0.171979166666666;-0.0343958333333331;0.103187499999999;0.171979166666666;0.0687916666666664;0.103187499999999;0.447145833333331;0.275166666666665;0.0687916666666664;-0.0343958333333331;0.309562499999998;0.515937499999997;0.0343958333333329;0.171979166666666;0.171979166666666;0.481541666666664;0.687916666666663;0.0343958333333333;0.206374999999999;0.791104166666662;0.653520833333330;0.550333333333331;0.447145833333331;0.309562499999998;0.584729166666664;0.343958333333331;0.412749999999998;0.825499999999996;0.584729166666663;0.447145833333331;0.653520833333330;0.687916666666663;0.584729166666663;0.825499999999996;0.584729166666663;0.412749999999997;0.928687499999996;0.825499999999996;0.791104166666663;0.825499999999996;0.791104166666661;0.791104166666663;0.756708333333330;0.894291666666662;0.791104166666663;0.756708333333330;0.928687499999994;1.03187499999999;0.791104166666663;0.825499999999996;1.10066666666666;1.03187499999999;0.825499999999996;0.825499999999994;1.03187500000000;1.10066666666666;1.03187499999999;1.03187499999999;1.03187499999999;0.963083333333331;0.963083333333331;1.03187499999999;1.03187499999999;1.03187500000000;1.10066666666666;1.10066666666666;1.10066666666666];
t_95=[0;0.0169014084507042;0.0338028169014085;0.0507042253521127;0.0676056338028169;0.0845070422535211;0.101408450704225;0.118309859154930;0.135211267605634;0.152112676056338;0.169014084507042;0.185915492957746;0.202816901408451;0.219718309859155;0.236619718309859;0.253521126760563;0.270422535211268;0.287323943661972;0.304225352112676;0.321126760563380;0.338028169014085;0.354929577464789;0.371830985915493;0.388732394366197;0.405633802816901;0.422535211267606;0.439436619718310;0.456338028169014;0.473239436619718;0.490140845070423;0.507042253521127;0.523943661971831;0.540845070422535;0.557746478873240;0.574647887323944;0.591549295774648;0.608450704225352;0.625352112676056;0.642253521126761;0.659154929577465;0.676056338028169;0.692957746478873;0.709859154929578;0.726760563380282;0.743661971830986;0.760563380281690;0.777464788732394;0.794366197183099;0.811267605633803;0.828169014084507;0.845070422535211;0.861971830985916;0.878873239436620;0.895774647887324;0.912676056338028;0.929577464788732;0.946478873239437;0.963380281690141;0.980281690140845;0.997183098591549;1.01408450704225;1.03098591549296;1.04788732394366;1.06478873239437;1.08169014084
```

```
4507;1.09859154929577;1.11549295774648;1.13239436619718;1.14929577464789;1.16
619718309859;1.18309859154930;1.20000000000000];

I=sim('Boat_model_matlab_a');

plot(t,x_dot(:,1))
hold on
plot(t_105,Vel_105,'.r')
title('Comparison Between Experimental and Analytical Solutions for Input of
105')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
legend('Model Simulation Results','Experimental Results','location','se')
grid on

figure
plot(t_95,Vel_95,'.r')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
title('Experimental Results for Input Signal of 95')
grid on
figure
plot(t_100,Vel_100,'.r')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
title('Experimental Results for Input Signal of 100')
grid on
figure
plot(t_105,Vel_105,'.r')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
title('Experimental Results for Input Signal of 105')
grid on
```

Experimental Velocity Analysis

Input 95

```
close all
%clear all
clc

load('Position95')

for i=1:72
    Time1(i,1)=Time(1,i);
end


Vxx=gradient(X(:,1))/gradient(Time1);

figure,plot(Time1,-Vxx(:,40),'.')

Vel_95=-Vxx(:,40)
```

```
t_95=Time1
```

Input 100

```
%Derivative
close all
clear all
clc


load('Position100')

for i=1:length(Time)
    Time1(i,1)=Time(1,i);
end


Vxx=gradient(X(:,1))/gradient(Time1);

figure,plot(Time1(1:43),-Vxx(1:43,35),'.')

Vel_100=-Vxx(1:43,35);
t_100=Time(1:43);


% Axx=gradient(Vxx(1:39,1))/gradient(Time1(1:39));
%
% figure,plot(Time1(1:39),Axx)
```

Input 105

```
%Derivative
close all
clear all
clc


load('Position105')

for i=1:39
    Time1(i,1)=Time(1,i);
end


%  plot(Time,X(:,1))
figure,plot(Time(1:39),Vx(1:39,1))

figure
Vxx=gradient(X(1:39,1))/gradient(Time1(1:39));
```

```
Vel_105=-Vxx(:,35);
t_105=Time(1:39);

figure,plot(Time(1:39),-Vxx(:,35),'.')

% Axx=gradient(Vxx(1:39,1))/gradient(Time1(1:39));
%
% figure,plot(Time1(1:39),Axx)
```