DESIGN AND FABRICATION OF TEST BED FOR EVALUATING THE LOCOMOTION OF A ROBOT WITH COMPLIANT ACTUATORS

By

JULIAN DONAVAN ROYAL

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Mechanical and Aerospace Engineering

Written under the direction of

Aaron Mazzeo

And approved by

_____

_____

_____

New Brunswick, New Jersey

May 2020

ABSTRACT OF THE THESIS

DESIGN AND FABRICATION OF TEST BED FOR EVALUATING THE LOCOMOTION OF A ROBOT WITH COMPLIANT ACTUATORS

By JULIAN DONAVAN ROYAL

Thesis Director:

Aaron Mazzeo

The main objective of this project was to fabricate a test bed for a robot capable of locomotion on land and in water. After completing the main objective, we desired to measure the cost of transport and compare the robot's efficiency when either the rigidity or shape of the actuator was changed. Ultimately, a robot capable of swimming using oscillating actuators was successfully fabricated. This thesis describes the mechanical design and fabrication as well as the LabView program used to control the robot and acquire data. Before fabrication, the components of the robot were modeled in SolidWorks so we could avoid having issues properly placing components. The CAD models also allowed us to improve the robot's design when these issues were noticed. The LabView program allows the operator to manually control the robot. The program also allows the robot to swim autonomously along a straight path using feedback from its IMU sensors. These sensors return information on the robot body orientation and heading. By collecting

data through video, in conjunction with a MATLAB program written to analyze the images, it will be possible to determine the robot's displacement, and furthermore, its efficiency. The results of this project will hopefully lead to improvements in current water-based robot design.

ACKNOWLEDGEMENT

First, I want to express my gratitude towards my advisor Professor Mazzeo for his guidance, patience, and encouragement while working on the amphibious robot. In addition, I would like to thank Xiyue Zou for his tremendous contributions to this project and mentoring me during this time. I would like to acknowledge Varun Suresh, Meng Yu, and Sandeep Alankar for their help in the design, fabrication, and testing of the robot. Lastly, I would like to thank the reviewers of this thesis Professor Baruh and Professor Bilgen for their willingness and patience.

Table of Contents

LIST OF FIGURES

1. INTRODUCTION

1.1 Literature Survey

In recent years, the use of robots for professional, personal, and academic purposes has become increasingly prevalent. As people find an increasing number of situations where they can be employed, engineers and researchers try to improve robot's performance or enable new capabilities. It is reported that fish can achieve propulsion efficiencies of nearly 90% while humanity's aquatic devices do not even approach that value. [1]. A number of researchers have started looking at biology for inspiration on improving robots' designs. In South Korea, a team of researchers built a soft robot that mimics how a sea turtle swims. Since motors and conventional actuators impose weight and size constraints on a design, they sought to use smart soft composites (SSC) in their place. The SSC also made it possible to mimic the motion of a turtle's flipper more accurately than if motors had been employed.[7][13]Jellyfish make use of passive energy recapture (PER) which has granted them the honor as nature's most efficient swimmers and a soft robot capable of experiencing PER was successfully created and tested.[16]

While these advancements are impressive, these robots share a disadvantage common to most robots. That disadvantage being that they can only operate in one environment: either land, sea, or air. However, researchers have already been making strides to design robots capable of traversing in multiple environments, of particular interest here are those robots which can travel on land and in water. Both SeaDog and the robot designed by Boxerbaum et al. possesses uniquely designed wheels for getting around. SeaDog's intended purpose is to be used for detecting mines. It is capable of traveling straight into the water whereupon the wheels act like the paddles on a paddle steamer.[8] The robot

designed by Boxerbaum et al. is based on the amphibious Whegs™ concept but whereas SeaDog's wheels maintain their orientation throughout operation, this robot was meant to rotate its wheels so they could function as propellers.[2]

However, if only conventional methods of propulsion are employed the only benefit would be the expansion of the robot's operational area. Thus, once again researchers have looked to nature for inspiration. Most of these amphibious robots can be divided into two broad categories: legged robots and snake-like robots. Snake-like robots excel at traversing in soft ground while legged robots perform better at overcoming obstacles and rough terrain. [11] Legged robots such as AQUA and AmphiHex have a hexapod configuration and their actuators assume a curved shape while on land. In the water, flexible oscillating actuators are used to propel the robot instead of the conventional propeller.[5][10][20][21] Originally, the actuators on AQUA had to be manually interchanged, but AmphiHex was designed so that the actuators could change their rigidity to transition between locomotion modes.

In some cases, it may be necessary to imitate nature in its entirety, but more functionality may be obtained by imitating what is needed and eschewing what is not. In China, an amphibious modular robot was created so that the robot's shape could be reconfigured to suit operators' needs.[6] Another modular robot created by a different group propels itself through undulation. It possesses a swiveling device so the robot can switch between oscillating laterally like a fish or vertically like a dolphin.[18][19]

In their paper, Buren et al. identified four major types of swimming that animals employ: Oscillatory, Undulatory, Pulsatile, and Drag-based. The robot discussed in this paper employs oscillatory swimming to propel itself. As the name implies, oscillatory

swimmers include creatures that propel themselves by oscillating a semi-rigid caudal or tail fin. Compared to drag based swimmers, oscillatory swimmers have a lower cost of transport (COT). It depends on the animal, but oscillatory swimmers also experience similar or lower COT as undulatory and pulsatile swimmers. The cost of transport is the amount of energy required to move from one location to another. Therefore, if one is to mimic an efficient mode of transport, the oscillatory method is a valid and viable option.[3] Other studies conducted with rectangular panels have found that efficiency is increases at as the flexibility of the panel increase and higher oscillating frequencies.[12][14] For certain flexibilities, since the amplitude at the trailing edge is greater than for a rigid panel, the thrust can be up to one and a half times greater.[3] Unfortunately, as speed increases fluid separation occurs along the surface of the panel and the amplitude of the trailing edge decreases. This corresponds to a drop in efficiency which is more detrimental as the flexibility of the panel increases. Worth noting is that while higher amplitudes result in greater thrust, the efficiency decreases due to higher energy input.[12][14]Thrust generated by a pitching motion is the result of added mass forces. Buren et al. present an equation for determining how much force is generated which is

$$F_{am} = m_f \frac{\partial u}{\partial t}$$

where m_f is the mass of the fluid put into motion, $\frac{\partial u}{\partial t}$ is this fluid's acceleration, and F_{am} is the force/thrust.[3]

1.2 Objective

Our main objective is to design and build a robot capable of locomotion in water. This robot will serve as a test bed for future projects that will achieve amphibious

locomotion instead of being restricted to land or water. We started from designing the robot before moving on to fabrication and verifying that it can swim. Additionally, we designed a LabVIEW program for controlling the robot and allowing it to correct its course on its own. After achieving this objective, we wished to explore the hypothesis that flexible actuators will have lower cost of transport than their rigid counterparts. To that end, several actuators of varying thicknesses and rigidity have been prepared for testing.

1.3 Mechatronic Implementation with Linear Control

The LabVIEW program makes use of PID to control the motor and other subsystems within the robot. PID is used extensively because although it is simple compared to other control systems out there it is relatively easy to understand, easy to implement, and can ensure satisfactory outcomes for a variety of processes.

For a simple open loop system, a user enters some input, this value goes to the plant which manipulates this value or uses it as is to decide what output the system should return. For many processes that is not good enough because the system cannot tell when something unexpected happens. As a result, the system cannot make adjustments to compensate for the error that will appear.

A closed loop system addresses this issue by constantly monitoring the output of interest to the user, the process variable. During each loop, the system calculates the difference between the user defined set point, and the output from the previous loop. The difference is sent to a compensator, which in our case is the PID control, which will convert the error into a command or input that the plant can use. This entire process will repeat once the plant produces another output that needs to be compared to the set point.

PID is an acronym for Proportional, Integral, and Derivative; changing these values will change how the system responds to a difference between the set point and output.

Before discussing them further four characteristics of closed-loop response should be discussed first. The first characteristic is rise time which is the amount of time it takes for the output to rise beyond 90% the desired level. Overshoot is the amount that the output goes beyond the final output. Setting time is the amount of time it takes for the system to converge to a steady state. Lastly, steady state error is the difference between the steady state output and desired output.

The equations for the proportional, integral, and derivative responses are respectively as follows,

$$P = K_P e(t)$$

$$I = K_I \int_0^t e(\tau)\, d\tau$$

$$D = K_D \frac{de(t)}{dt}$$

K_P, K_I, and K_D are known as the gains for each response. Here, $e(t)$ is the error and t is time. The proportional response is only concerned with the difference between the set point and process variable. The higher the proportional value is, the quicker the system will act to correct any deviation from the set point. However, if the proportional gain is too high the system will overshoot its target and may even cause the output to oscillate. The integral portion of PID sums up the error over time and reduces steady-state error. Additionally, it is useful for resisting outside disturbances. If the integral gain is too high the system's response to error will be sluggish. Lastly, the derivative response reduces the amount of overshoot caused by the proportional response. Similar to the integral response, if the derivative gain is too high the system's response to error will be sluggish. All three responses are summed up before being sent to the plant. To obtain a proper response time

and output value, the PID controller should be tuned before use. One method of achieving this is the Ziegler-Nichols method. This method reduces the rise time, overshoot, and setting time while minimizing steady state error. First the integral and derivative gains should be set equal to zero before slowly increasing the proportional gain. Once the output begins oscillating, one would need to record the ultimate gain at the threshold of stability, K_u, and the period of oscillation when the system becomes unstable, P_u. With these two values found, the following equations can be used to find K_P, K_I, and K_D.

$$K_P = 0.33 \times K_u$$

$$K_D = K_P \times \frac{P_u}{3}$$

$$K_I = 2 \times \frac{K_P}{P_u}$$

1.4 Preceding Work

Before work on the current robot began, a smaller prototype was created to test the feasibility of what we planned to do. The main body of the prototype consisted of two plastic containers filled with the necessary motors and electronics. The overall design was similar to that of the current version, which will be discussed in more detail later. The actuators for the prototype were made from silicone sheets. We decided to use silicone so that the actuators would remain unfurled and oscillate while the prototype was in water before curling into wheels for travel on land. During initial testing, the prototype was successful at traveling on land with these silicone fins at a speed of roughly 6 cm/s. When testing the prototype's swimming capabilities, sheets made of LDPE were used instead. This prototype did not possess the necessary hardware to communicate with an operator or travel in any direction other than forward. The succeeding version of this robot would improve in those two areas.

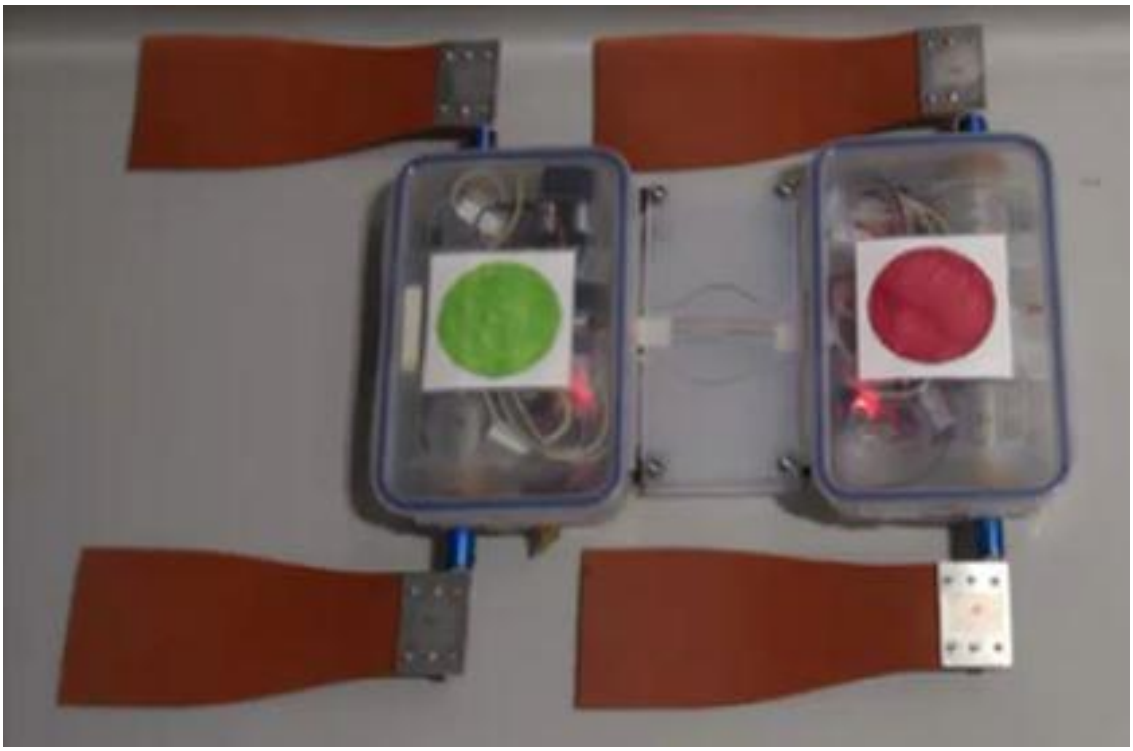


Figure 1: The prototype of the amphibious robot. The actuators shown here are made of silicone rubber. The red and green circles were used for tracking the position of the robot as it traveled forward.

2. Amphibious Robot Design

After successfully fabricating a prototype for the amphibious robot, we began searching for components and materials that would allow us to make improvements and increase its size. A larger size was necessary if we were going to include the necessary hardware to allow for more functionality. Early in this process, it was decided that the main body would be comprised of two ABS plastic junction boxes. According to the supplier, the junction boxes were waterproof and would be light enough to use in a robot that was meant to swim. A list of items necessary for the project was compiled before we began modeling each of these items in SolidWorks. This step was taken to verify that all these items would fit properly inside the robot. Completing a CAD model also helped solidify where each item

would be placed inside the robot. Seeing how much space was left inside the model of the box we created also helped us refine aspects of the design as well.

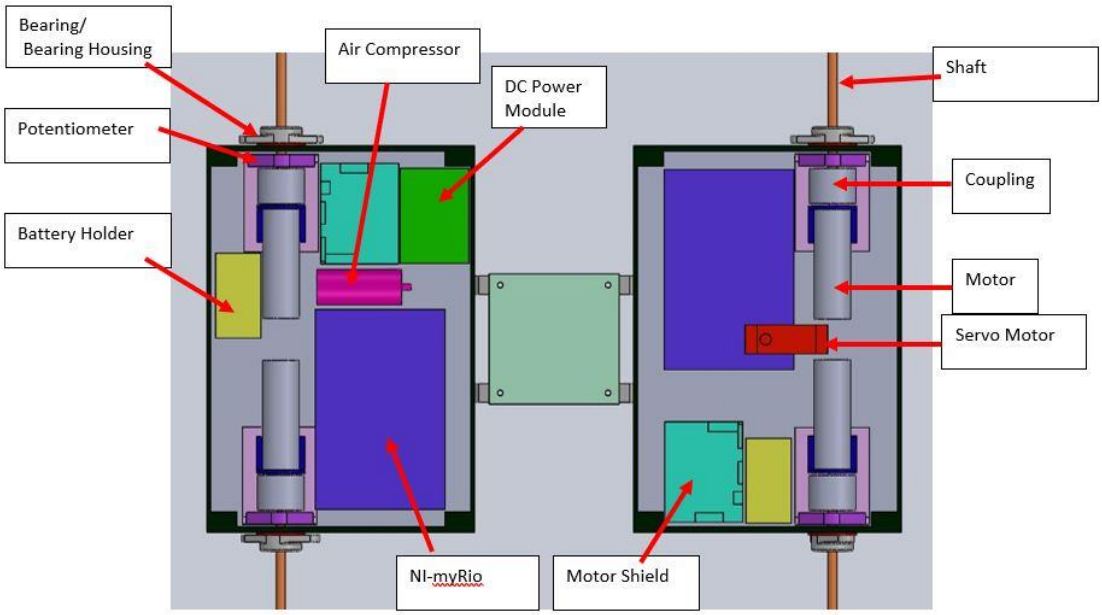


Figure 2: This is the CAD model of the amphibious robot that was created during initial design phase in SolidWorks. Some aspects of the design were changed after this model was created. For example, the width of the plate connecting the two boxes was increased while the air compressor was removed entirely.

Whatever components could not be purchased were fabricated in the school's machine shops, either by ourselves or with assistance. Assistance was necessary to ensure that the parts were fabricated correctly and according to our designs. Due to our limited experience working in the machine shop, having more experienced individuals work on these parts ensured everyone's safety as well. Once the body of the robot was complete, our efforts were refocused onto the electrical and control aspects of the project. We wanted the robot to have more functionality than the prototype and these electrical components were necessary for gathering data needed to complete our other objectives. We were

provided with one of National Instruments' myRIO embedded hardware and made use of LabVIEW to control the robot.

2.1 Mechanical Design

As stated earlier, the main sections of the robot's body are comprised of two waterproof ABS plastic junction boxes. Two acrylic plates connect these boxes together, being secured via corner brackets. Wires for communication also run between these boxes. From the front junction box, a wireless transmitter was added to allow communication between the robot and a computer. To ensure that water would not leak into the robot, every opening we made in the junction boxes was covered in flex seal. The area around the switches that power the robot on and off were covered in wax for the same reason.

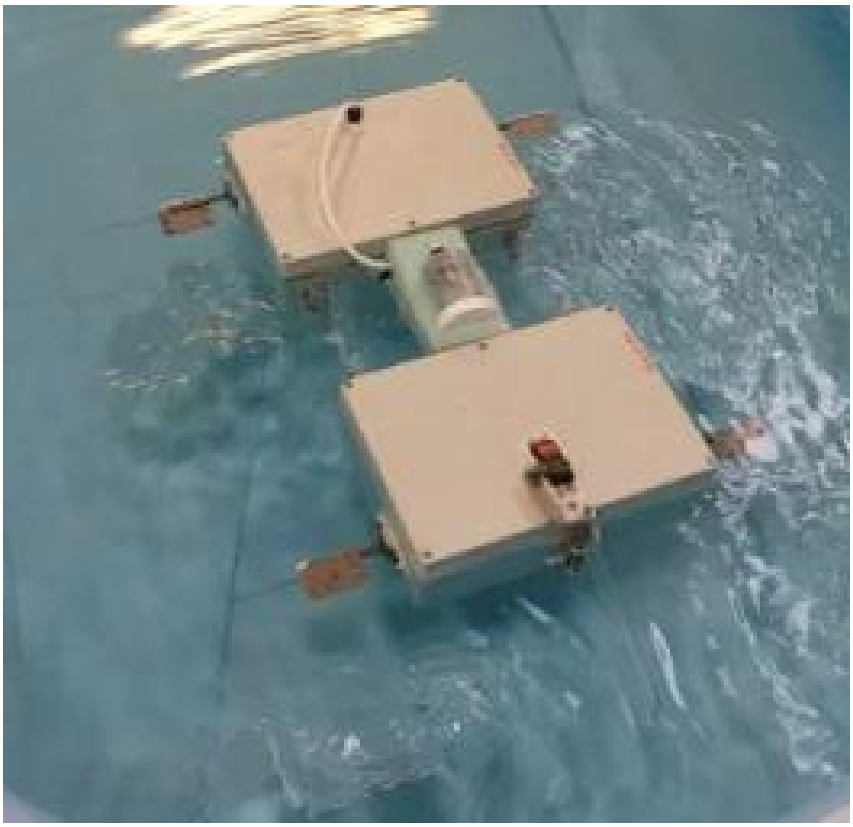


Figure 3: This is the robot swimming in a pool during initial testing. For the tests, two pools were inflated inside the lab. The inner pool held the water the robot would swim in while the outer pool ensured that no water would spill into other areas. They actuators used here were made of clear acrylic and may be difficult to identify.

A rudder was added to this version of the robot to assist in maneuvering. To create this subsystem, a hole was cut into the cover of the rear junction box in order to place a servo motor there. The main portion of the rudder consists of an acrylic blade rigidly attached to a 1/8" diameter steel shaft. The shaft is in turn held in place at the back of the robot by two low profile ball bearings. Three links was used to connect the servomotor with the shaft of the rudder. This simple setup allows us to easily translate the rotation of the servo motor to the rudder blade.

During the initial stages of this project the robot's actuators were cut from acrylic sheets using a laser cutting machine. In order to accomplish the objectives of this project, these rigid acrylic fins were eventually replaced by ones cut from LDPE plastic. These were created so that the fins could flex slightly while the robot was swimming.

Inside both boxes were acrylic baseplates meant to support the other mechanical and electrical components inside. Each box contained two motors with brackets holding them in place. These brackets were attached to aluminum blocks which were then attached to the baseplate. Couplings connected a shaft to these motors; these shafts would pass through a potentiometer before exiting the walls of the box in order to support the robot's actuators. These potentiometers were added so the robot could accurately determine how much rotation each shaft had performed. In order to have a snug fit between the shafts and potentiometers, each shaft was modified to have a D-cut. The potentiometer was attached to the base plate via a corner bracket. The robot also contained a motor shield and DC power supply module.

On the outside wall of the junction boxes where the shafts passed through, ball bearings encased in bearing housing were placed to further support the shaft. The actuators

were not connected directly to the shaft coming from the motors. Instead, small aluminum plates were made with one side connected to the actuator and the other side to the shaft.

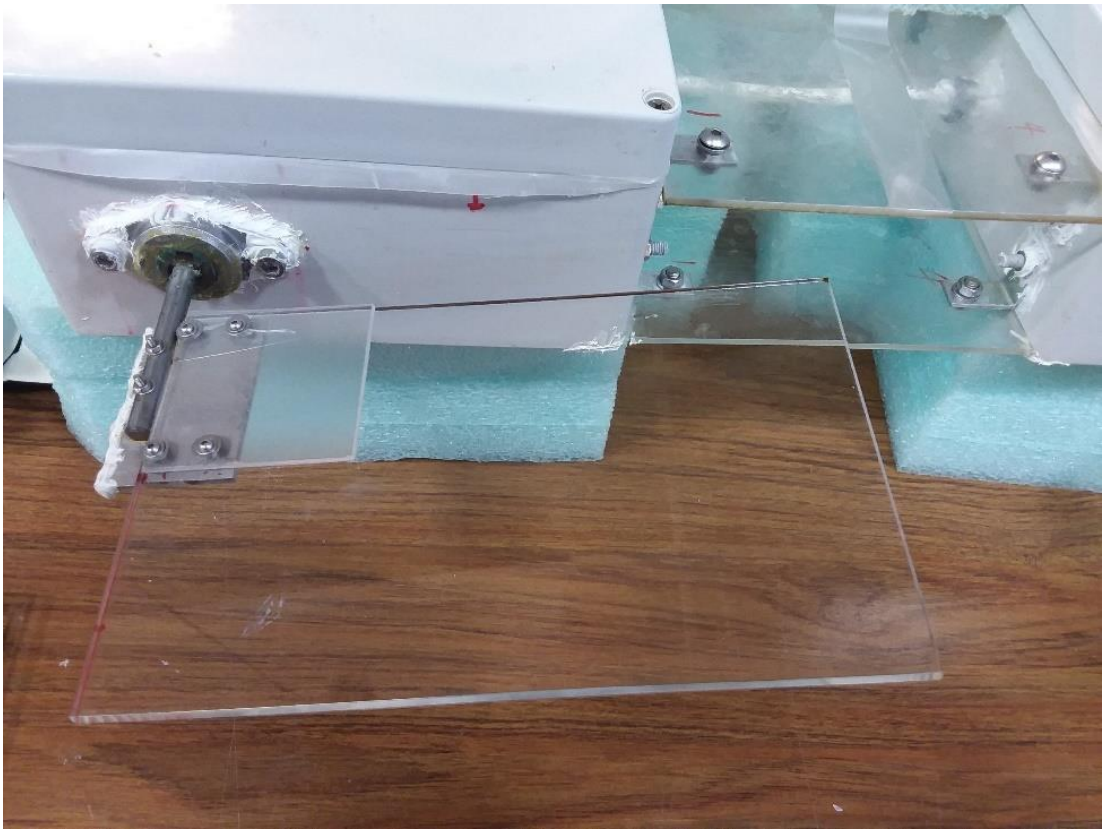


Figure 4: This is a closer look at one of the rigid actuators used on the robot.

Back inside the junction boxes, standoff spacers were used to allow for a second acrylic support platform to be placed inside. These platforms were used to hold the robot's electronics including the National Instruments myRIO microcontroller. Two battery holders, one of which could hold six AA batteries while the other held eight, were used to power to the robot and its components. These battery packs were connected to the other components after passing through a switch. This allowed the operator to power the robot on and off. Lastly, to aid in balancing the robot, lead weights were also placed inside.

2.2 Control Program

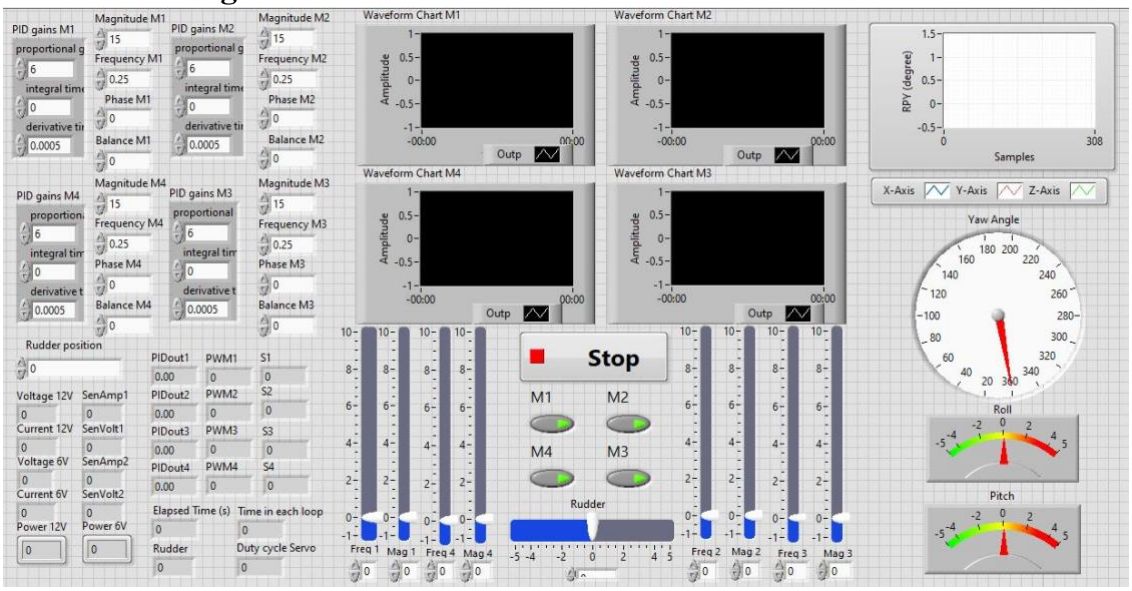


Figure 5: This figure shows the front panel of our LabView program and all of the instruments used to directly control the robot. The PID gains for each motor can be assigned before the program starts using the clusters in the top left of the image. The magnitude, frequency, phase, and balance for the motors can be set using the numeric controls or sliders. The position of the rudder is done in the same way. Whenever a pushbutton is pressed, the corresponding motor can be switched on or off without stopping the program or effecting the other motors. During operation, the four main waveform charts display the input square wave signal to each motor in red and the motor's response in white. The other waveform chart displays acceleration measurements taken by the myRIO. The numeric indicators display information such as the PID outputs, the amount of power the robot is using, and elapsed time, among others. The two meters and a gauge display the roll, pitch, and yaw measurements from the robot's IMU.

Control of the robot and data acquisition from it were performed through the myRIO hardware placed inside it. The program uploaded to the myRIO was created using LabVIEW 2018. In LabVIEW, a flat sequence structure was used to initialize, execute, and terminate the control program.

During the initialization phase, the operator can specify the magnitude, phase, frequency, and balance for each of the robot's four motors. Simply put, the magnitude determines how great of an angle the actuators will sweep through while the balance

determines about which point the actuators will oscillate. The higher the frequency is, the faster the actuators will rotate. Finally, if the phase for all of the motors is kept the same then they will behave identically. However, if one motor's phase is different form the others it will act out of sync from the others, either earlier or later depending on the entered value. Typically, after initialization the frequency, phase, and balance for all the motors were kept equal and were not changed. Only the values of the magnitudes were changed to get the robot to perform as desired. These four values are saved in the program as local variables.

Although the values must be set before running the program, the minimum PWM wave for each motor is also set at this time. The starting position of the rudder is also given during this step. The input to the rudder and motors cannot be used as is, they need to be manipulated in the program to produce the desired results. As an example, the input to the rudder needs to be converted to a duty cycle that LabView's PWM express VI can use to control the rudder's servomotor. We add 185 to the input before their sum is multiplied by 0.002444 to calibrate the servomotor. Afterwards, 0.165 is added to get the duty cycle. The frequency for the express VI was set to 330 Hz.

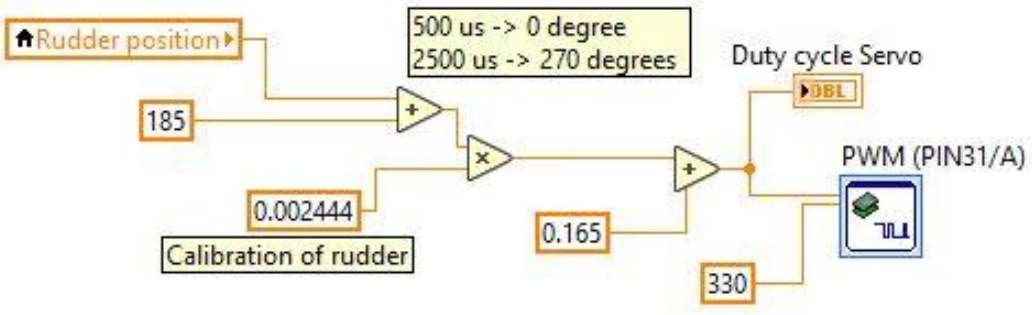


Figure 6: The block diagram used to control the servomotor that controls the robot's rudder.

The last portion of this phase involves initializing the Inertial Measurement Unit (IMU) used in the robot, a 9 DOF MPU-9250.

In the execution phase of the program, the values for the motors go to identical SubVIs that control the motors. The operator can still change the values that control the orientation of the rudder the values controlling the motors using pointer sliders located on the front panel. The values from these sliders are passed to a formula node residing in a case structure. These values are then passed outside of the case structure as the new local variables. The reason why a case structure was included was to allow the operator to switch between controlling the robot directly and allowing the robot to self-correct itself to travel straight. The self-correct mode was achieved by using magnetometer data from the MPU-9250. A simple virtual joystick was also added to make manual steering of the robot easier for an operator.

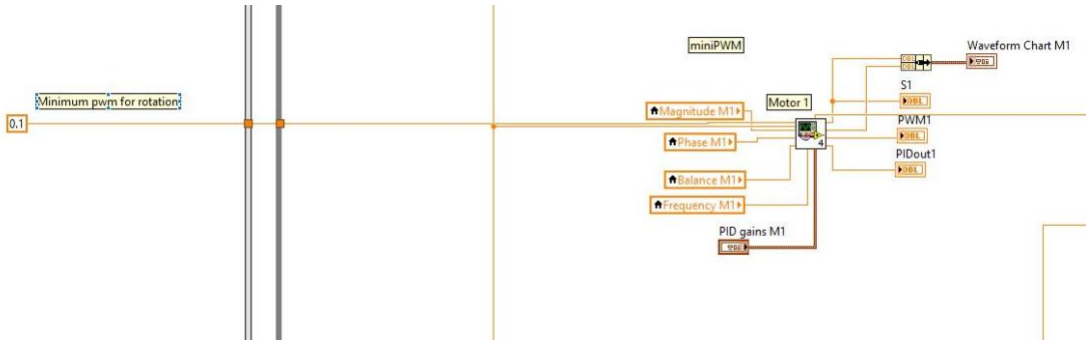


Figure 7: The minimum PWM wave and local variables are passed to the SubVI that controls motor one. The SubVI returns the motor's PID output, PWM, and output to the main VI.

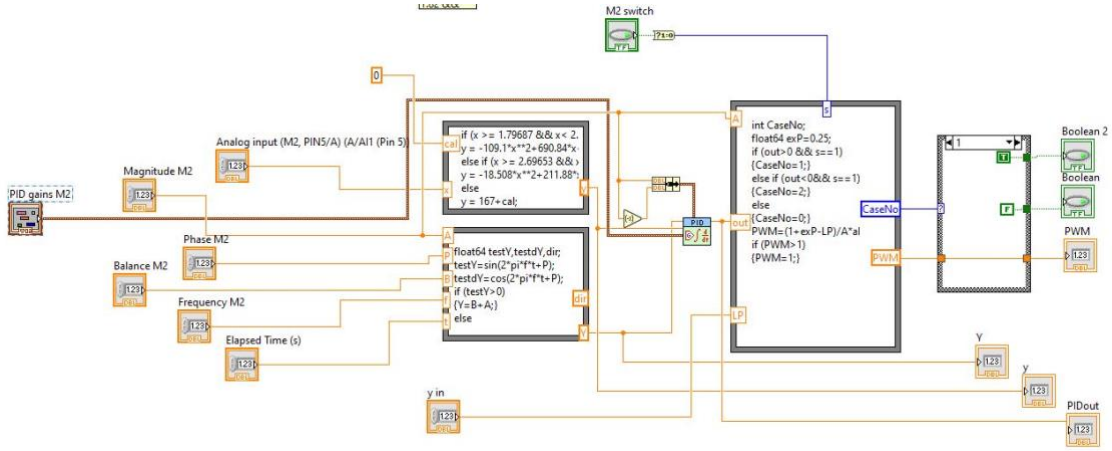


Figure 8:The contents of the motors' SubVI program in LabView. The variables passed to this VI are used to determine the motor's response to user input.

The final structures of note in the execution phase gather information on the current and voltage used by the robot from its two battery packs. These values can then be used to calculate the amount of power used by the robot. As would be expected, in the termination section of the program, the motors and rudders are all instructed to turn off.

2.3 Manual Control

During manual control, the formula node changes the entered values of frequency, magnitude, and rudder position into values the program can use. All four frequencies are manipulated using the same equation

$$F_n = 0.1 \times FREQ_n$$

where $FREQ$ is the user input, F is the frequency the motors will operate at, and n denotes which motor this information should be sent to. Similarly, all four magnitudes are manipulated with the equation,

$$Mag_n = 5 \times MAG_n.$$

Mag is the magnitude the motors will achieve while MAG is user input. The amount the rudder rotates is determined by the equation

$$Rudder = 10 \times DIR$$

with DIR being the user's input. The rudder is said to be at zero degrees when it is orthogonal to the back face of the robot. To prevent the rudder from moving too far in either direction, the formula node contains an if-elseif statement that constrains its movement to ±60º. If the value of DIR causes $Rudder$ to be greater than 60 or less than -60, the program will instruct the servomotor that controls the rudder to hold its position.

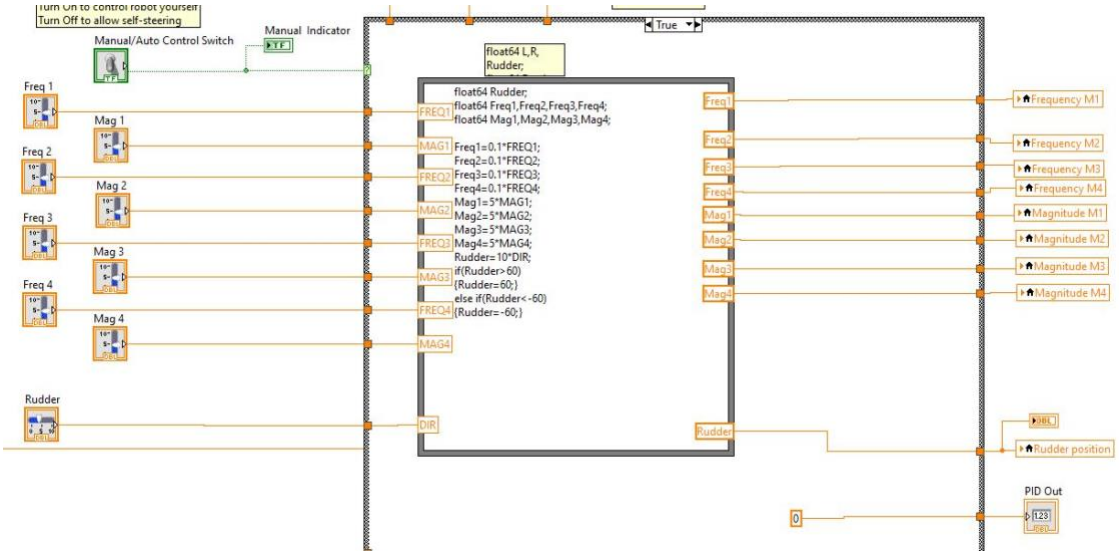


Figure 9: This is the case structure and formula node that controls the robot during manual control. A switch determines whether the case structure executes code for manual control or automatic control. The input from the point sliders for frequency and magnitude are passed to the formula node inside the case structure and execute the equations that were described previously in the main body of the text. The formula node's outputs are saved as the new local variables. During manual operation the PID output is zero.

As mentioned earlier a simple virtual joystick was added to the LabView program as well. A radio button VI was used as the basis for the front panel part of the joystick because it returns different values for each selection. Additionally, with the radio button VI, only one selection can be made at a time, so we didn't have to worry about the operator accidentally instructing the robot to move in two different directions. Radio buttons were

created to instruct the robot to head forward, turn right or left, or to make a quick right or left turn. The radio buttons were replaced with push buttons to be more aesthetically pleasing but these push buttons behaved in the same manner. After figuring out what values the radio button VI would return for each direction, we wrote the formula node to turn motors on or off depending on those values. For example, if the robot was instructed to head straight, the radio button VI would return two as the output. The formula node would receive the number two as an input and then instruct all four motors to operate. If we wanted the robot to turn in a direction, only the motors on the opposite side of the robot's body would operate. We discovered that the back motors made the robot make a sharper turn, so for a quick right turn only the back-left motor would operate. The opposite would be true if we wanted to make a quick left turn. To ensure that the normal manual operation and joystick operation wouldn't interfere with each other, we added another push button that would enable the joystick. If this pushbutton was not activated, the instructions in the joystick formula node could not execute.

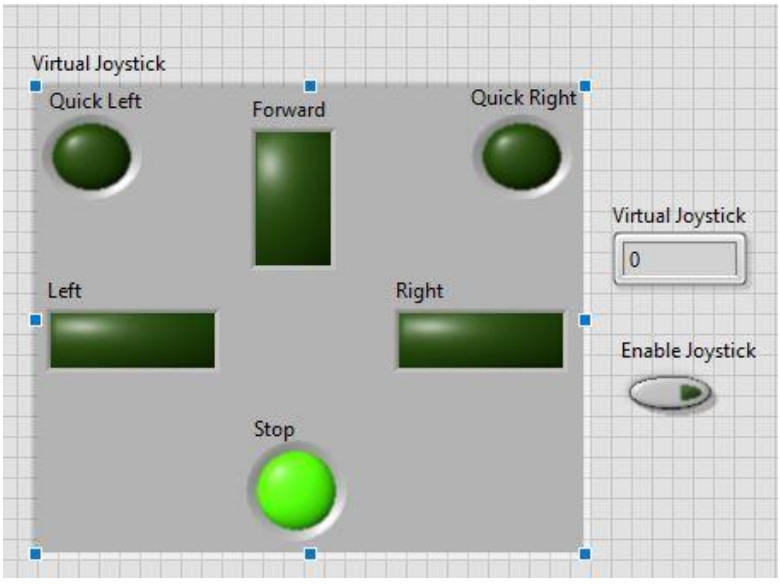


Figure 10: This is the virtual joystick that can be used as an alternate form of manual control.

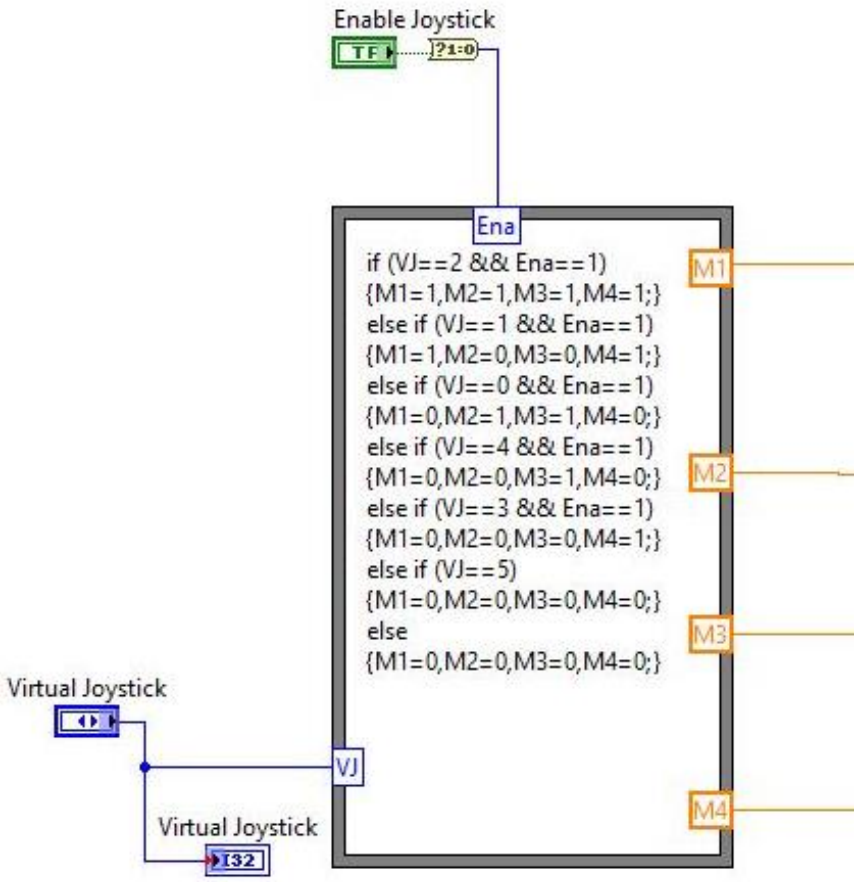


Figure 11: This figure shows the formula node associated with the virtual joystick. Depending on the input received from the radio button VI, the formula node will either instruct a motor to operate or remain still.

2.4 Automatic Control

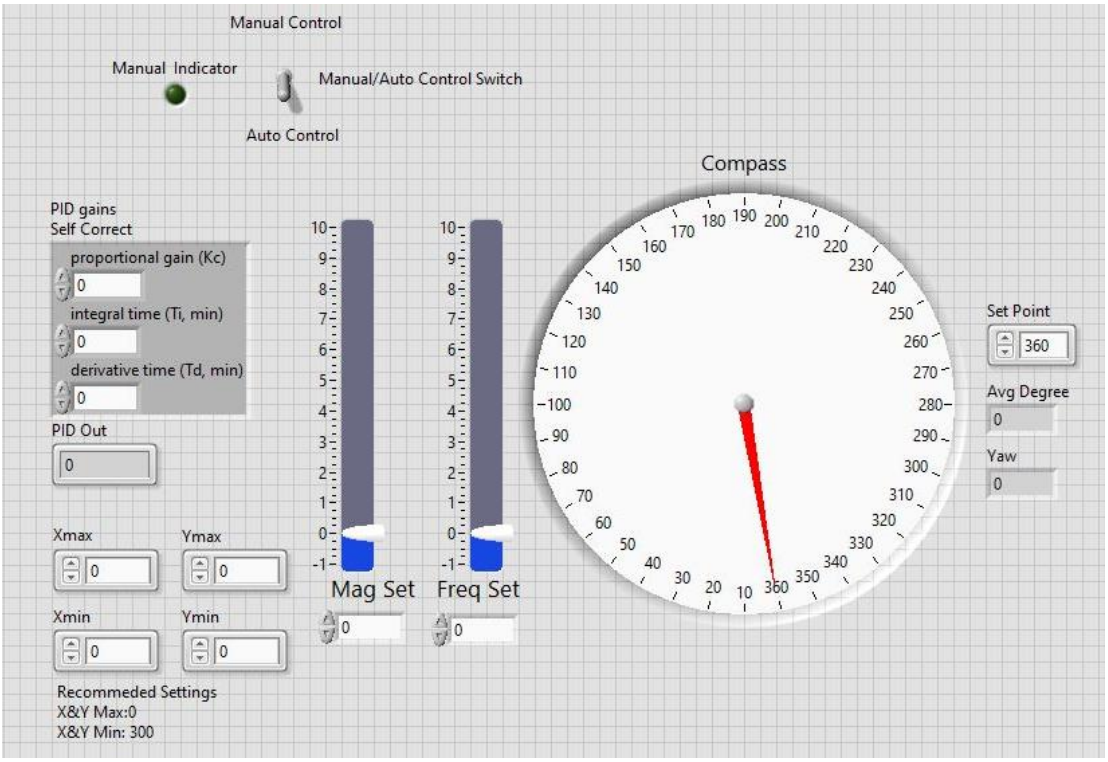


Figure 12: These are the elements on the front panel involved with the robot's automatic control. The switch allows the operator to remain in manual control or allow automatic control. An indicator was added that shines when the operator is in control of the robot. The PID gains can be set from the cluster's numeric controls. The Xmax, Xmin, Ymax, and Ymin numeric controls were added for calibrating the magnetometer and getting accurate information from it. We recommend setting the maximums to zero and the minimums to 300 so the program will record the true values during calibration. The gauge displays the heading angle from the magnetometer calculations while the set point numeric control is used to determine which direction the robot will travel in.

For experimental purposes, we wanted the robot to be able to correct its course if it started deviating from a straight course. This was achieved largely by using LabVIEW's PID VI and formula node in conjunction with magnetometer data from the MPU-9250. After calibrating the magnetometer, if the robot is held in place the displayed angle typically oscillates ± 5º about a point. A rough estimate for this mean value can be entered as the set point for the PID VI. The actual data is passed to this same VI as the process variable. Another formula node was added to more accurately calculate the average yaw

angle while the robot is stationary. This was added so the user doesn't need to rely on dead reckoning alone to pick a set point.

All the equations used during automatic steering are the same as those used during manual control. The only difference is that the output from the PID VI, which ranges between ±30, determines which combination of these equations are used. Since the values from the magnetometer fluctuate by a small amount, the range of the PID output could not be too small otherwise it was found that the output would rapidly shift from one extreme value to the other. On the other hand, if the range were too large, the robot's correcting response would be too slow. As the requirements become more lax, the robot would be permitted to drift for a longer amount of time defeating the purpose of having this functionality in the first place. That is the reason why a range of ± 30 was chosen; it is large enough so that the robot will not switch between two extreme responses while being small enough so that appropriate action can be taken in a timely manner.

Unlike with manual operation, there is only one slider bar each for frequency and magnitude selection. In the formula node the equations for frequency remain the same as with manual operation, which was discussed earlier. Thus, during automatic steering all motors operate at the same frequency all the time. In an ideal situation where the robot travels perfectly straight, the PID VI would have an output of zero. As the robot drifts to the left the value of the output increases until it reaches thirty. Naturally, as the robot drifts to the right, the value of the output instead decreases towards a minimum value of -30. The formula was written in such a way that if the value of the PID output was between ±15, the magnitude of the motors would all remain the same. The equation to convert from user input to what the commands the motor receives remains the same as with manual operation.

If the PID output becomes greater than 15 but remains less than 30 the magnitude of the two motors on the left are decreased. Instead of being multiplied by five, the inputted magnitude of motor two, which is the motor located on the front left motor side of the robot's body, is multiplied by three. The magnitude of motor three which is located behind motor two is set to zero. The reason why the magnitude of the motor three is set to zero is because during testing it was observed that the back motors have a greater contribution to the speed of turning than the front motors. Once the PID output equals 30, the magnitudes of both motor two and motor three are set equal to zero. A similar operation occurs if the robot starts drifting to the right. When the PID output is between -30 and -15, the magnitude of the back-right motor, motor four, is set to zero while the equation for motor one's magnitude becomes

$$Mag_1 = 3 \times MAG_1.$$

If the robot continues to drift to the right, the magnitudes of both motors on that side are set to zero. The rudder is unaffected by the switch from manual to automatic control and can be controlled in the same manner as before.

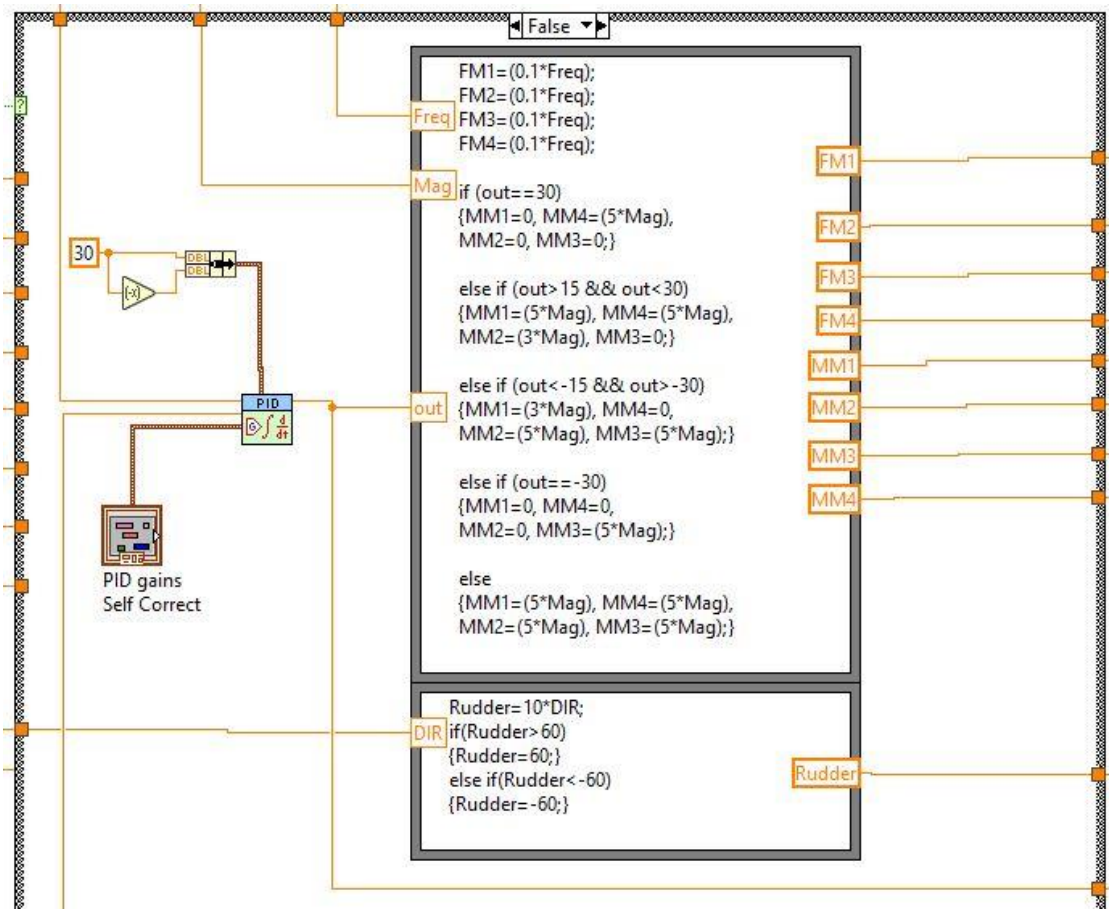


Figure 13: This is the formula node containing the equations that control the robot during automatic control. The back-panel icons for the PID cluster and the PID VI can also be seen in this image.

2.5 MPU-9250 - Accelerometer

The MPU-9250 consists of accelerometer, gyroscope, and magnetometer sensors. In the control program, the roll and pitch are calculated using data from the gyroscope while yaw is determined by the magnetometer.

An accelerometer consists of a mass, referred to as a proof mass, supported by a spring. By making use of Newton's second law and Hooke's Law, an equation for the proof mass in the inertial reference plane can be written as

$$m\ddot{x}_m = F_s - mg$$

Where m is mass, F_s is the force on the spring, g is gravitational acceleration, and $\ddot{x}_m$ is the mass' acceleration. From Hooke's Law we know that the force on the spring is equal to the multiplication of the spring constant, k, and the spring's displacement, d.

$$F_s = kd$$

So, our first equation for acceleration can be rewritten as

$$m\ddot{x}_m = kd - mg$$

The natural length of the spring is represented by l_o in the next equation. In our inertial reference frame, when the spring's natural length and displacement are subtracted from the position of the body, we find the position of the proof mass.

$$x_m = x_b - (l_o + d)$$

If we take the double derivative of the previous equation and substitute it for $\ddot{x}_m$ in our rewritten equation for acceleration we get

$$\ddot{x}_b - \ddot{d} = \frac{1}{m}(kd - mg)$$

We assume $\ddot{d} = 0$ and recognize $\ddot{x}_b$ as the acceleration of the accelerometer, a, so the equation can be rewritten and rearranged to become

$$d = \frac{m}{k}(a + g)$$

Using this relationship, an accelerometer can tell us its acceleration by measuring its spring's displacement. Depending on the orientation of a stationary sensor, instead of zero it will measure a positive acceleration of one due to gravitational acceleration. After

converting the acceleration experienced by the accelerometer in the body frame to the world frame it is possible to obtain velocity by integration and position by performing an additional integration.

$$v = \int a(t)dt$$

$$p = \int v(t)dt$$

Unfortunately, we were unable to properly perform these integrations from our acceleration data in LabView. Partially as a result, we would rely on cameras and a MATLAB tracking program to determine the robot's position.

Although the data is not being used for further calculations, the robot's acceleration in the X, Y, and Z directions is determined by sending the raw data through several Max and Min function VI's to find the median offset, or bias. This bias is subtracted from the raw data before we divide the new values by the scaling factor of 8192. Dividing by the scaling factor converts the values from LSB's into g's.

2.6 MPU-9250 - Gyroscope

The gyroscopic sensor MEMS can detect orthogonal displacements in the plane when the plane is rotated about an axis normal to it. The rotation about the axis is measured and the sensor returns the angular velocity. This angular velocity is measured in the sensor's body frame.

When we try to turn a classical gyroscope, it resists the change and instead turns in a different direction. This behavior can be descried by the equation

$$\tau = \omega \times h$$

Where τ is torque, ω is the gyroscope's angular velocity, and h is angular momentum. The angular momentum acts parallel to the axis of spin. The magnitude of the angular momentum is equal to the gyroscope's inertia, J, times its rotational speed, $\bar{\omega}$.

$$\|h\| = J\bar{\omega}$$

Within our LabView program, the information obtained by the gyroscope was used to determine the robot's roll and pitch. The raw gyroscope data in the X and Y directions was passed through several Max and Min function VI's in order to find the median of the two values. This median value was the bias for each value and was subsequently subtracted from the raw data. After removing the bias, the data needs to be divided by the appropriate sensitivity scale factor to convert the units from LSB to degrees/sec. In our case, we divided the data by 65.5. We used LabView's Elapsed Time Express VI in conjunction with a formula node to determine how much time had passed between each instance of the IMU recording data. The express VI sent the number of seconds that had elapsed since the program started to the formula node. During each iteration, the formula node would subtract the current and previous number of seconds to determine the change in time between each reading.

This time interval was multiplied by the X and Y data to convert our readings from degrees/sec to degrees. The pitch and roll angles were initialized at zero each time the program started. When the gyroscope started collecting data, the change in degrees it detected was added to the previous iteration's angle. We were able to obtain reasonable pitch and roll readings when the robot was first turned on. However, as time progresses, the readings would become less accurate because of the sensor's tendency to drift. Since

the magnetometer was less susceptible to drift than our gyroscope, we decided not to use the gyroscope to also determine the yaw angle.

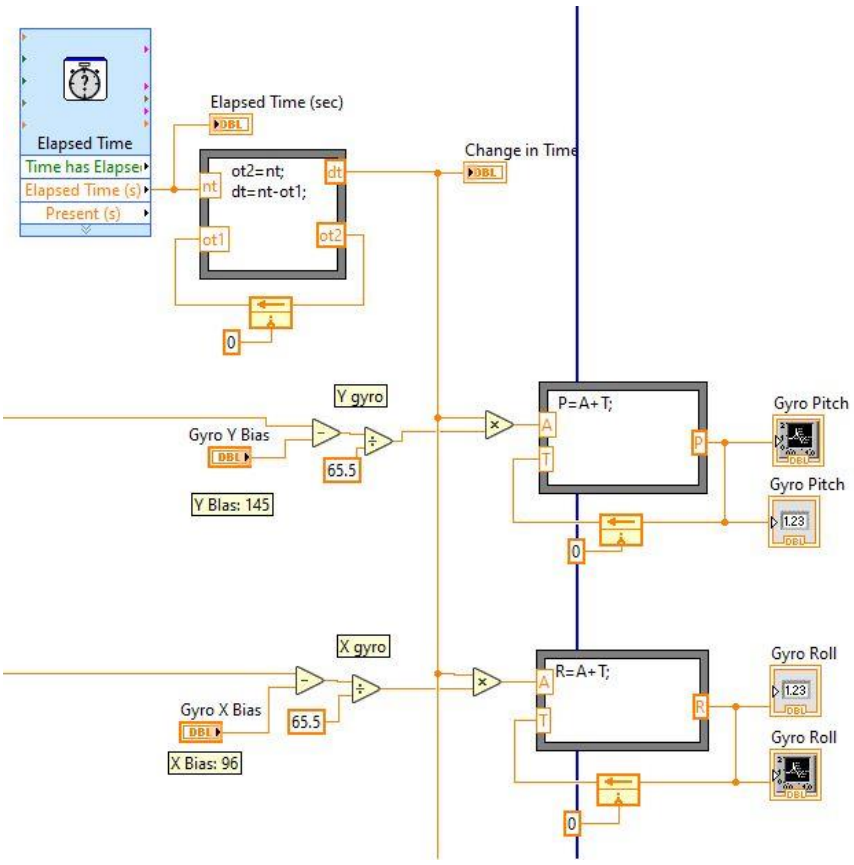


Figure 14: This figure shows how roll and pitch angles are determined in the LabView program. The Max and Min function VI's are not displayed in the image.

2.7 MPU-9250 - Magnetometer

Magnetometers use Hall-effect sensors to produce a voltage that's proportional to the magnetic field's strength. The magnetic field that's being measured will be normal to the current flow through the sensor.

Each time the robot is powered on, it needs to be rotated 360 degrees to calibrate the magnetometer. To obtain accurate information from the magnetometer, we first need to eliminate errors caused by hard and soft iron distortions. Ideally, when the magnetometer

is rotated in a full circle, the data points should create a circle centered at the origin of a cartesian plane. Hard iron distortions will translate this circle in the X and Y directions. Objects that produce magnetic fields, such as electric motors, will cause hard iron distortions. In contrast, soft iron distortions will not translate the circle to a new position, but they will distort its shape to that of an ellipse. This distortion is caused by metals such as iron or nickel being present near the magnetometer.

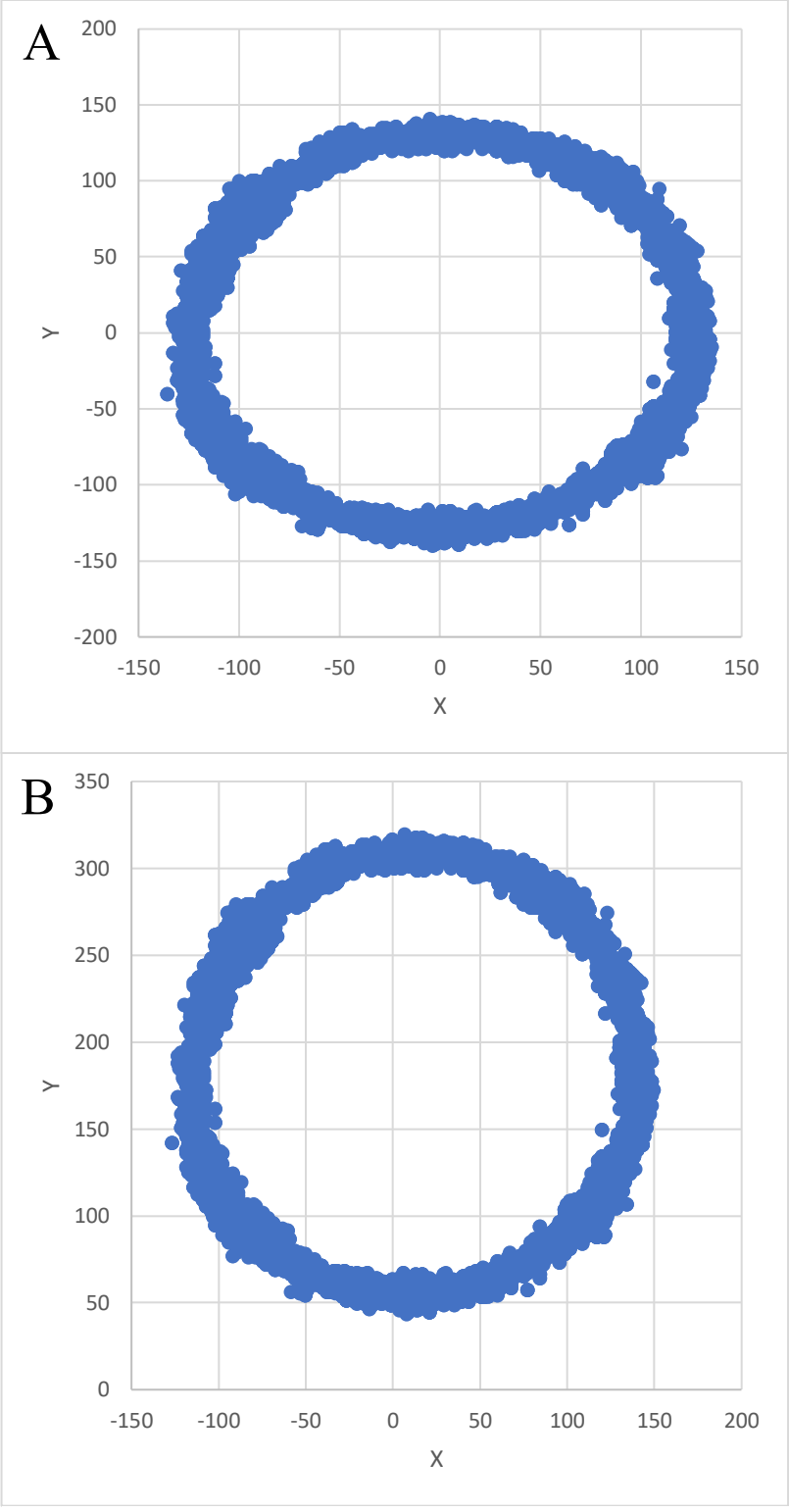


Figure 15: (a) This is what a graph of the magnetometer data would look like if only the hard iron distortion was corrected. Note the elliptical shape the data forms. (b) If only soft iron distortion is corrected the data will appear circular, but it is not centered at the origin.

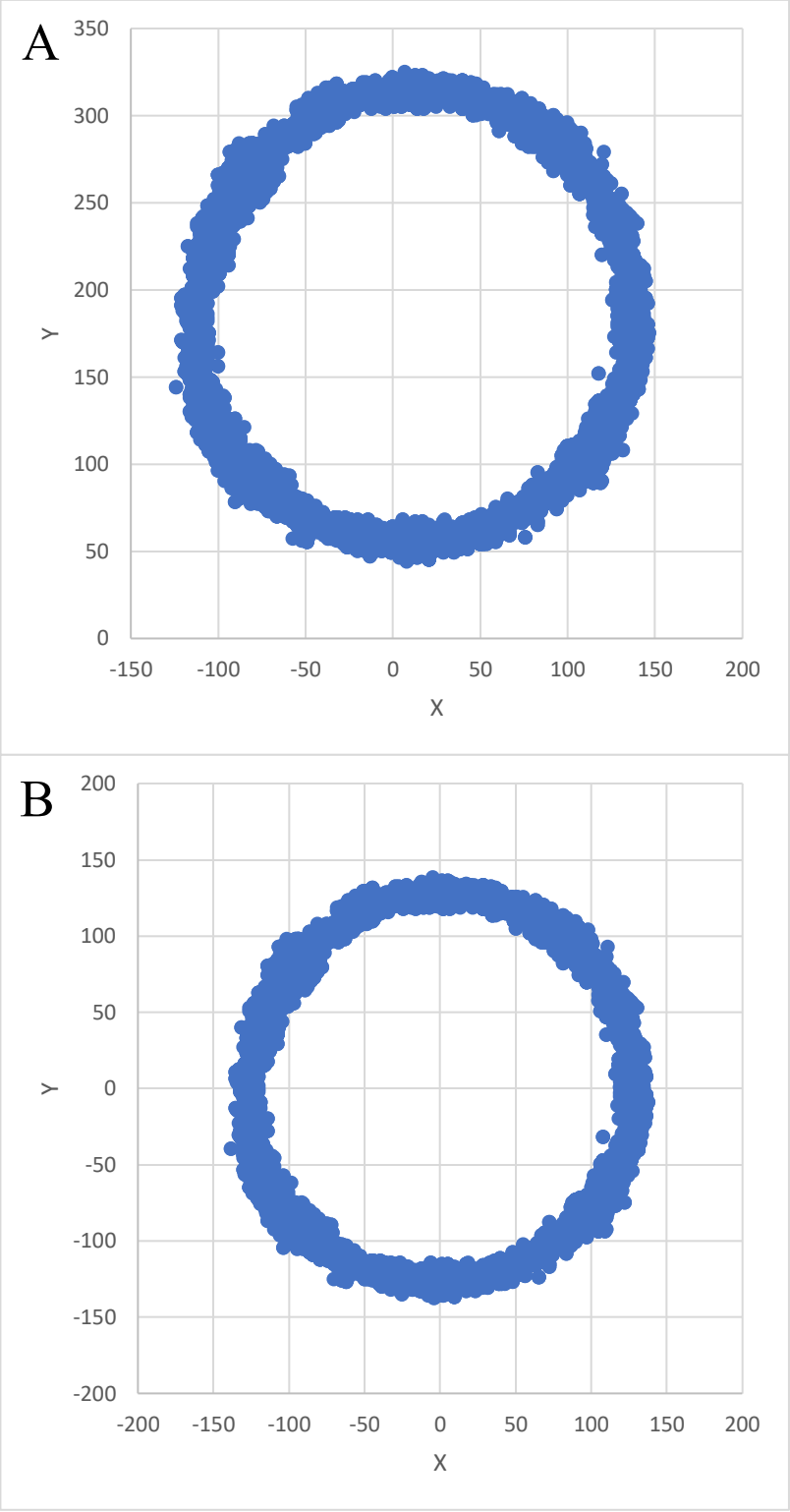


Figure 16: (a) The raw data from the magnetometer. Although it appears circular, it is not centered at the origin. (b) This is how the data appears when the hard and soft iron distortions have been removed.

To eliminate both of these errors, the program was written so that the raw X and Y data from the magnetometer would pass through several Max and Min function VI's. These four new values (X_{max}, Y_{max}, X_{min}, Y_{min}) are then passed to a formula node along with the raw X and Y values. What follows is an explanation of the equations LabView uses to correct hard and soft iron distortions. The equations for X and Y are identical so only the equations for X will be shown here. The offset for hard error distortion is calculated using the equations

$$HX = \frac{X_{max}+X_{min}}{2}.$$

Subtracting this offset from the raw data will return the center of the circle to the origin. The process for correcting soft iron distortion is slightly more involved. Normally, one would use a 3x3 transformation matrix to correct the data, but for our purposes it was sufficient to re-scale the data. First, we need to find the delta for both X and Y using the equations

$$DX = \frac{X_{max} - X_{min}}{2}$$

and

$$DY = \frac{Y_{max}-Y_{min}}{2}.$$

The average delta is found by adding DX and DY before dividing by two

$$AD = \frac{DX-DY}{2}.$$

The scaling factor for X or Y is found by dividing the average delta by the corresponding delta. For example, to find the scaling factor for X you would use the equation

$$SX = \frac{AD}{DX}$$

and for SY you would perform a similar operation. By combining the raw data, offset, and scaling factor into a single equation we can get accurate readings from the magnetometer. The final equations are as follows

$$X_{corrected} = SX \times (X - HX)$$

and

$$Y_{corrected} = SY \times (Y - HY).$$

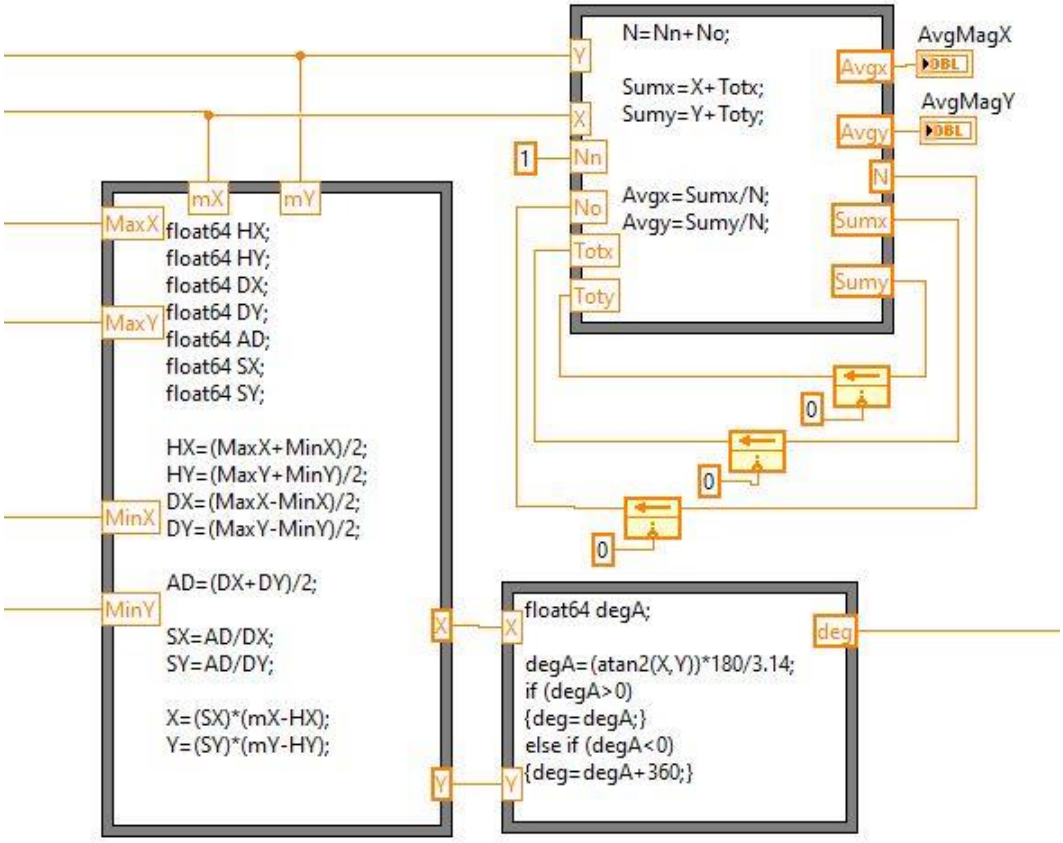


Figure 17: The formula node on the left contains the equations to remove hard and soft iron distortions from the magnetometer data. The corrected values of X and Y are sent to the formula node in the lower right of the image. The values of X and Y are used to find the angle of the robot's heading.

Using these corrected X and Y values, we can then calculate an angle using arctangent. The units were converted from radians to degrees to make interpretation of the results easier to understand. The equation is as follows

$$\left(\frac{180}{3.14}\right) \times arctan\left({X_{corrected}}/{Y_{corrected}}\right).$$

If the value of the angle returned by LabView is negative, an additional 360 to the value. It was decided to use the magnetometer to determine yaw instead of the gyroscope because the magnetometer was less susceptible to drift over time.

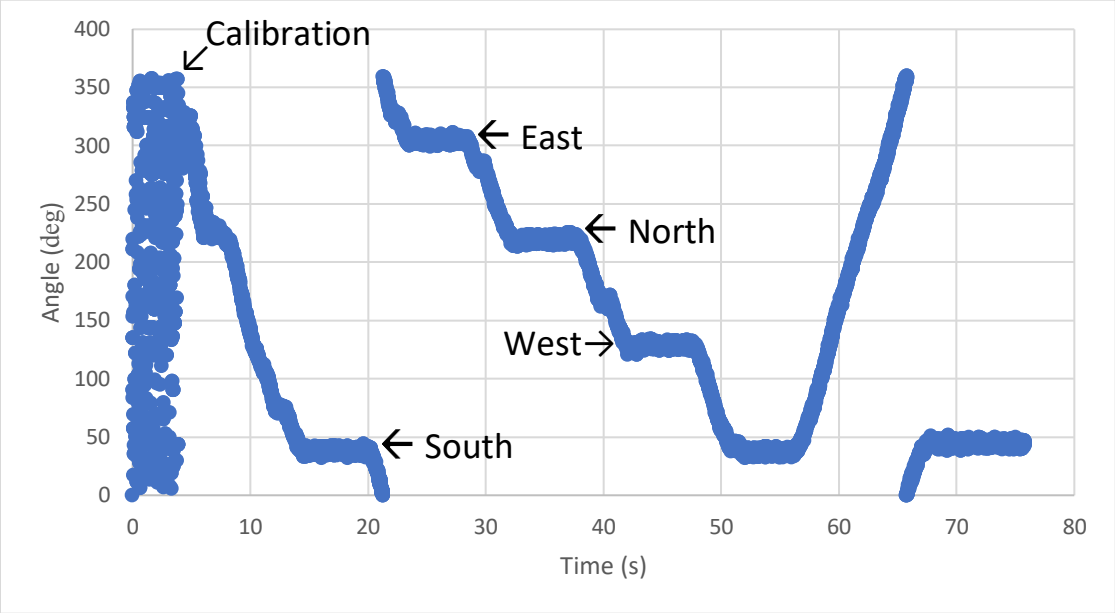


Figure 18: This graph shows the angle of the robot's heading over time. The uninterpretable mass of data points on the left are from before the robot is calibrated. The robot must be spun 360 degrees to capture the maximum and minimum X and Y values. Only after this is done will the data start to make sense. After conducting tests, we found that the average angles for South, North, East, and West are approximately 35º, 220º, 305º, and 125º, respectively.

Figure 18 shows the results of a test illustrating how the magnetometer could be used to identify the robot's heading. To get this data, the robot was placed atop a table and spun 360 degrees. Using the compass app on a phone, we were able to mark the four cardinal

directions for reference. Due to how the program was written, every time the robot is turned on the magnetometer needs to be calibrated by spinning it in a complete circle. If the calibration is not performed at the start, the magnetometer returns unintelligible data as can be seen on the far-left side of the chart. During the test, the robot was made to face each cardinal direction, held in position for a few seconds, before being made to face the next cardinal direction until they were all covered. As was hoped, when the robot was still, the magnetometer readings oscillated around an average value. These values were separated from one another by the appropriate amount. What we mean is that one would expect the values corresponding to North and West to be 90º apart, North and South are 180º apart, and so on which is reflected in the figure. A clockwise rotation can be identified by its positive slope while the opposite rotation results in negative slope. Since the values must be between zero and 360, whenever the robot passes that imaginary boundary there's a break in the graph. At the moment, North, South, East, and West are not represented by 0º/360º, 180º, 90º, and 270º respectively because we suspect that the constant values found here will change based on geographic location. If we desired to have a more conventional designation for these directions, we'd only need to add a constant to our calculations from earlier.

2.8 Kalman Filter

A Kalman filter is useful for ignoring the random errors that are present in a sensor's readings. Through an iterative process, the Kalman filter can estimate, with a certain amount of uncertainty, the true value of the parameter being measured. The readings we receive from the myRIO and MPU-9250 also fluctuate around the true value for acceleration and angular velocity, so we wished to use a Kalman filter to get more accurate data to use in our equations for roll and pitch.

The Kalman gain, KG, is the ratio between the error in the estimate, E_{EST}, and the sum of the error in the measurement, E_{MEA}, and the error in the estimate. For our calculations it should have been sufficient to use the variance of the data or the published sensor accuracy provided by the manufacturer as the value for, E_{MEA}.

$$KG = \frac{E_{EST}}{E_{EST} + E_{MEA}}$$

The value of the Kalman gain ranges from zero to one. The current estimate, EST_t, is the estimate from the previous iteration, EST_{t-1}, plus the Kalman gain multiplied by the difference between the current measurement, MEA, and previous estimate. When the Kalman gain is large, this means the error in the estimate is large compared to the error in the measurement. That being the case, we want to rely more heavily on the measurement to compute the current estimate. If the Kalman gain is small, we assume there's significant uncertainty in the measurement and rely more heavily on our estimate from the previous iteration.

$$EST_t = EST_{t-1} + KG \times (MEA - EST_{t-1})$$

The current error in the estimate, E_{EST_t}, is calculated using the equation below. The error in the estimate from the previous iteration, $E_{EST_{t-1}}$, is multiplied by on minus the Kalman gain. Again, if the Kalman gain is large we have confidence in the measurements so the error in the estimate will quickly decrease and the filter will converge to a value close to the measurements we're receiving. On the other hand, if the Kalman gain is small the error in the estimate will remain large so the filter slowly converges to the true value without being thrown off by errors in the measurement.

$$E_{EST_t} = (1 - KG) \times E_{EST_{t-1}}$$

Instead of using a matrix, our LabView program consisted of three sets of the previously discussed equations: one set for the X, Y, and Z axes. After calculating E_{EST_t}, it would be used as the value of E_{EST} in the next iteration while EST_t would be used as the new EST_{t-1}.

After writing this code in LabView we tested its capabilities using acceleration data from the IMU. Figure 19 shows that our Kalman filter successfully removed the noise in the signal and converged to the mean value. Unfortunately, upon further evaluation we realized that this Kalman filter was slow to reflect changes in values when the IMU was moved. Additionally, if the MPU experienced rapid or extreme changes the filter could not keep up and would stop returning data. For these reasons, we ultimately decided not to continue using this Kalman filter in the robot's control program.

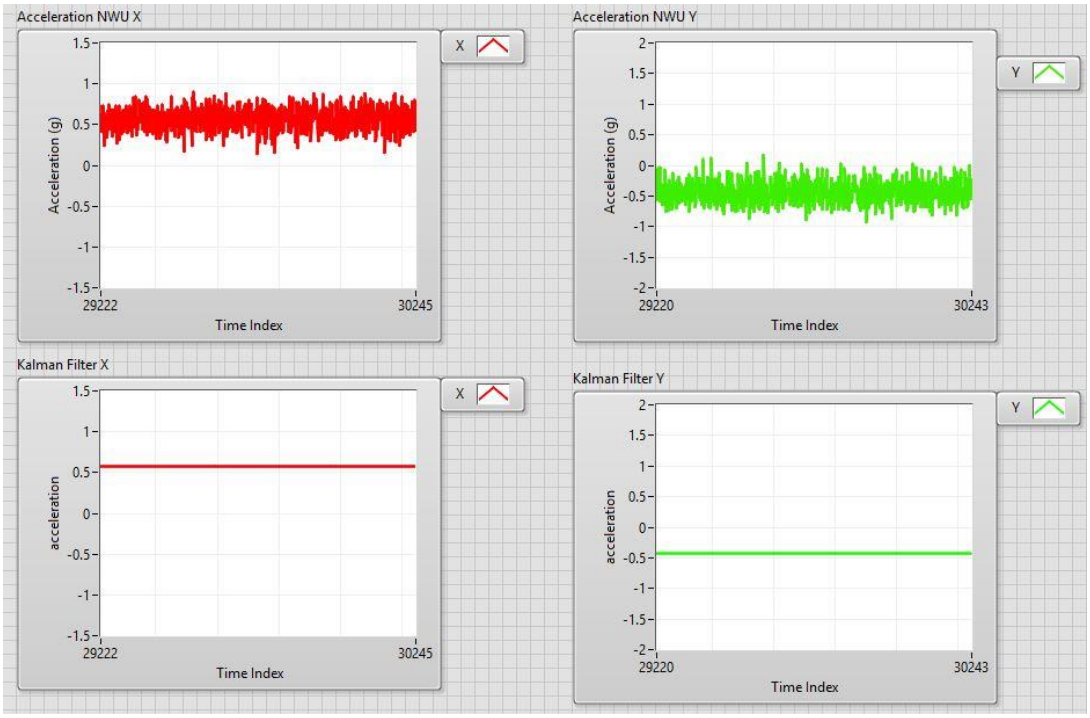


Figure 19: The wave charts at the top of the figure show the raw acceleration data in the X and Y directions obtained by the IMU. Below them are graphs showing how our Kalman filter successfully removed the oscillations in the measurements.

3. CONSTRUCTION

After receiving all the components that were ordered, we set about constructing the robot. A drill was used to create all the necessary holes in the sides and bottom of the junction boxes. From the CAD drawings, we knew the distance from the edges of each box to the center of the holes we wished to make. These locations were carefully measured with a ruler and marked using pencil. While one person held the box firmly in place, another person made a pilot hole in its side. The pilot hole made it easier to drill the final hole without having to worry about the drill slipping. If the pilot hole was slightly off the mark, we would reposition the drill accordingly and make a new hole. Thankfully, the pilot hole was never too far from the intended position, so the error would be removed once the larger hole was made. The holes drilled in the bottom of the junction boxes were made so the base plates inside could be fastened in place by screws.

Originally the baseplates had holes cut out in specific places where we could attach the electrical components and motors. After we discovered small errors in the holes' placement, they were made into slots so the position of the motor shafts could align with the holes in the side of the junction boxes.

The actuators, baseplates, rudder, and connecting sheets between the junction boxes were all cut from acrylic plastic on a laser cutter. The component would first be modeled in SolidWorks and saved as a DXF file. This DXF file was then imported into Inkscape where it could be converted to a pdf. The laser cutter we used would use this pdf to determine what cuts to make. Before starting the laser cutter, the settings had to be adjusted so the laser cutter knew what thickness to cut through and what material was being cut. Properly setting these parameters was especially important when cutting the flexible actuators because if the thickness selected on the laser cutter was too large, the LDPE

plastic would melt and ruin the actuator. As a result, when we cut out the flexible actuators, we set the thickness to one-third of the material's real thickness. Using the shallow outline as a guide, we would finish cutting out the actuator using a boxcutter. For the rigid acrylic actuators, the laser cutter would also cut out the holes. For the flexible actuators we used a drill to make them. The hole in the cover of the rear junction box that allows the servomotor to sit at the top and control the rudder was also cut out using the laser cutter.

To fabricate and modify several of the metal components on the robot it was necessary to use the machines in the machine shop. The rudder shaft, its connecting arm, and the actuator support were fabricated for us by the more experienced instructors and student attendants in the school's machine shop. We created drawing sketches in SolidWorks and communicated with these individuals to ensure the parts were made as we envisioned them. In the case of the connecting arm and rudder shaft, we were told our designs were too difficult to feasibly fabricate and had to make the adjustments to the drawing that were suggested to us.

Since one side of the couplings we used to connect the motors' shafts with the shafts extending outside of the robot needed to be enlarged, we used the lathe in the machine shop to do so. The third link in the rudder subsystem was also fabricated there. We used the bandsaw to cut a piece of metal down to size before rounding the edges at the belt sander. Lastly, the holes were created on the drill press. Almost every part of the robot used screws to fasten in place, so once the parts were ready, we fastened them in place.

Lead weights covered in plastic were placed inside the robot to help it balance in the water. The weights were covered in plastic to avoid touching the lead directly with our hands at a later time. Before placing the robot in water, we had to waterproof it to protect

the electronics inside. Around most places where we had made openings in the junction boxes, a layer of flex glue was applied. Since the switches that power the robot on needed to move, wax was applied around their bases to keep water out. This layer of wax needed to be checked every time before placing the test bed in water and reapplied if necessary. Around the edge of the junction boxes lids, sheets of parafilm were also placed to keep water out. The reasoning was that once the lid was screwed in place the parafilm would fill any gaps that remain between the lid and the body of the junction box.

4. EXPERIMENTAL SETUP

Due to the dangers posed by Covid-19, the university was shut down and it became impossible to perform swimming experiments with the amphibious robot test bed. However, the following section will describe the methods we planned to use to gather data and interpret the results of its performance.

To prove that our methods would have worked we performed tests with a robot that had been built previously. This robot was a simple four wheeled and remote-controlled vehicle. A red circle, five inches in diameter was printed on computer paper and taped to a cardboard support. This support was then attached on top of the robot as can be seen in Figure 20. Every 30 centimeters, a marking was made on the wall behind the robot. These markings were necessary to act as references so the distance traveled by the robot could be determined. Since the robot was simple, it was difficult to stop at a specific location each time. To ensure that the robot started at the same position and traveled the same distance for each testing run, constraints were placed at the beginning and end of the testing area. A phone camera was used to record video of the tests since a camcorder was not available.

The first step in the procedure we took was to place the robot in position and turn it on. Afterwards, the phone was placed so that it could capture the testing area in its entirety. After starting the video, we would return to a place behind the robot where the remote could communicate with the robot and drive it forward until it hit the constraint at the far end. The video recordings from the phone would be uploaded to a computer. For the MATLAB code to function, it was necessary to save the recording in the same folder where the code was.

Figure 21 shows the distance the robot traveled over time from four different testing runs. These tests were carried out to prove that the MATLAB code would provide reliable and consistent results. As the graph shows, MATLAB produced nearly identical results each time the robot was recorded. Additionally, the results agree with what could be observed visually without the help of MATLAB.

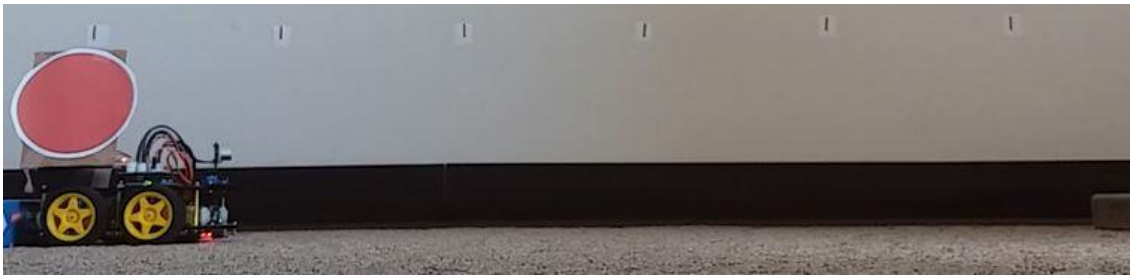


Figure 20: The robot used for testing the effectiveness of the MATLAB code. A red circle five inches in diameter was attached to a cardboard support which was itself attached to the top of the robot. Markings were placed thirty centimeters apart on the wall behind the robot so we could determine how far it had traveled.

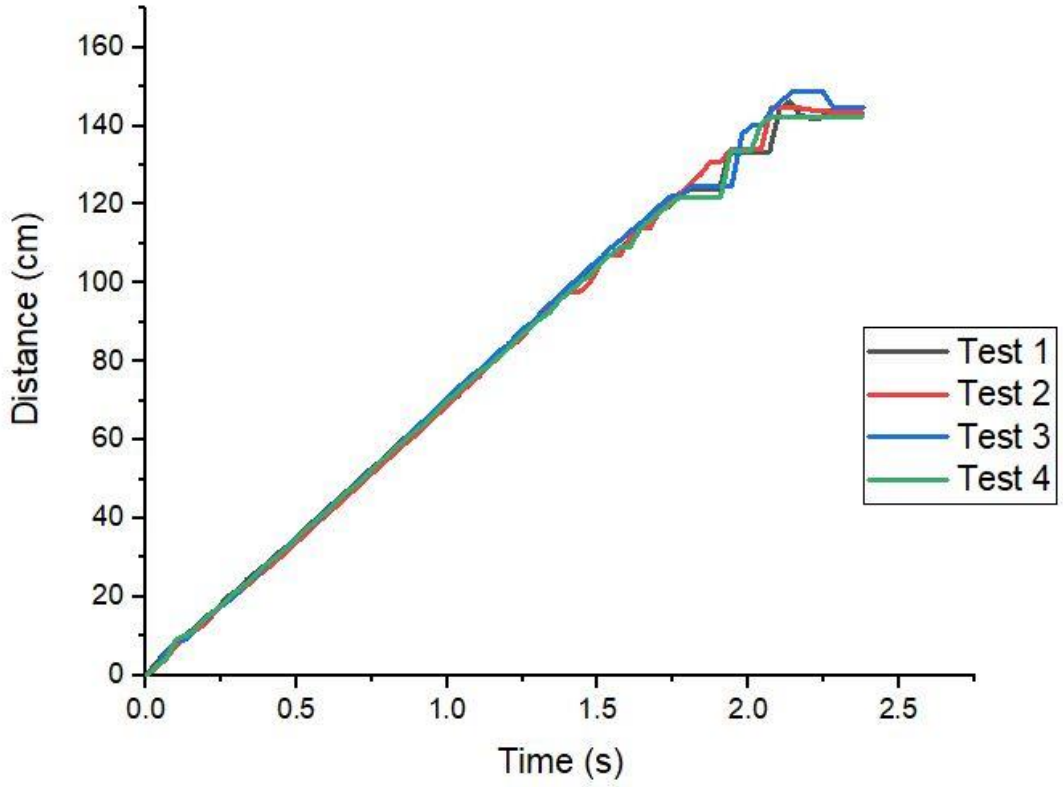


Figure 21: The graph shows the results of four tests to see if MATLAB could track the change in the robot's position over time. As we expected, MATLAB returned nearly identical results for all four tests and the results also agreed with what we observed ourselves.

We felt it was necessary to show that MATLAB could differentiate between forward and backward motion. To accomplish that aim, the robot was again constrained to traveling within a set distance. The forward and reverse operation was performed twice. In Figure 22, the graph displays a positive slope when the robot is traveling forward and a negative slope when the reverse occurs. Before changing directions, the robot was kept

stationary for approximately a second each time and this is evident when the slope is equal to zero.

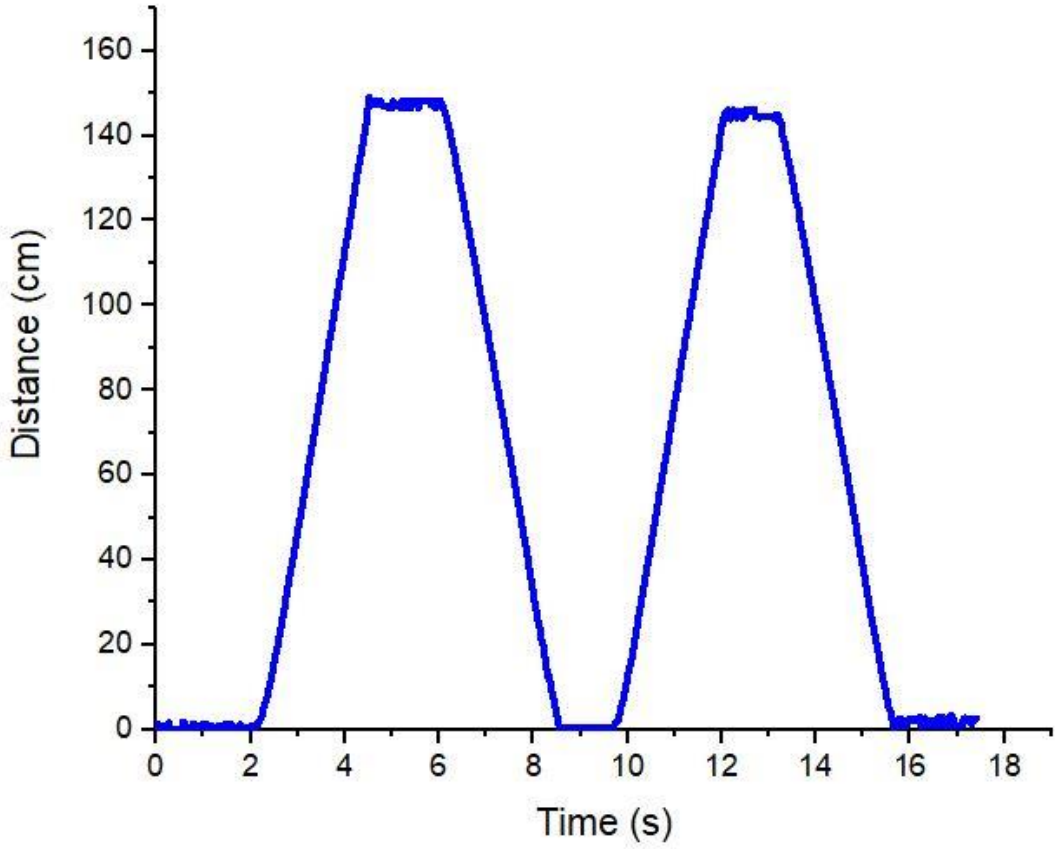


Figure 22: The robot was driven forward and in reverse to prove that MATLAB could reliably differentiate between both.

Unfortunately, each time MATLAB collects data and produces a graph displaying the displacement of the robot it does not start at zero. This happens because the MATLAB code tells the user the location of the red circle in a reference plane defined by the video's dimensions. Therefore, it is necessary to subtract the starting position value from all other values in order to determine displacement. Regardless, these tests should be sufficient to show MATLAB's suitability for use in experiments.

The original plan was to perform a series of tests and use the data we collected to calculate the cost of transport (COT) for the amphibious robot test bed. Afterwards we would compare how changes in certain parameters affected the robot's COT and swimming performance. One of the parameters we intended to change were the flexibility of the panels serving as actuators. Previous studies stated that increasing flexibility had a positive impact on efficiency while having a detrimental effect on thrust. We hoped that our experiments would verify these results. To accomplish that goal, three sets of actuators with identical dimensions but varying thickness had been fabricated.

The equation to calculate COT we planned to use was

$$COT = \frac{Energy}{mass \times distance}$$

where $Energy$ is the amount of power consumed by the robot. As explained earlier, MATLAB would be used to determine the distance traveled. The data MATLAB collected could also be used to find the speed at which the robot was moving.

Current and voltage sensors were placed among the electronics of the robot so we would know how much power the robot was using. The output of the voltage sensor was multiplied by five to get the result in volts. Before the information from the current sensor could be used, we added -2.5 to remove the offset then multiplied the data by 10. We would know the amount of power being consumed after multiplying the current and voltage readings.

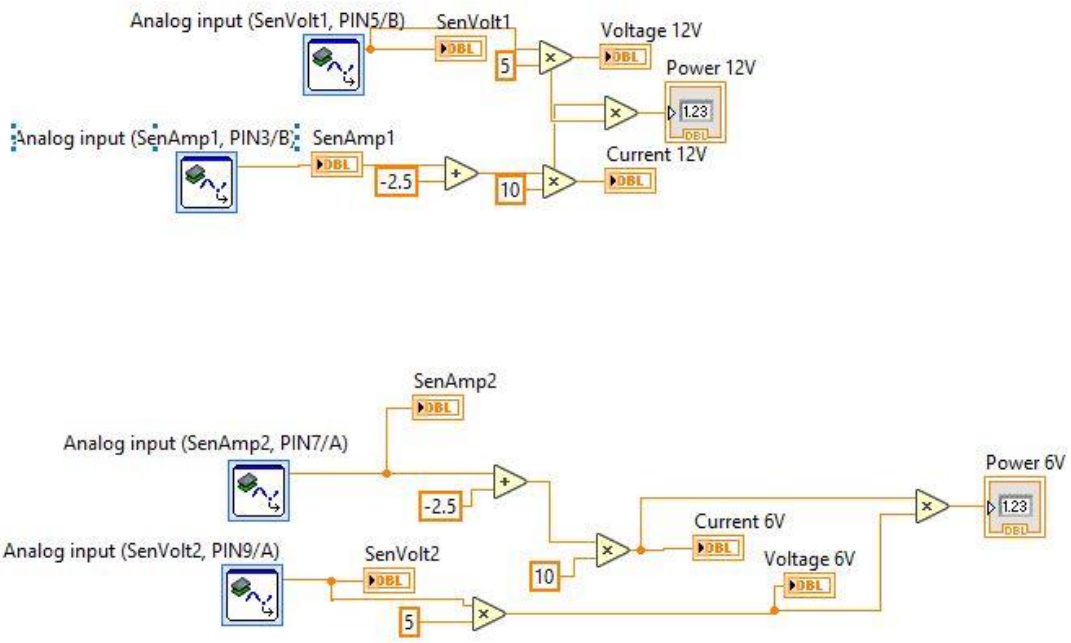


Figure 23: The readings from the current and power sensors were sent to the myRIO. In LabVIEW, these readings would be used to calculate the amount of power the robot was using.

We also desired to use actuators of different widths and lengths to observe how these changes would affect the robot. While we were still refining the robot's control program and overall design, we observed that when the actuators were too long the robot was not able to swim. Going further, we wanted to see if we could find a relationship between the width or length of the actuators and COT. This information could then be useful in making further improvements to this amphibious robot or inform design choices in other robots.

The final parameters we wished to vary were the amplitude, frequency, and balance of the motors. Previous studies stated that increasing amplitude or frequency would lead to an increase in the amount of thrust generated by the actuators. All else being equal, an

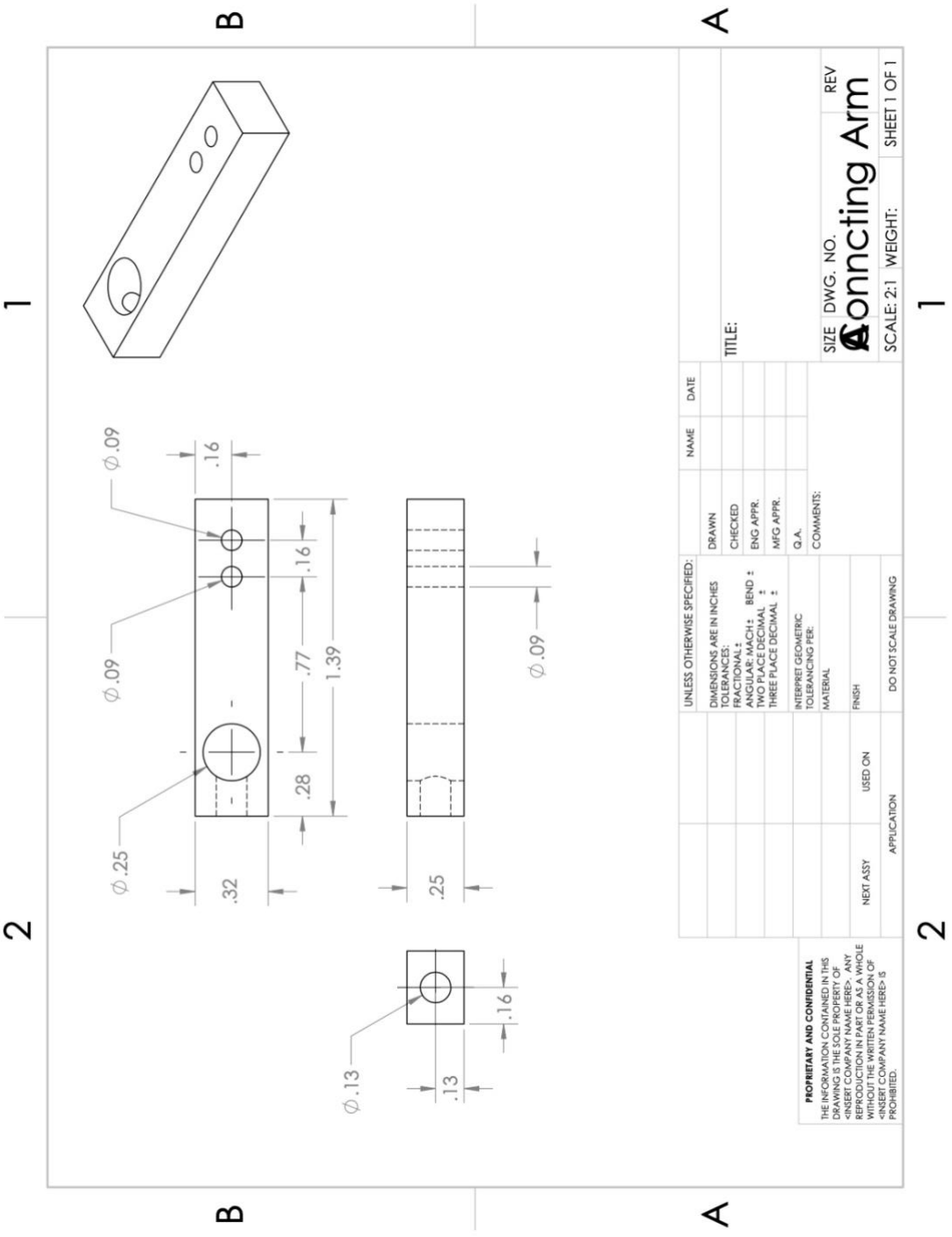
increase in thrust should lead to some increase in speed, which we could observe. Therefore, we would be able to verify the previous studies. While verifying that the robot could swim, we observed that it would not move when the actuators were positioned at a certain angle, but it would move once the balance of the motor was changed. These tests may prove more useful for improving our amphibious robot, but it is believed that knowing which angle for balance is more efficient would still be useful information.

5. CONCLUSION

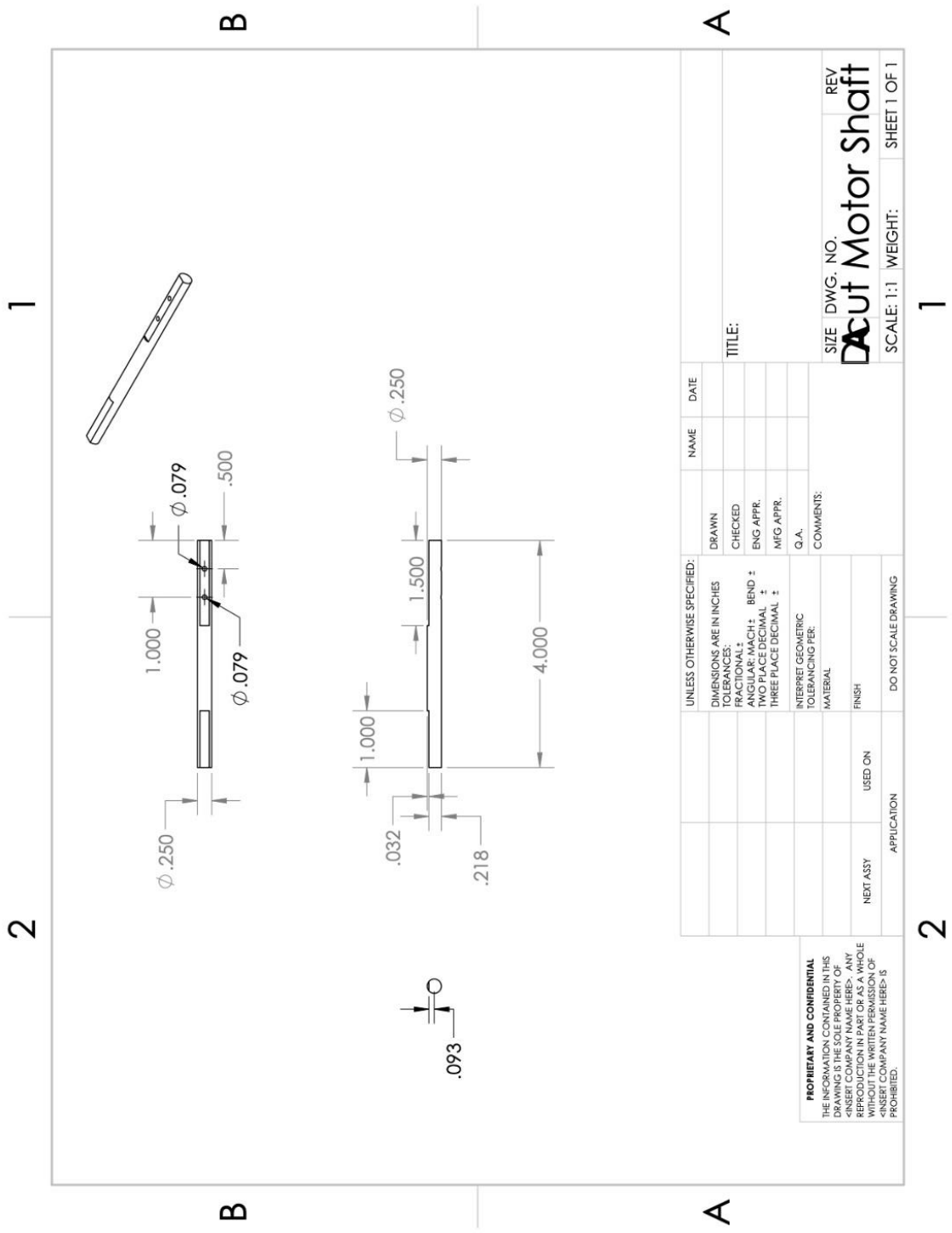
5.1 Future Work

The main objective of this project was to create a better version of our amphibious robot prototype, focusing specifically on the swimming ability of the robot. This goal was accomplished in its entirety. With that completed, it is believed that in the near future, we will be able to test the hypothesis that compliant actuators lead to higher locomotion efficiency. The robot was allowed to control its heading autonomously so it would travel along a straight path. If the robot travels in a straight line, the MATLAB program will be able to provide a better estimate of the robot's displacement. Voltage and current measurements during operation will be used in conjunction with displacement data to calculate the COT. After changing parameters such as the actuators' rigidity and shape over a series of tests, the COT for each difference in parameter will be compared. Additionally, work has already been started to create computer models of the robot for simulation purposes. These models can then be used to verify experimental results and provide insights on how the design can be further improved.

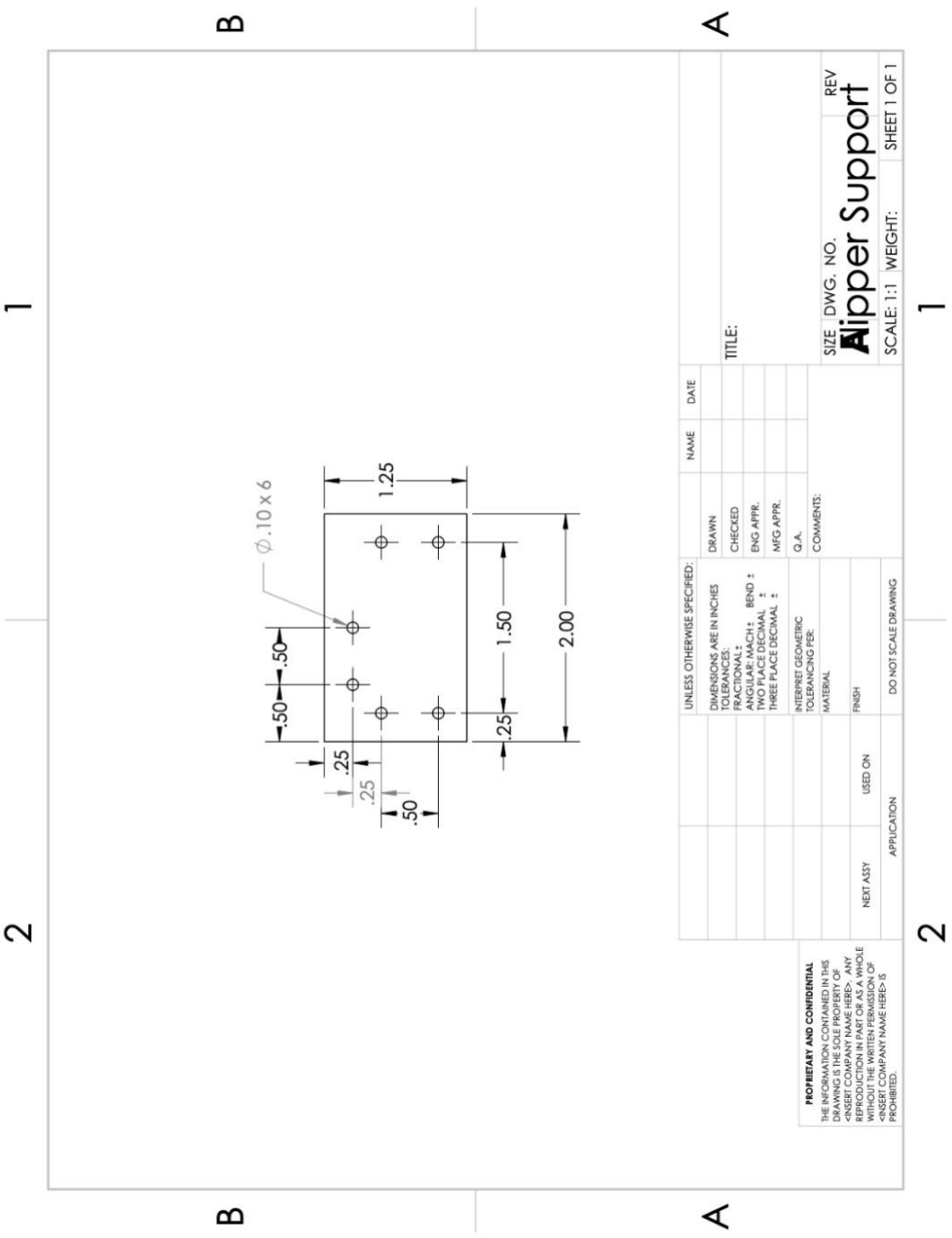
APPENDIX A: Supplementary Figures



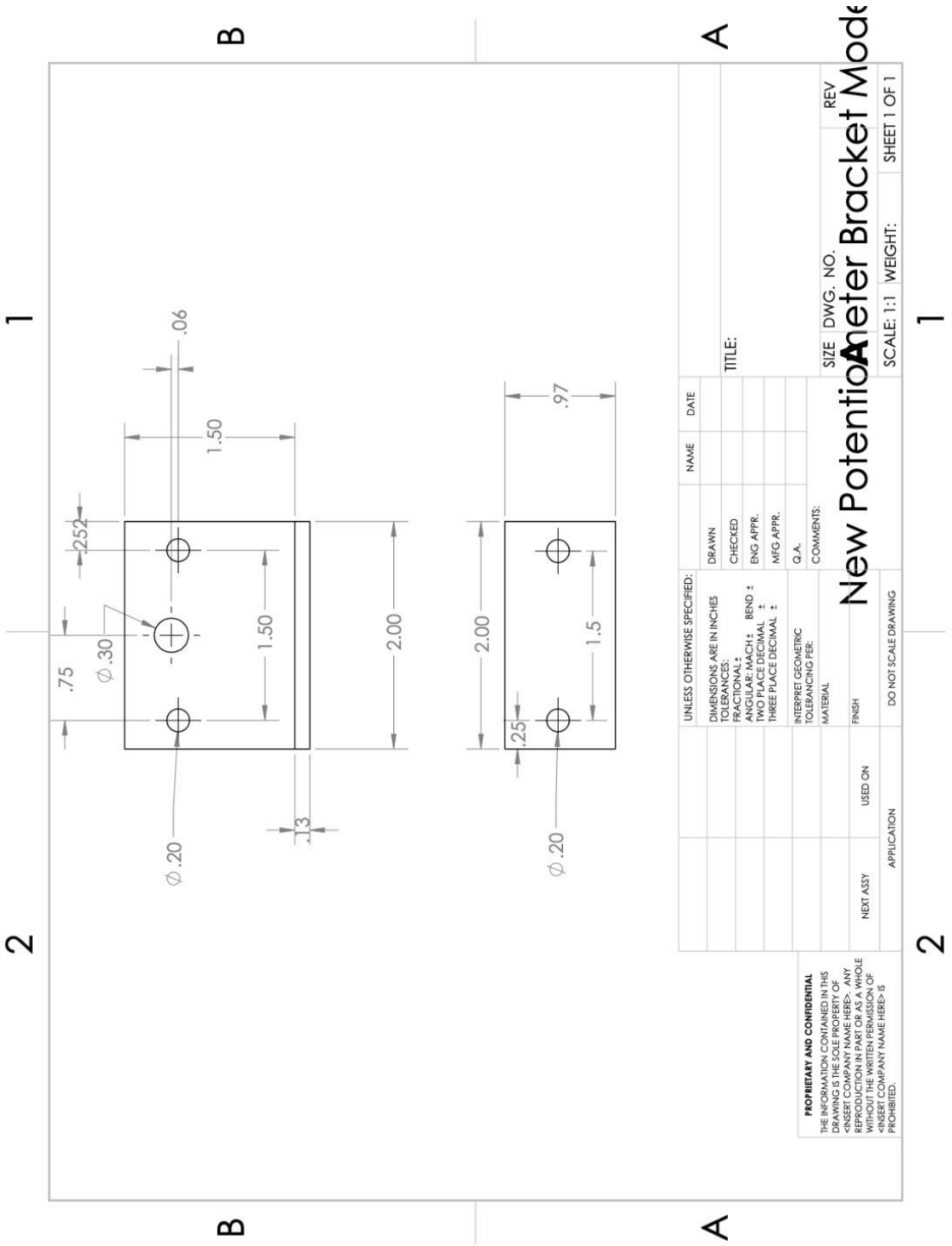
Supplementary Figure 1: The SolidWorks drawing of the connecting arm for the rudder subsystem. The rudder shaft passes through the largest of the three holes on top of the connecting arm and is rigidly attached via a screw at the back.



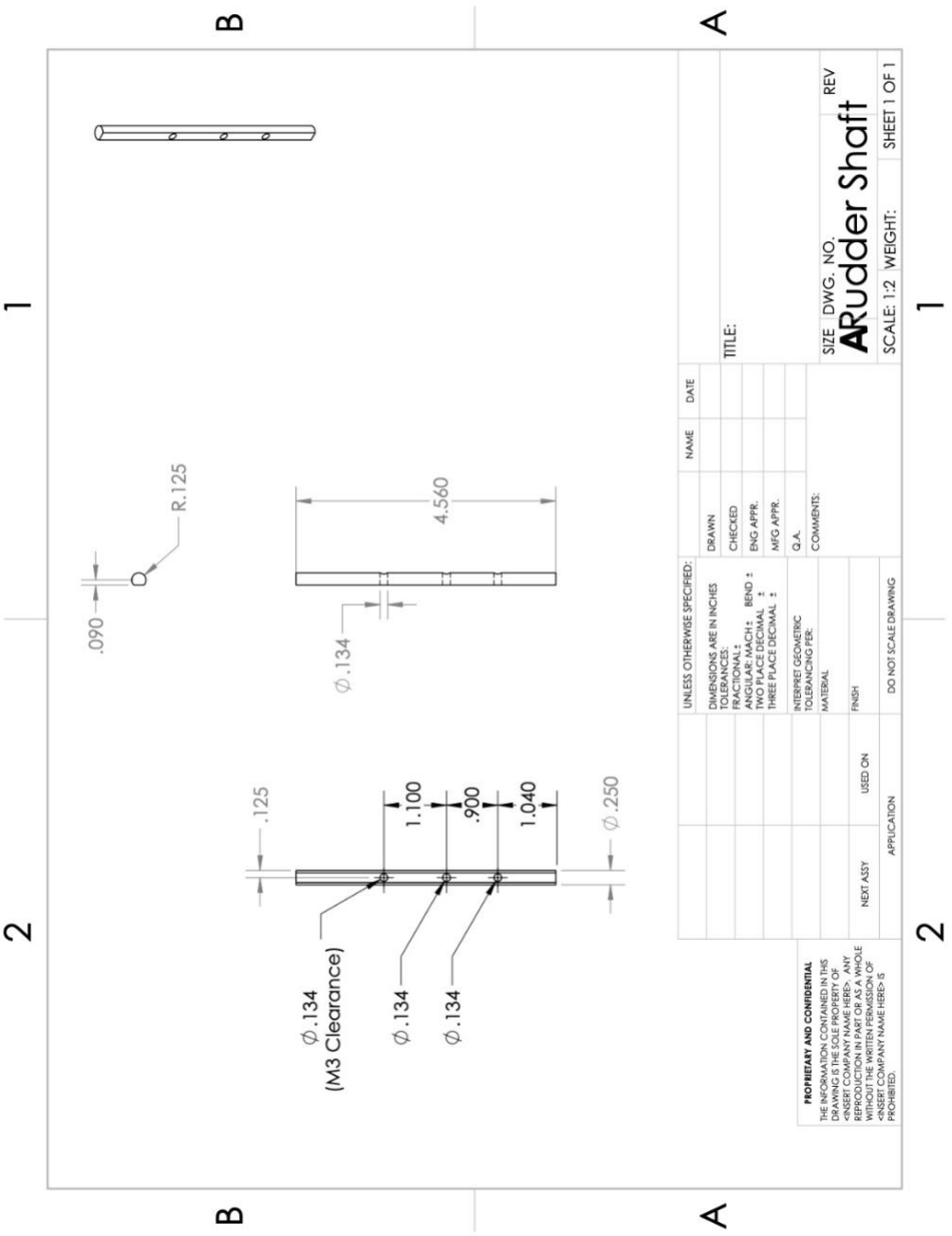
Supplementary Figure 2: One end of the motor shaft is attached to the motor via a coupling. The D-cut on this side was made so the shaft would fit snugly through the potentiometer. Since the potentiometer is what allows us to know the shaft's orientation it was important that there were no significant gaps. The longer D-cut was made to securely attach the actuator supports to this shaft.



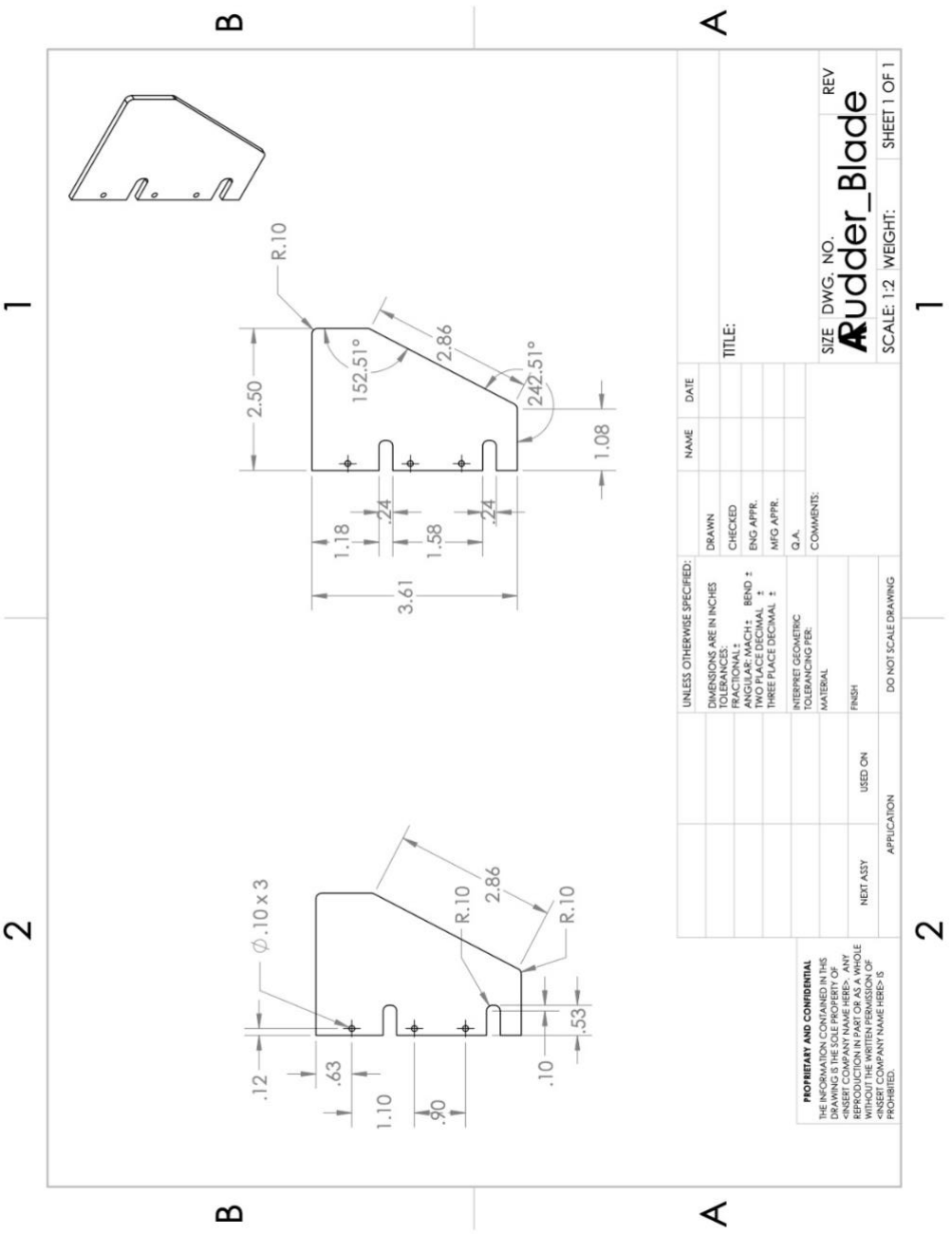
Supplementary Figure 3: The actuator support has two holes near the top edge for attaching to the motor shaft. The remianing four holes connect the support to the robot's actuators.



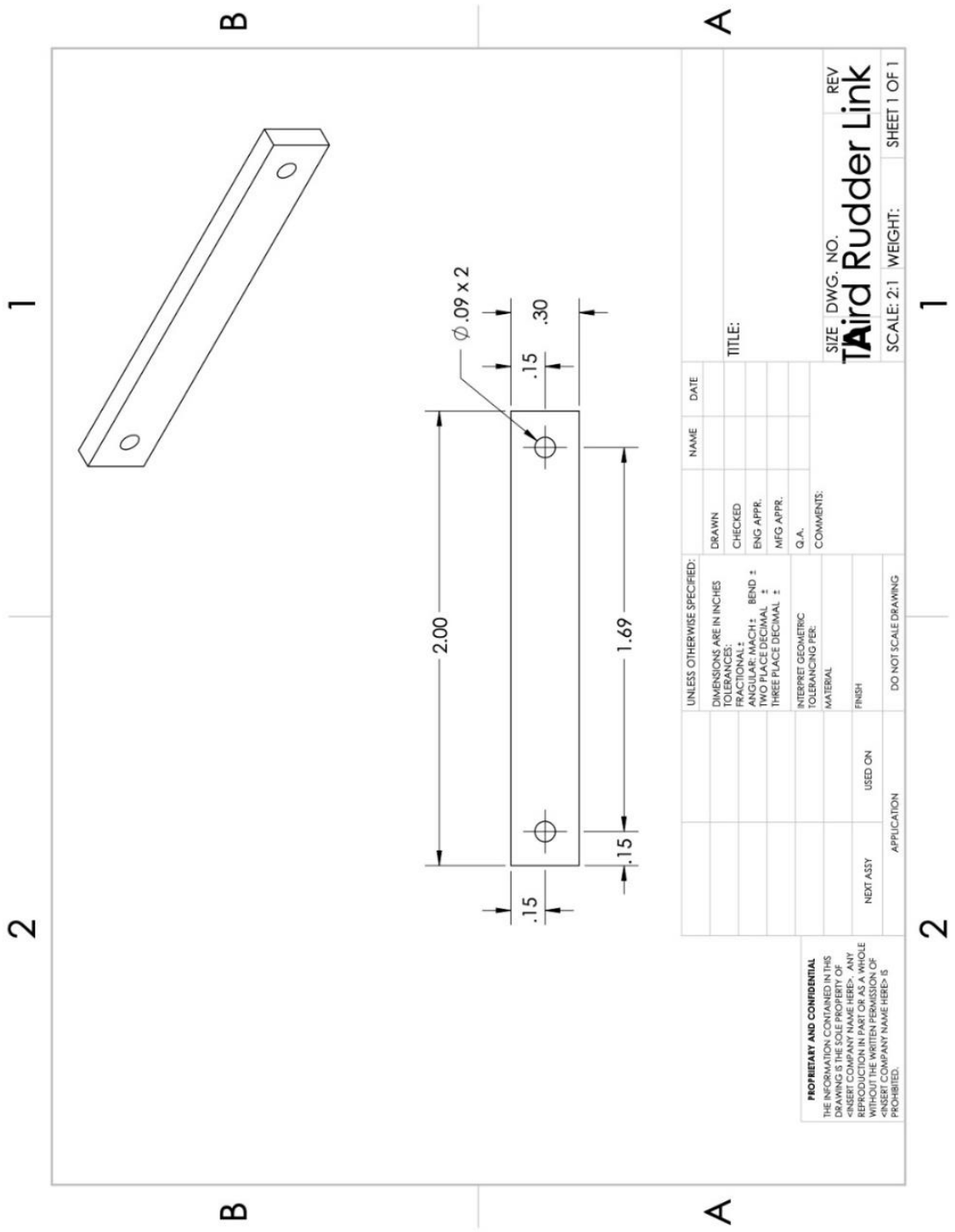
Supplementary Figure 4: The bracket that supports the potentiometer was fabricated using a corner bracket we ordered. There were two holes on the base to secure the bracket to the baseplates. On the face, the middle hole allowed the motor shaft to pass through while the holes on the side held the potentiometer in place.



Supplementary Figure 5: A flat surface was cut along the side of a metal rod so the rudder blade could be securely fastened to it via three screws. The height of the shaft is just slightly less than the height of the robot.



Supplementary Figure 6: This is the original design for the robot's rudder blade. A second version was made in which the length was increased so that the rudder blade would extend further from the robot's body. The semi-circular gaps are present so the rudder blade will not interfere with the ball bearings that support the rudder shaft.



Supplementary Figure 7: This is the third link in the rudder subsystem. This link connects the connecting arm with the horn of the servomotor.

APPENDIX B: MATLAB Code

1. The user selects which video they want to analyze. MATLAB returns such information as the number of frames, the frame rate, the duration of the video, and the dimensions of the video.

```
% Read and process videos
movobj=VideoReader('videoname.mp4');
nFrames = movobj.NumberofFrames; %returns number of frames in video
Duration = movobj.Duration; %returns duration of video in seconds
vidHeight = movobj.Height;
vidWidth = movobj.Width;
vidFPS = movobj.FrameRate; %returns rate in frames per second
```

2. Define or refine the parameters to ensure proper analysis of the video.

```
data=zeros(1,4);
cut_P1=[427 54]; %cut_P1 and cut_P2 are the corners of the rectangular
                 %area where analysis takes place
cut_P2=[965 1058];
threshold=0.1; %Lower threshold = Higher chance of detecting a circle,
               %but it also increases the chance false positives
Rmin=60; %minimum detectable radius of red circle in video
Rmax=75; %maximum detectable radius of red circle in video
```

3. Image tracking code for the position of the amphibious robot. In each frame, MATLAB identifies red circles that fit the previously defined parameters and records their position.

```
for i=1:nFrames
    i
    ref_frame= read(movobj,i);
      % creates an image from frame i of the video
    ref_red_frame=ref_frame(:,:,1);
      %creates an image that identifies all red objects.
    ref_gray_frame=rgb2gray(ref_frame);
      %converts ref_frame to a grayscale image
    ref_diff_frame=imsubtract(ref_red_frame,ref_gray_frame);
    %subtracts each element of ref_gray_frame from ref_red_frame


    figure
    imshow(ref_frame)
    figure
    imshow(ref_diff_frame)



ref_frame_cut=ref_diff_frame(cut_P1(2):cut_P2(2),cut_P1(1):cut_P2(1),:)
;
    %Creates an image from ref_diff_frame whose dimensions are defined %
    by cut_P1 and cut_P2.
```

```
    ref_img_bin=im2bw(ref_frame_cut,0.1);


    figure
    imshow(ref_img_bin)


    [centers,radii] = imfindcircles(ref_img_bin,[Rmin Rmax],...

'ObjectPolarity','bright','Sensitivity',0.95,'EdgeThreshold',...
        threshold);


    figure
    imshow(ref_img_bin);


     viscircles(centers, radii,'EdgeColor','b');


     [m,n]=size(centers);


     %In the following ideal situation, MATLAB detects just 1 circle
     if m==1
        data(i,1)=i;
        data(i,2)=centers(1,1); %x coordinate of center
        data(i,3)=centers(1,2); %y coordinate of center
        data(i,4)=radii(1);     %radius of circle
```

```
    %In this case, more than 1 circle is detected
    elseif m>1
       data(i,1)=i;

        for p=1:m
            distance_last_frame=((data(i-1,2)-centers(p,1)).^2+...
                (data(i-1,3)-centers(p,2)).^2).^0.5;

            if distance_last_frame<20
                data(i,2)=centers(p,1);
                data(i,3)=centers(p,2);
                data(i,4)=radii(p);
            end
        end

    %If no circles are found, the previous loop's data is used
    else
        data(i,1)=i;
        data(i,2)=data(i-1,2);
        data(i,3)=data(i-1,3);
        data(i,4)=data(i-1,4);
    end

end

plot(((data(:,3)).^2+(data(:,2)).^2).^0.5)
 %Plots the frame number vs distance traveled in pixels.
```

4. A ruler or another object of known length should be included in the video so the distance traveled by the amphibious robot can be converted from pixels to real-world units.

```
x1=461; %Enter the x coordinate for one end of the ruler here
y1=673; %y1 of ruler
x2=909; %x2 of ruler
y2=651; %y2 of ruler
length=((x1-x2)^2+(y1-y2)^2)^0.5;


ratio=length/10;
%Find the ratio between the ruler's length in pixels and
%its real-world units. For example, if the ruler is 10 cm long,
%divide by 10 to know how many cm are represented by a pixel


ratio_t=nFrames/Duration;
      %The Ratio between # of frames and time in seconds
```

5. Plot displacement with respect to time. MATLAB also stores the time and position data.

```
plot(data(:,1)/ratio_t,((data(:,3)).^2+(data(:,2)).^2).^0.5/ratio)
%Plot x=time vs y=difference in position
data1=data(:,1)/ratio_t %elapsed time
data2=((data(:,3)).^2+(data(:,2)).^2).^0.5/ratio %position
```

REFERENCES

1. Bayat, Behzad, Alessandro Crespi and Auke Ijspeert. "Envirobot: A Bio-Inspired Environmental Monitoring Platform ." *IEEE* (2016): 381-386.
2. Boxerbaum, Alexander S., et al. "Design of an Autonomous Amphibious Robot for Surf Zone Operation: Part I Mechanical Design for Multi-Mode Mobility." *International Conference on Advanced Intelligent Mechantronics*. Monterey, California: IEEE, 2005. 1459-1464.
3. Buren, Tyler Van, Daniel Floryan and Alexander J. Smits. "Bio-inspired underwater propulsors." 2018.
4. Clark, R.P and A. J. Smits. "Thrust production and wake structure of a batoid-inspired oscillating fin." *Journal of Fluid Mechanics* 562 (2006): 415-429.
5. Dudek, Gregory, et al. "AQUA: An Amphibious Autonomous Robot." *Computer* 40.1 (2007): 46-53.
6. Jia, Xinghua, et al. "Biological Undulation Inspired Swimming Robot." *International Conference on Robotics and Automation*. Singapore: IEEE, 2017. 4795-4800.
7. Kim, Hyung-Jung, Sung-Hyuk Song and Sung-Hoon Ahn. "A turtle-like swimming robot using a smart soft composite (SSC) structure." *Smart Materials and Structures* 22.1 (2012).
8. Klein, Matthew A., et al. "SeaDog A Rugged Mobile Robot for Surf-Zone Applications." *International Conference on Biomedical Robotics and Biomechatronics*. Rome, Italy: IEEE, 2012. 1335-1340.
9. Lauder, George V., et al. "Fish biorobotics: kinematics and hydrodynamics of self propulsion ." *The Journal of Experimental Biology* (2007): 2767-2780.
10. Liang, Xu, et al. "The AmphiHex: a Novel Amphibious Robot with Transformable Leg-flipper Composite Propulsion Mechanism." *International Conference on Intelligent Robots and Systems*. Vilamoura, Algarve, Portugal: IEEE, 2012. 3667-3672.
11. Liu, Huikang, et al. "Platform Design for a Natatores-like Amphibious robot ." *International Conference on Mechantronics and Automation*. Changchun, China: IEEE, 2018. 1627-1632.
12. Quinn, Daniel B., George V. Lauder and Alexander J. Smits. "Scaling the propulsive performance of heaving flexible panels." *Journal of Fluid Mechanics* 738 (2014): 250-267.
13. Song, Sung-Hyuk, et al. "Turtle mimetic soft robot with two swimming gaits." *Bioinspiration & Biomimetics* 11.3 (2016).

14. Sousa, Paulo J.S.A. Ferreira de and James J. Allen. "Thrust efficiency of harmonically oscillating flexible flat plates." *Journal of Fluid Mechanics* 674 (2011): 43-66.

15. Tytell, Eric D., Leftwich, Megan C., et al. "Role of body stiffness in undulatory swimming: Insights from robotic and computational models." *Physical Review Fluids* 7.1 (2016).

16. Wang, Lillian. "A Jellyfish-based Aquatic Locomotor With Tunable Gaits." MS Thesis. 2017.

17. Xu, Weinan and David H. Gracias. "Soft Three-Dimensional Robots with Hard Two-Dimensional Materials." *ACS Nano* 13.5 (2019): 4883-4892.

18. Yang, Qinghai, et al. "Preliminary Development of a Biomimetic Amphibious Robot Capable of Multi-Mode Motion." *International Conference on Robotics and Biomimetics*. Sanya, China: IEEE, 2007. 769-774.

19. Yu, Junzhi, et al. "On a Bio-inspired Amphibious Robot Capable of Multimodal Motion." *Transactions on Mechatronics* 17.5 (2012): 847-856.

20. Zhong, Bin, et al. "Locomotion Performance of the Amphibious Robot on Various Terrains and Underwater with Flexible Flipper Legs ." *Journal of Bionic Engineering* (2016): 525-536.

21. Zhu, Jing, et al. "Initial development of an amphibious robot with flexible straight flipper-legs." *Conference on Real-time Computing and Robotics*. Kandima, Maldives: IEEE, 2018. 417-420.