Exploring and Developing Tools for Autonomous Surface Vehicles

***Abstract*—Autonomous Surface Vehicles (ASVs) that operate in field experiments require tools to help understand, manage, and develop the hardware and software. Building these resources requires expertise in many areas of engineering including: electrical engineering, computer science, mechanical engineering, physics, and mathematics. This project explores a wide range of software and hardware tools utilized in deploying ASVs. The software tools explored aim to design different behaviors for both miniature surface vehicles (mASVs) and a larger Unmanned surface vehicle (USV). Behaviors are any set of instructions given to a robot for it to perform. Two behaviors are studied: one for the mASV to interact with objects, and one to allow the larger USV to explore real world environments. Simulation serves as a visualization tool and a test environment where behaviors can be examined and analyzed before the experiment is conducted on the actual robots. Physical experiments are then performed to verify the performance of behaviors on hardware. In order to run experiments, it was necessary to develop control and hardware for the robots, including designing and fabricating a printed circuit board (PCB).**

I. Introduction

In the last decade, robots have increasingly become more sophisticated and are no longer limited to mundane tasks. One tool that has become increasingly investigated in the field of robotics is autonomy. When given a particular task, various obstacles or factors may impede a robot from successfully completing its job. This is where autonomy would prove useful. Autonomous robots are robots that can change their behavior in response to unanticipated events during operation. The idea of combining robotics with autonomy creates opportunities to improve upon the precision and efficiency robots have in carrying out the tasks they are instructed to perform. For instance, autonomous boats could be instructed to track algae blooms [1], and monitor oil spills or survey uncharted regions of the ocean [2]. Autonomous robots come in many different forms including: space, air, ground, underwater, and surface vehicles.

This project focuses on exploring autonomous surface vehicles and developing the sensing and control mechanisms needed to allow robots to move and operate in the real world. Robots can perform extremely well in a structured and controlled environments, like a factory or laboratory space. However, robots are needed in various real-world environments which are unstructured, making robotics all the more challenging. Autonomous behaviors are explored in two types of Autonomous Surface Vehicles (ASVs): the Clearpath Heron, and the ScalAR Lab miniature Autonomous Surface Vehicles, mASVs.

There are three main steps that allow a robot to perform autonomous behaviors. These steps are: sensing, reasoning, and acting. Sensors are what allow for these robots to gain information about their environment. After the robot has sensed different properties of its environment, it then reasons or develops a plan to perform a behavior. After reasoning, the robot then performs some action to achieve the desired behavior. Controllers are instruments used in the reasoning process to plan a robots movements. One controller widely used in the field of robotics is a proportional-integral-derivative (PID) controller. PID controllers can also be used to regulate behaviors in a robot, where a behavior is a single or series of action(s) performed by a robot [3]. Examples of behaviors include having a team of robots swarm together [4], using a USV to navigate through an obstacle course [5], or instructing a USV to push an object to a particular location [6]; there are numerous unique behaviors that can be achieved for robots using controllers. This paper investigates how to develop the behavior that allows for autonomous surface vehicles to push an object to a particular goal. ASVs that can push an object can have many major applications, such as creating robots to support construction or build structures on their own in a remote environment [6], [7].

Pushing objects is a basic behavior that gives robots the ability to manipulate their environments to be in a more desirable manner. Enabling robots with the ability to push objects would allow them to manipulate the spaces around them, thus significantly increasing the types of tasks robots can be instructed to perform. In this project, the surface vehicles use the force produced by their motors to move an object to a target location. This project begins with one mASV that receives instructions on how to apply thrust to its motors. The mASV will then use these instructions to push a Styrofoam ball to a specified target location. After which, more robots were to be introduced to identify if objects can be manipulated more efficiently with the aid of multiple robots.

The contributions accomplished during the course of the SUNFEST program are:

1) A discussion of new tools learned throughout the summer
2) Development of a PCB for RS232 to USB
3) Design and evaluation of a USV behavior
4) Field experiments with a fully autonomous USV in the river

II. Background

Experimental robotics requires hardware and software to work together to enable a robot to move and sense in the world. This project focuses on learning about the software and hardware necessary to run experiments with robotic boats.

A. Robot Simulation

One of the software related tasks this project covers is writing a python point robot simulator which uses a second order differential equation model of a robot's state [8]. This was extended to consider different control methods of multiple robots, for example potential fields [4] and PID controllers [3], to generate different robot behaviors. In addition, this work involves writing python code to control the miniature Autonomous Surface Vehicles (mASVs) for experimental testing. Finally, this project covered the foundations of the Robot Operating System (ROS) and applying those skills to develop nodes which can control a larger Unmanned Surface Vehicle (USV).

B. Robot Control

PID controllers utilize a closed loop feedback mechanism that allows them to generate output commands to a robot for more precise operations [3]. Utilizing the information from the sensors, controllers can adequately make adjustments to the robot to achieve the desired behavior. This would be very useful for instance if a surface vehicle needed to move to a specific location. The vehicle could initially be on track to get to the desired destination, however during this process, the vehicle may veer off course as a result of drag or some other impediment. A PID controller allows the surface vehicle to automatically adjust itself using the data it obtains from its different sensors. In this case the vehicle could use it's GPS sensor to relay information about the vehicle's current position to a controller. The controller could then output adjusted motor commands to correct the vehicle's course. PID controllers are not only useful for keeping a surface vehicle on course, but have many other practical applications such as controlling temperature, pressure, angular speed and many other properties.

C. Robot Hardware

This project also involved building electronic components to improve the wiring on the Clearpath Heron USV. This consisted of designing, testing, and installing a printed circuit board which allows more sensors to be connected to the Clearpath Heron USV [9]. Through this process, critical information about sensor functionality and application was acquired to identify how robots can utilize sensors to improve their behaviors.

III. Methodology

A. Robotics Tools

Autonomous Surface Vehicles depend on many different systems to ensure they run properly. In this project, one of the primary software-related tools utilized was a framework called Robot Operating System (ROS) [10]. ROS is an open source software framework that is used to design software for robots. This project utilized ROS to handle message-passing from the robot to the programmers algorithm. ROS is important for this project because it allows access to read in different properties of the robot as well as send instructions to the robot to execute. ROS uses "topics" to divide up different properties of a robot. A ROS topic acts as a channel through which information can be read from or sent through. For example, the command "rostopic echo /position" tells ROS to get the information about the robots current position, while the command "rostopic pub position [given location]" tells ROS to publish a command to the robot to go to the given location. ROS plays a key role in this project by setting up the framework necessary for communication to the autonomous surface vehicles. ROS is supported for linux based systems, however this project is conducted primarily using a virtual machine.

Robotics is a highly collaborative field, and as a result, version control is a integral part of this project. Github is the designated version control software used in this project. It is used to store code and access the necessary repositories to run the experiments on the autonomous surface vehicles.

B. Design and Production of a PCB

This project covers a hardware related tasks, which includes designing a printed circuit board (PCB) for the Heron. The purpose of the PCB is to allow for multiple peripheral devices or sensors to be connected to the Heron. The Heron uses RS232 signals, which are a widely used communication protocol. However, most modern sensor devices now use a Universal Serial Bus (USB) communication protocol, as it is a more simple and efficient form of communication [9]. The problem is that the Heron and the sensors communicate using different protocols. The solution is to develop a PCB to convert RS232 signals to USB, extending the number of additional sensors that can be added to the Heron. The PCB design used Kicad, an open source software for electronic designs [11]. To begin fabricating the RS232 to USB converter, a schematic is created. The schematic consisted of all the electronic components necessary to build the PCB. A list of components can be found in the appendix.

After designing the schematic, a footprint of the PCB is created. During the footprint process, copper tracks and copper fills are placed down onto the board. Copper tracks act as the wiring for each component, while copper fills are large areas of copper that allow for easier tracing for similarly traced components. Once all the copper traces and fills are placed, a check is done to ensure all the components are connected correctly. Examples of these diagrams can be found in Figure 1.

Once everything is properly connected, the footprint for the PCB is shipped to a manufacture and the components are ordered. The PCB is soldered together following the outline in the footprint.

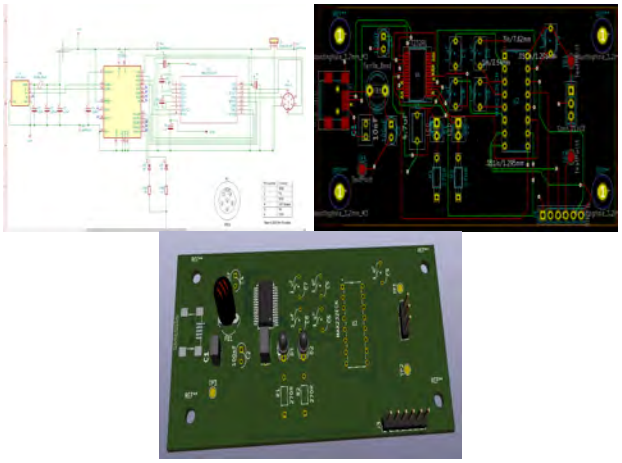

Fig. 1. Top left is schematic of PCB, top right is the footprint, bottom is digital 3D view of PCB

C. Unmanned Surface vehicle Behavior Design

1) Simulation Framework: A simulation is developed in Python and utilizes two libraries, namely PyGame for visualization and Pymunk for physics handling. The simulation consists of three main elements: an object, a boat object, and the target position. The generic object should initially be stationary and able to respond to collisions, the boat class's velocity and heading should be controllable, and the target position should be a stationary point in the simulation environment. In the simulation, a pushing behavior is defined for the boat object. This behavior is just a function that orients and moves the boat object in a manner that gets the generic object closer to the target destination with each time step. The purpose of this simulation is to aid in testing the behaviors before they are implemented on a miniature surface vehicle.

D. Pushing an Object

There are three steps to perform a pushing behavior, Figure 2. An overview of these steps are as follows:

1) Find the vector from the center of the object to the target position. There are many different ways to find this vector, but the most straight forward method is to subtract the target position from the objects position. The outcome of this operation will result in the desired vector, pointing from the center of the object to the target position. This vector is useful because it provides information about the magnitude and direction the object needs to move to reach the target position.
2) After identifying the vector, the next thing to find is the starting position of our boat object. To find this position, take the vector from step one, normalize it and flip it in the opposite direction (this is accomplished by multiplying it by -1) to ensure that the starting position is behind the object to be pushed. The vector can then be scaled using the dimensions of the boat object to prevent the boat from colliding with the object. After finding the starting position, orient the boat towards this position and move it to the start.
3) After the boat has successfully made it to the starting point, we will need to utilize the original vector from step one. The goal is to have the boat push the object directly along the vector from the object to the target position. The boat's position is already aligned with this vector, so all that is left to do is ensure that the orientation of the boat is facing the object and activate the motors to provide the boat with acceleration to push the object towards the target position.

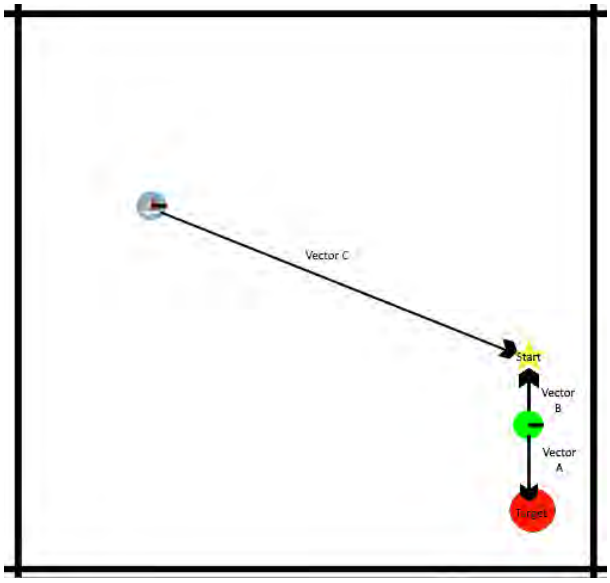

Fig. 2. Illustration of vectors used for pushing behavior

1) Surveying Behaviors: A different high fidelity simulation framework, ROS Gazebo, is used to design surveying behaviors for a larger ASV. The specific surveying pattern is to traverse a desired area along the Schuylkill River in a lawn mower pattern. This section discusses the function created to generate the waypoints in a lawn mower pattern along the river.

The waypoint generator takes in as argument the (x, y) coordinates of a four-sided boundary. The function assumes that the given boundary is a convex quadrilateral and returns a list of waypoints within this quadrilateral in the shape of a lawnmower pattern. The function allows the user to specify how many legs of the lawn mower pattern are desired, as well as, the direction to draw the pattern in, vertically or horizontally.

To begin, it is important to distinguish between the attributes of a vertical lawn mower pattern and a horizontal one. A vertical pattern would alternate from top to bottom (or vice versa) and would have shorter horizontal segments. A horizontal pattern would alternate from left to right (or vice versa) and would have shorter vertical segments. This paper will mainly discuss the algorithm used to generate the vertical

lawn mower pattern, but the horizontal pattern can be created using a similar technique.

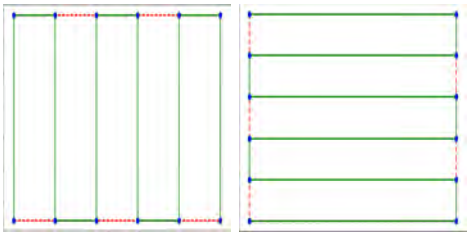

Fig. 3. Example of vertical and horizontal patterns. Left vertical, Right horizontal

For any vertical lawn mower pattern being drawn on a quadrilateral course, the waypoints will lie on two vectors: vector A and vector B. These vectors correspond to two opposite sides of the given course. The function assumes that the vector A is pointing from coordinate 1 to coordinate 2 and the vector B is pointing from coordinate 4 to coordinate 3. To determine how many points each vector should have, the function utilizes the specified number of lawn mower patterns the user entered. After the first lawn mower pattern, each pattern needs 2 additional points to be considered a complete pattern. Thus, the equation for the total number of points needed to achieve the desired amount of patterns is as follows:

$$T = 2 + (2 * P) \tag{1}$$

where T is the total number of waypoints, and P is the number of desired patterns.

To find the number of waypoints on each vector, the total number of waypoints (which will be an even integer) is divided in half. This ensures both vector A and B will also have an equal number of points on both sides. The function will then divide the length of each vector by the number of points on each respective side. This will tell the function the length of each sub-segment on a given vector. This length will be used to ensure that each point along a given vector will be equally distanced away from other points along that same vector (It is important to note that the sub-segment length should be calculated twice, once for vector A and another for the vector B). The function will start at coordinate 1 and generate points in the direction of coordinate 2 that are all equal distanced away from each other. These points will then be stored within an array, and in the end, the function will have a array of points along vector A. The same process should be repeated for the vector B, which is the vector pointing from coordinate 4 to 3. At the conclusion of this process, there will be two separate arrays that contain points that lie along two opposite sides of the given course.

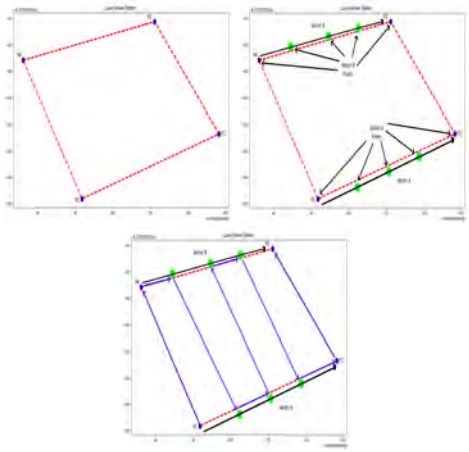

Fig. 4. Illustration of how lawn mower pattern is created

Finally, a third array is created which will serve as an ordered array of all the waypoints. This array is populated using the arrays that contain points along vector A and vector B. As seen in figure 4, for a vertical pattern, the function uses one point from the vector A, then one from vector B, before alternating to use one point from vector B, then one from vector A. The function arranges the third array in this manner by alternating between which array it pulls from first. For example if array A is the array of points along vector A, and array B is the array of points along vector B, then if the function first pulls from the array A before array B, then the next time it will pull from array B before array A.

At the end of this process, the function will return an array of ordered waypoints that follow a lawn mower pattern constrained within the specified boundary. Examples of the lawn mower function can be found in Figure 7

IV. Experimental Setup

A. Miniature Autonomous Surface Vehicles

Miniature autonomous surface vehicles (mASVs) are unmanned mobile robots that utilize differential drive to move around an environment. These mobile robots were developed in ScalAR lab and are placed in a $\approx 3x4x2$ meters tank. The mASVs consist of two motors, an arduino, a rubber lid, and a hull to house all the components. The arduino is used to directly control the mASV's motors. The arduino outputs commands to the motors, telling each one how fast it should spin. The turning of the motors generate thrust and allow distinct movements to be performed by the mASVs. In addition, the arduino can receive remote commands from an Digi XBee. An Xbee is a device that can remotely communicate with an arduino via WiFi. The purpose of this setup is to connect the Xbee to a computer which will then be able to send instructions to the arduino on how to move the mASVs motors.

A mASV's movement is determined as follows:

- When the left motor spins at a faster rate than the right motor then the mASV will turn right.

- Similarly, when the right motor spins faster than the left, the mASV will turn left.
- When both motors spin at the same rate and in the same direction, the mASV will move forward.

(Note: While there are many more movements that can be achieved, for the purpose of the project these are the three focused on)

While the mASVs have the ability to move freely in a given space, as stated earlier, in water, there are many impediments to a surface vehicles movements. To counteract this, a PID controller is implemented. For the PID controller to work properly, it needs information about the mASVs geometry. To obtain this information a motion capture system is mounted above the tank to track the orientation and position of the mASVs. The motion capture system used in this project is Optitrack, created by NaturalPoint, Inc, and the software used to interact with the capture system is Motive. The Motive software streams data about a rigid body - in this case the mASVs, which is then published in a ROS node. After properly setting up the Optitrack and the Motive software, the full state of the mASV is used by the PID controller. This information can then be used to derive the angular and linear velocities of a mASV. In Python, a boat class is implemented to track information about a single mASV. The boat class has four main methods:

1) "update_boat_state" - Updates the state of the boat every time data is received from the Optitrack
2) "set_desired_speeds" - Sets the desired velocity and heading of the mASV.
3) "boat_pid" - PID controller that outputs motor commands to obtain desired position and heading.
4) "send_command" - This method is responsible for sending motor commands to the Xbee to send to the remote Arduino on the mASV.

The PID controller will ensure that the desired velocity and heading is achieved by the mASV. With this setup a proper closed loop control is achieved; if the boat veers off course, it will now be able to correct its self, and properly complete the instructions it was given. Using this set up, the pushing behavior can now be implemented and executed by the mASV.

B. Heron

The Heron is ScalAR lab's larger USV. It was developed by Clearpath Robotics and is equipped with various sensors that can be used for a variety of different experiments. Some sensors on board the Heron include a GPS, IMU, camera, and depth sensor. The Heron is a pontoon boat, with water jet propulsion that allows it to move around in an aquatic environment. It can be controlled manually using its remote controller or autonomously by sending it commands. To achieve autonomy, the Heron uses a ROS based autonomy framework. For example, a user can obtain information about the depth sensor, position, or velocity of the boat. In addition to reading from these ROS topics, the user can write commands to these topics as well, allowing users to manipulate different properties within the Heron.

V. Results

A. Miniature Autonomous Surface Vehicles

The mASV experiments were not finished due to time constraints. However, the simulation results show that the procedure descried earlier for pushing objects was effective. As seen in Figure 5 the distance between the object and the goal decrees over time as the robot pushes the object to the goal.

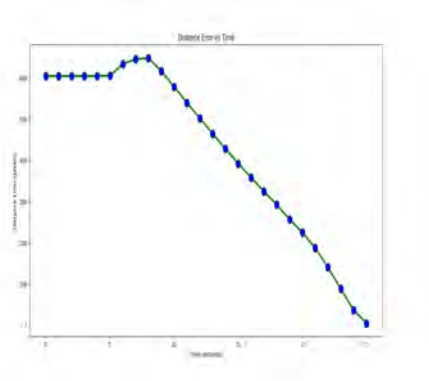

Fig. 5. Distance Error vs Time

B. Heron Field experiments

The Heron had many moving components. This work focused on the development of a PCB and generating waypoints.

1) PCB Final Result: A limited amount of sensors could be connected to the Heron at a given time, since it used a RS232 protocol to transmit data. However this issue was resolved after creating a printed circuit board to convert RS232 signals to USB. Figure 6 shows the completed PCB used on the Heron.

Fig. 6. Image of assembled RS232 to USB Converter

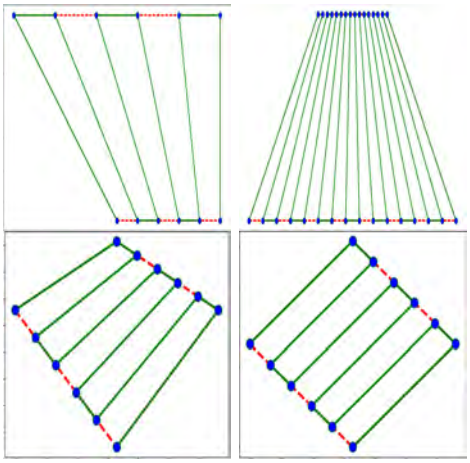

Fig. 7. Example of other lawn mower patterns on quadrilateral courses

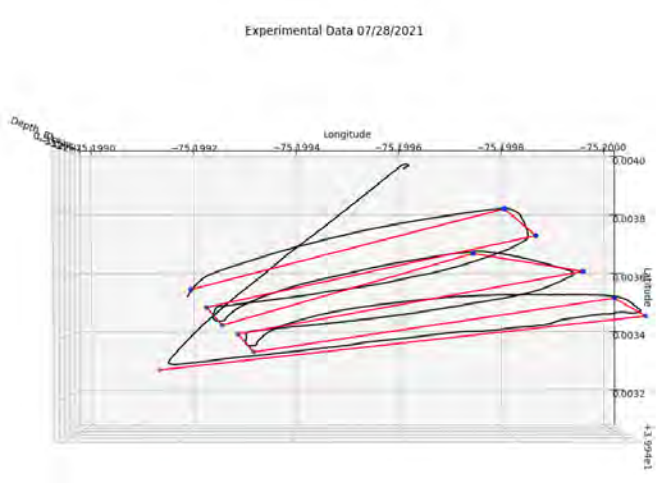

Fig. 8. Red line is the point to point path between supplied waypoints. The black line is the trajectory the boat took.

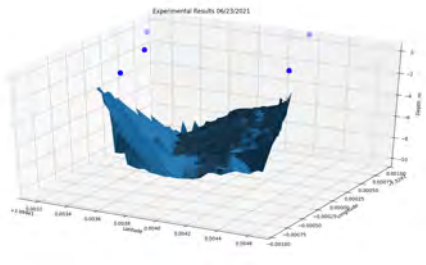

Fig. 9. 3D reconstruction of Schuylkill riverbed using depth sensor

2) Heron Surveying an Environment: Overall, many improvement have been made to the Heron to create a powerful tool for research. The Heron is now equipped with a fully functional depth sensor, IMU, GPS, camera, and other sensors that allow it to collect useful data about its environment. More sensors can now be connected to collect more information about the boat's environment. In addition, the boat can now perform autonomously, meaning it can carry out much more complex tasks and more properly adjust to unexpected events in its environment. Following waypoints while mapping the riverbed of the Schuylkill river was just one example of the Heron's use case. This fully function autonomous surface vehicle can collect a wide array of sensor data which can be used to: map the river bed, understand water currents, capture salt fronts or other time varying properties of the river.

The first results demonstrated that the robot was able to collect data from ROS topics. More specifically, the depth sensor ROS topic needed to be examined for accuracy. To do this, the boat was placed in the Schuylkill river and driven manually around the river. After driving the Heron in an open loop square, a 3D plot of the depth sensor data was generated using matlab. Using the depth sensor ROS topic, the plot accurately reconstructed the riverbed, Figure 9

The next task was to get the Heron moving autonomously. To achieve this, another PID controller was implemented to maintain the desired behavior. The PID controller, similar to the mASVs, controlled the position and heading of the Heron. In contrast to the mASVs, the Heron contains an internal GPS to acquire information about its position, and a IMU that outputs information about the boats orientation. To achieve the best performance in the PID controller, the sensors were tested for accuracy. Finally, to test autonomy, the lawnmower waypoint generator was used. The function generated a list of coordinates for the Heron to autonomously follow. In the end, the boat was successfully able to follow the lawnmower-patterned coordinates, while utilizing its depth sensor to map out the Schuylkill river, Figure 8.

VI. Discussion

One of the important questions that arose throughout this project was regarding the fidelity of a simulation. Simulations are important as they provide visualizations and a general understanding of how a robot will perform if given the same commands in the real world. While simulations play an important role in robotics, an important question to address is how realistic does a simulation need to be before considered "good" enough. A simulation's fidelity has to do with how well the simulation imitates what will happen in the real world. When designing a simulation for the mASVs, various properties could be included such as modeling the drag force of water, or realistic collisions. While adding all these properties that are found in the real world add more fidelity and complexity to the simulation, it comes at the price of time. Mimicking the real world within a simulation can quickly become time consuming, and in most cases it is impossible to model every phenomenon possible in an experiment. Instead a threshold should be set in place to asses whether or not a simulation is "accurate" enough to give a reasonable conception of what will happen in an experiment. In this project, the simulation only modeled what was vital to the experiment. For the mASVs pushing behavior, collision between the boat and the Styrofoam object only needed to be modeled. To handle this, Pymunk, a simple physics engine, was used to model how the

object would move based on collisions with the boat. While complex/realistic simulations are a great tool for testing out ideas, it is important to balance accuracy with efficiency as well.

One distinct problem arose during mASV pushing objects simulation evaluation. Depending on where the boat is placed, this procedure could result in the boat colliding into the Styrofoam object to get to the "start" position. Since the start position is found by flipping the vector pointing from the Styrofoam object to the target position, if the boat is ahead of the Styrofoam object then Styrofoam object's position could intersect with the vector pointing from the boat to the start position. This would result in the boat colliding with the Styrofoam object in an undesirable manner. The simulation revealed that, if this were to occur in the actual experiment, the boat could potentially push the Styrofoam in the wrong direction in an attempt to get to the starting position. To resolve this issue, one solution would be to drive the boat to a intermediate "start" position that is behind the Styrofoam object, but not on the vector pointing from the Styrofoam object to the target position. To do this, another vector can be created that is perpendicular to the vector pointing from the Styrofoam object to the target position. This vector should pass through the original starting position and should be scaled according to the Styrofoam objects size. If the boat travels to this intermediate point first before proceeding to the "start" point, then it should no longer collide with the Styrofoam object prematurely.

Field experiments highlighted another important question which is the importance of sensors. Sensing is an important aspect of humans life; it is through sensing that humans are able to interact with the world around them. For autonomous vehicles to interact with their environment, they too must also have the proper sensing mechanisms. It is important for robots to have accurate and reliable sensor data in order to ensure proper autonomous behavior. For example, the Heron had a faulty IMU sensor was able to completely disrupt the PID controller. The IMU sensor gave the controller wrong information about the orientation of the Heron, and as a result it was unable to properly perform the instructions given to it. The depth sensor produced a lot of noise in its readings, but after adjusting its position, this problem was resolved. Likewise, during UAV experiments, an uncalibrated sensor prevented the drone from properly taking off. In short, sensors are critical to the development of autonomous surface vehicles. As sensor technologies continue to advance, robots will be able to autonomously carry out more complex tasks in an efficient manner.

A. Lessons Learned

This project explores some of the interdisciplinary aspects of research in Robotics. Robotics is a field that requires a combination of principles from other fields such as mechanical engineering, electrical engineering, and computer science. As a computer science major, it was important to acquire skills and knowledge in both software and hardware in order to assist with research within the field of robotics. Although software and hardware may seem to contrast one another, when brought together opens opportunities for for intricate and dynamic machines.

VII. Conclusion

In conclusion, autonomy is a powerful tool for robotics. It allows robots to adapt and adjust to obstacles they may face within their environments. This project aims at presenting an interdisciplinary approach to exploring some of the tools needed to properly run experiments and test behaviors on autonomous surface vehicles (ASVs). A combination of simulations as well as physical experiments are incorporated into this work to test behaviors on ASVs, as well as verify them in the physical world. Sensor technologies are an important aspect of the field of robotics. As sensor technologies continue to advance, autonomous surface vehicles will become increasingly more precise and efficient in their behaviors, allowing more complex and dynamic tasks to be assigned to robots.

VIII. Appendix

A. PCB Components

1) 1 FT232RL chip
2) 1 MAX232EIN
3) 2 100nf Capacitors
4) 1 4.7uf Capacitor
5) 1 10nf Capacitor
6) 5 1uf Capacitors
7) 1 Red LED
8) 1 Green LED
9) 1 Boost Converter
10) 1 USB
11) 1 Ferrite Bead

IX. Acknowledgments

I would like to thank University of Pennsylvania's SUNFEST REU for the opportunity to work and gain valuable experience in the field of robotics this summer. Thank you to Torrie Edwards, my research mentor, for all your guidance and support throughout this program. Thank you Dr. Hsieh for being an incredible PI and allowing me to be apart of your lab this summer. Special thanks to everyone at ScalAR labs for incorporating me into the lab and making my experience an extremely valuable one. I would also like to thank the National Science Foundation for funding this program and making this opportunity possible (Grant Number 1950720).

References

[1] K. G. Sellner, G. J. Doucette, and G. J. Kirkpatrick, "Harmful algal blooms: causes, impacts and detection," Journal of Industrial Microbiology and Biotechnology, vol. 30, no. 7, pp. 383–406, Jul 2003. [Online]. Available: https://doi.org/10.1007/s10295-003-0074-9

[2] T. Salam and M. A. Hsieh, "Heterogeneous robot teams for modeling and prediction of multiscale environmental processes," arXiv preprint arXiv:2103.10383, 2021.

[3] Y. Chung, C. Park, and F. Harashima, "A position control differential drive wheeled mobile robot," IEEE Transactions on Industrial Electronics, vol. 48, no. 4, pp. 853–863, 2001.

[4] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents part i: dynamic topology," in 42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475), vol. 2. IEEE, 2003, pp. 2016–2021.
[5] P. C. Chen and Y. K. Hwang, "Practical path planning among movable obstacles," Sandia National Labs., Albuquerque, NM (USA), Tech. Rep., 1990.
[6] J. Chen, M. Gauci, W. Li, A. Kolling, and R. Groß, "Occlusion-based cooperative transport with a swarm of miniature mobile robots," IEEE Transactions on Robotics, vol. 31, no. 2, pp. 307–321, 2015.
[7] J. Stüber, C. Zito, and R. Stolkin, "Let's push things forward: a survey on robot pushing," Frontiers in Robotics and AI, vol. 7, p. 8, 2020.
[8] M. W. Spong, S. Hutchinson, and M. Vidyasagar, Robot modeling and control. John Wiley & Sons, 2020.
[9] V. Vijaya, R. Valupadasu, B. R. Chunduri, C. K. Rekha, and B. Sreedevi, "Fpga implementation of rs232 to universal serial bus converter," in 2011 IEEE Symposium on Computers & Informatics. IEEE, 2011, pp. 237–242.
[10] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org
[11] J.-P. Charras, F. Tappero, and W. Stambaugh, KiCad Complete Reference Manual, 2018.