Investigating the Capabilities of Miniature Autonomous Surface Vehicles through a Game of Pong

Abby Joseph (University of Maryland, Baltimore County, Computer Engineering), *SUNFEST Fellow*

Dr. Ani Hsieh, Department of Mechanical Engineering and Applied Mathematics

Abstract— **Monitoring underwater weather is important to understanding and maintaining the health of large water bodies, such as rivers, lakes, and oceans. Leveraging autonomous surface vehicles (ASVs) with on-board sensing capabilities can provide more useful and consistent information that captures the state of these water bodies. While large ASVs are currently in development at the ScalAR Lab, we simulate the performances of similar, yet smaller boats, miniature ASVs (mASVs). Investigating the performance of planning algorithms on board these mASVs can allow for a deeper understanding of the capabilities of coordinated tasks with teams of ASVs. In this project, a real-time planning algorithm for the mASVs is developed in Python and implemented to run in the Multi-Robot Tank (MR tank), to simulate a game similar to pong. In this game, miniature boats deflect an object between one another with a given set of boundaries. With the development of these planning algorithms, these concepts can be implemented into completing tasks for teams of larger ASVs. The implications may include the coordination of these vehicles in completing their given objectives, such as transferring waste materials collected from the surface of the river between multiple ASVs.**

Index Terms—**Marine Vehicles, Unmanned Autonomous Vehicles**

I. Introduction

Rivers have been an invaluable resource in engineering, urbanization, and industrialization. [1] However, the development of cities, climate change, and pollution have drastically altered the ecosystems and biodiversity of these rivers over time. [2] While the development of cities has affected the way the rivers deliver water, sediment, and mud, climate change has affected the volume of floods and periods of dry spells. Rivers that are tidally influenced also have an additionally increased risk of flooding due to rising sea levels. [2] The changes to these rivers also further negatively impact the health of estuaries and their ecosystems. [3] Monitoring underwater weather is important to understand occurrences causing issues like mud depositions and, therefore, important to understand the health of rivers and estuaries. [2, 4]

Currently, the observations made of underwater weather variables, which include salinity, water temperature, and more, do not frequently cover a large enough area. This problem results from stationarily mounted sensors or infrequent human-operated surveys. A solution to this issue would be automating the process by using Autonomous Surface Vehicles (ASVs). [2] An ASV refers to a robotic vehicle that sits on the surface of a body of water while collecting data – in this case, conditions like salinity, bathymetry, chlorophyll levels, and more. Doing so would eliminate the issue of stationary sensors and the need for human operation.

Larger ASVs are currently in development at the ScalAR Lab for deployment and monitoring of the Schuylkill River in Philadelphia. However, miniature ASVs (mASVs) are a good way to practice and implement potential tools for future ASV uses. This project utilized the mASVs with implemented software that allowed them to play a game of pong.

The movement of these mASVs simulates a game of pong by using an autonomous boat as a ball. The implications of using mASVs to play a game of bong include coordinating the movements of ASVs to complete tasks like transferring materials between one another, and generally having awareness to the positions of other ASVs in a team. This problem is particularly challenging because of the nature of the environment when utilizing ASVs. The centralized command center will have to constantly keep track of the boats current and desired positions, while correcting them due to the movement of the water in the tank. This can have many implications beyond simply simulating a game when you consider the coordination of movements of the boat when using a single command center.

II. Background

Autonomous surface vehicles are notable for their high mobility, low cost, autonomy, and ability to complete tasks without risking human lives. Currently, ASVs are utilized in several fields, including the military, civil engineering, search and rescue, environmental monitoring, and more [5]. The current ASVs in development at the ScalAR lab include the Clearpath Heron and an originally designed lightweight ASV. As previously mentioned, these boats roam the water's surface and collect data using multiple sensors, including bathymetry, suspended sediment concentration, and inertial measurement unit. They are designed to run autonomously in a lawnmower pattern to cover the area of the river. [2] These vehicles are deployed as a single unit to run and collect data individually. Being able to deploy multiple ASVs and coordinate their actions would provide several advantages, especially in this context of environmental monitoring. For example, they could more efficiently sample a larger area. However, this would introduce the challenge of coordinating the autonomous movements of the ASVs.

When focusing on coordinating the movements of several ASVs, motion control becomes the main challenge. There exist many challenges involved in motion control for ASVs as a field. However, this research focuses on utilizing motion control to explore the capabilities of ASVs through mASVs. Motion control for coordinated teams of ASVs can be broken up into groups of common scenarios. These scenarios include "dynamic positioning, trajectory tracking, path following, and target tracking" [5].

Dynamic positioning is the ability of an ASV to maintain its heading and positioning by use of active thrusters while being met with disturbances from the body of water. [6] This scenario is essential for ASVs like the Heron and lightweight ASV because they need to maintain their lawnmower pattern despite the river's flow. Trajectory tracking refers to making the vehicle drive along a pre-defined trajectory. Path following refers to driving the vehicle along a predetermined path [5]. This is one of the main motion control scenarios used in this research, as the control system uses pathfinding to determine the path for the mASVs. Target tracking, another motion control scenario utilized, is tracking a moving object by driving the ASV [5]. This scenario is necessary for tracking the target object so that it may continue to be pushed until it reaches the desired position.

Overall, the focus of this research is the software development to coordinate the movements of three mASVs to play a stable game of Pong. Coordination in this manner introduces the capabilities for larger ASVs to transfer materials between each other, like assembly lines. There are a few different control architectures used when coordinating teams of ASVs. These methods include centralized control, decentralized control, and distributed control.

Centralized control indicates having a single command center responsible for obtaining information about each vehicle, making decisions based on that information, and controlling each vehicle. This command center can be remote or stored with a single "leader" vehicle. In this case, a remote command center is utilized to control all the mASVs, which allows the design of the mASVs themselves to be kept very simple.

III. METHODOLOGY

The main components of these robots that would allow them to accomplish these goals are the mASVs hardware and the code used to control them – which is where the main contributions of this research lie.

A. Hardware

The hardware contained within the body of each mASV consists of an Arduino Fio and an XBee module. The Arduino Fio is a wireless microcontroller board that allows the boat to receive commands from the computer (the central command center) where the code is run. It is used with the XBee module, another wireless tool, to communicate with the boats. Commands are sent from the central control system to this hardware stored within the boats to coordinate the movements of multiple mASVs. In doing so, the mASVs' two motors are set to turn at different speeds depending on the intended direction and travel distance.

B. ROS and OptiTrack

A necessity for motion control is allowing the robot to understand where it is in its environment. For the mASVs, Robot Operating System (ROS) and OptiTrack (a motion capturing system) are utilized to obtain and use positioning data. These programs track the position of the boats in the MR tank and allow the command center to utilize this information as the code is running. In terms of the controller for the mASVs, they allow the user to set waypoints for the boat within a defined coordinate system in the tank, which the boats are made to move towards.

C. Pong Rules and Assumptions

While the boats are playing "pong," the mASVs are functioning under the assumptions that the ball is going to reflect off of the boundaries at the same angle it approaches the boundary (law of reflection). The same assumption will apply to collisions with the participating mASVs. Another assumption was that the opponents in the simulated version of Pong did not have a maximum velocity. On the other hand, the mASV opponents did have a maximum velocity. These disparities later caused difficulties in transitions from simulated pong to the real-life experiments.

D. Simulations

To develop the algorithms necessary for the final version of pong to run, simulations were created using matplotlib to accomplish the goals of the program in a virtual environment. The first simulations generated were that of

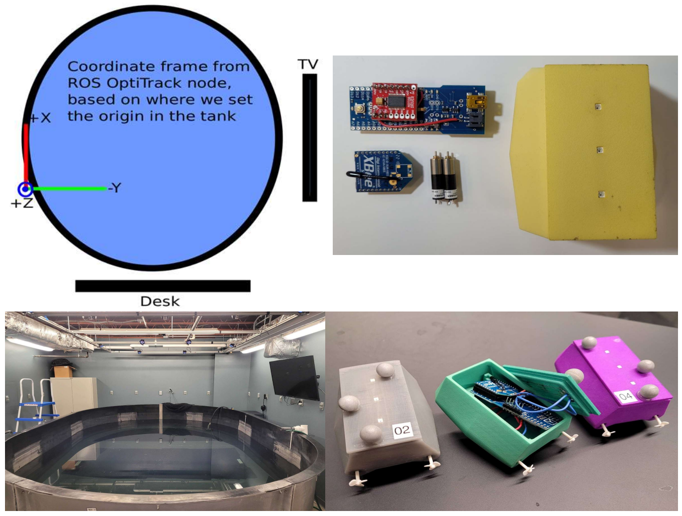
Coordinate frame from ROS OptiTrack node, based on where we set the origin in the tank
+X
-Y
+Z
TV
Desk
02
04

Fig. 1: A diagram of the coordinate system in the MR Tank (top left), the hardware of the mASVs used in the real-life experiments (top right), the MR Tank and OptiTrack setup (bottom left), and the three mASVs used in the real-life experiments (bottom right).

Algorithm 1 Pong Simulation Algorithm

```
state ← True
while state do
    counter ← 1
    if counter = 1 then
        the first opponent predicts where the ball will land along its axis
    end if
    if the angle the ball is travelling at is negative and the ball is in the top half of the boundaries then
        the first opponent predicts where the ball will land along its axis
    end if
    if the angle the ball is travelling at is positive and the ball is in the bottom half of the boundaries then
        the second opponent predicts where the ball will land along its axis
    end if
    if the first opponent has a target then
        Set the position to move towards the target
    end if
    if the second opponent has a target then
        Set the position to move towards the target
    end if
    Move the ball according to assumptions
end while
```

Algorithm 2 Pong Simulation - Ball Movement

```
x ← currentPosition[0] + (velocity * cos(angleOfMovement))
y ← currentPosition[1] + (velocity * cos(angleOfMovement))
if x < 3 or x > 17 then
    angleOfMovement ← -π - angleOfMovement
    x ← currentPosition[0] + (velocity * cos(angleOfMovement))
    y ← currentPosition[1] + (velocity * cos(angleOfMovement))
else if the distance from either robot < 0.2 then
    angle of movement = -(angle of movement)
    x ← currentPosition[0] + (velocity * cos(angleOfMovement))
    y ← currentPosition[1] + (velocity * cos(angleOfMovement))
end if
```

Algorithm 3 Pong Simulation - Opponent 1 Movement

```
targetPositionFound = False
x ← currentPosition[0]
y ← currentPosition[1]
steps ← 0
while targetPositionFound ! = True do
    steps ← steps +1
    xTemp ← x + (ballVelocity * cos(angleOfBallMovement)
    yTemp ← y + (ballVelocity * sin(angleOfBallMovement)
    if xTemp < 3 or yTemp > 17 then
        angleOfBallMovement = π− angleOfBallMovement        ▷ Use formula −π− angleOfBallMovement if Opponent 2
        xTemp ← x + (ballVelocity * cos(angleOfBallMovement)
        yTemp ← y + (ballVelocity * sin(angleOfBallMovement)
    else if yTemp < 4 then
        angleOfBallMovement * = −1
        xTemp ← x + (ballVelocity * cos(angleOfBallMovement)
        yTemp ← y + (ballVelocity * sin(angleOfBallMovement)
    end if
    x ← x Temp
    y ← yTemp
    if 4.0 <= y <= 4.1 then
        targetRositionReached ← True
    end if
end while
targetRosition ← [x, 4]
xDistance ← currentPosition - x
velocity ← xDistance / steps
```

Algorithm 4 Reality Pong Algorithm (After boats reach starting positions)

```
while there have been less than 10 returns in the game do
    if the ball reaches its current waypoint then
        the ball calculates its next waypoint
    end if
    if the second opponent reaches its current waypoint and the ball is moving at a negative angle and this opponents current waypoint differs from its last then
        the opponent calculates its next waypoint
    end if
    if the first opponent reaches its current waypoint and the ball is moving at a positive angle and this opponents current waypoint differs from its last then
        the boat calculates its next waypoint
    end if
    Update the state of each mASV and the controller of each mASV
    if any of the boats are less than 0.25 m away from the ball and the boat and the ball have the same waypoint and this boat was not the last boat to return the ball then
        add to the return count
    end if
end while
shut down all of the boats
```

Algorithm 5 Reality Pong - Finding the Ball's Next Waypoint

```
if currentWaypoint[0] = 1 or -0.5 then                ▷ the side boundaries
    if −π < ballAngleOfMovement < 0 or π < ballAngleOfMovement < 2∗π then
        ballAngleOfMovement ← −π− ballAngleOfMovement
    else
        ballAngleOfMovement ← π− ballAngleOfMovement
    end if
end if
if currentWaypoint[1] = -2.5 or -0.5 then        ▷ The axis of movement of the opponents
    if distanceFromCurrentWaypoint < 0.2 then
        ballAngleOfMovement ← -ballAngleOfMovement
    end if
    if ballAngleOfMovement < 0 then
        yDistance ← -2.5 - currentPosition[1]
    else
        yDistance ← -0.5 - currentPosition[1]
    end if
    xDistance ← yDistance / tan(ballAngleOfMovement)
    nextX ← currentPosition[0] + xDistance
    if 1 < nextX then
        nextX ← 1
        xDistance ← 1 - currentPosition[0]
        yDistance ← tan(ballAngleOfMovement) * xDistance
        nextY = currentPosition[1] + yDistance
    else if −0.5 > nextX then
        nextX ← -0.5
        xDistance ← -0.5 - currentPosition[0]
        yDistance ← tan(ballAngleOfMovement) * xDistance
        nextY = currentPosition[1] + yDistance
    else
        if ballAngleOfMovement < 0 then
            nextY = -2.5
        else
            nextY = -0.5
        end if
    end if
    Set waypoint controller to [nextX, nextY]
```

Fig. 2: The algorithms used in both simulation and in the real-life experiments. Similar to how Algorithm 3 utilizes Algorithm 2 to predict the movement of the ball, the mASV opponents in the real-life experiments use Algorithm 5 to predict the mASV ball's movement and sets their waypoints accordingly.

obstacle avoidance within a given boundary. This helped in getting familiar with the process of creating and using object classes to keep track of the important variables for moving the boats, and helped develop basic functions that would often be used in the development of the final code, including finding the distance between boats and their target positions, as well the angle in which the boat is travelling relative to the horizontal x-axis. When developing a point-based simulation for the pong code, a similar approach was used by creating classes similar to that of the simulation code when implementing the use of OptiTrack and ROS.

E. Implementation

There were multiple aspects of the mASVs that needed to be tracked within the software to ensure the success of the program.
While developing the algorithms necessary to run the experiment, there were several variables that were significant in getting the pong simulation to run. The first and most obvious piece that needed to be constantly called was the current position of the boats in the tank. Understanding where the boats were at any given time during the experiment ensured the proper autonomous prediction of the movement of the ball and that the mASVs could move to the correct positions to receive the ball.

Similarly, the angle at which the ball is moving relative to the x-axis was necessary to be able to predict where it would land for the boats to deflect. This prediction is what allowed the boats to generate their next waypoint at every return. As for waypoints, these needed to be kept as a variable so that the boats and the ball could utilize OptiTrack to allow the ball to move to its next collision with a boundary or a boat, and so the boats could move to their next collision point along their defined axis of movement.

A final important variable is a distance between the boats/ball and their desired waypoint. Knowing this distance allows the ball to generate its next waypoint due to a collision and allows the boat to realize it needs to wait for contact with the ball to generate its next waypoint.

F. Data Collection

A successful run was considered a rally with at least one return by each boat. To judge the accuracy of the trajectory of the boat throughout the game compared to the ideal path produce by the waypoints, the positions of the mASVs throughout the game was saved, as well as their actual waypoints.

IV. Experimental Results and Discussion

The data collected on the waypoints and actual positions of the boats during the game of pong allowed for the visualization of the true and ideal trajectories of the boats throughout the game. For example, Fig. 3 displays the real and ideal trajectories of the ball mASV and an opponent mASV over the course of two returns. You can see there is more wavy movement for the real trajectory of the mASV ball than there is for the ideal trajectory. Furthermore, the mASV opponent does not exclusively move along its axis of movement but moves back and forth around its waypoint when waiting for a collision with the mASV ball.

These visualizations show how much more unpredictable the movement of the ball is due to its wavy movement, in comparison to the perfectly linear movement in simulation.

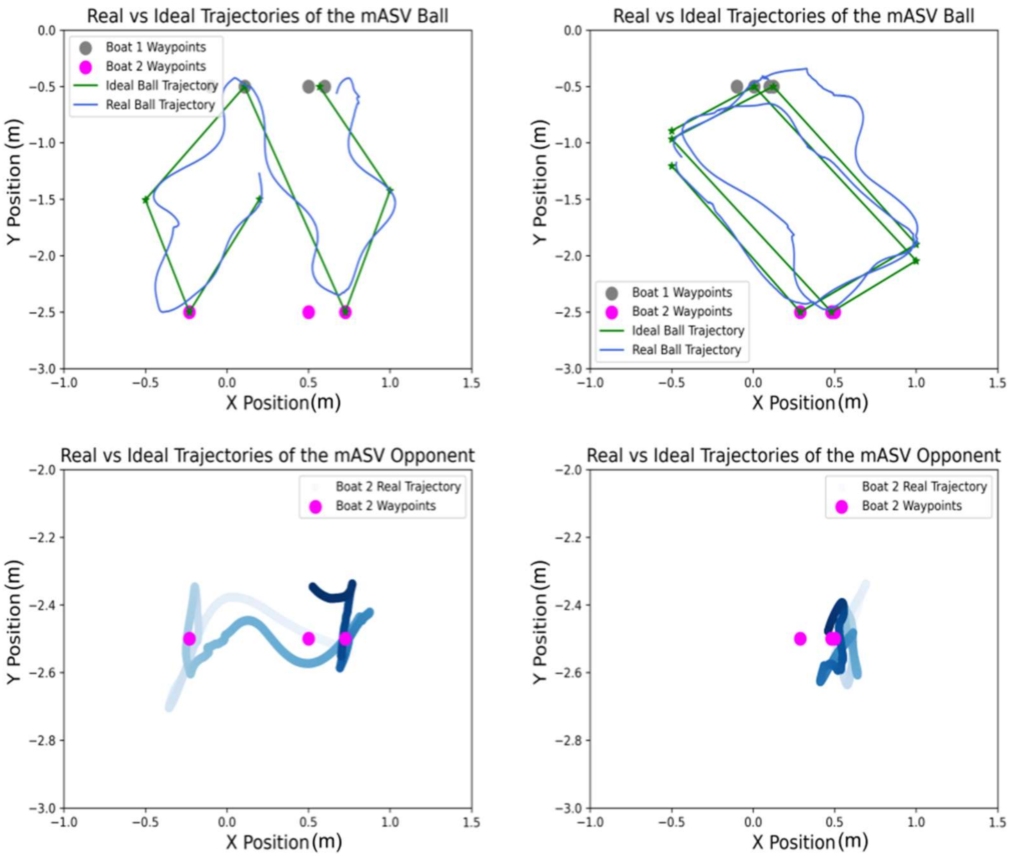

Figure 3: The real and ideal trajectories of the mASV Ball and the second boat opponent over two returns during two separate rallies (paired vertically).

This can be due to multiple different factors, including the additional forces on the ball mASV due to the water, as well as issues with the controller. The controller may have had difficulty in updating the current position of the mASV due to the structure of the program, which could have allowed for more errors in following the ideal trajectory. Thus, it would be worthwhile edit the structure of the pong algorithms to ensure OptiTrack is efficiently capturing the positions of the mASVs.

As previously stated, a successful run was considered a rally with at least a single return by each opponent. During the best runs, the opponents got up to four returns each. Thus, the experiments were successful. This success was reliant on the ability of the program to coordinate the movements of these robots so that they would collide. Although it may seem as though the robots are competing against one another in the game, they must be able to track and predict the movement by considering the ball's current position and movement patterns, as well as their own current position. When extending this level of coordination to larger ASVs, you can see how broadening the application can help in completing tasks like chagrining batteries because the vehicles need to be able to similarly perceive and plan to be able to interact with each other.

V. Conclusion

Overall, the goal of this project was to create a game of pong using the mASVs, which was successfully accomplished in both simulation and reality. Currently, this project assumes that the localization is completely correct. Given this, future work that can be done with this project includes introducing noise to the incoming OptiTrack data and testing to see if the boats can still successfully play pong.

Another potential future direction for this work is adding disturbances to the water while the boats are playing pong. Introducing these disturbances would add another layer to the complexity of the prediction algorithms, as they would have to predict how the ball will react to these flows to predict where its next waypoint should be. Furthermore, the boats would have to be able to understand the flow so they could resist it and continue the game of pong.

Acknowledgment

The author would like to thank her mentor, Alice K. Li; her P.I., Dr. M. Ani Hsieh; and everyone at the ScalAR Lab for helping create this great experience, as well as the SUNFEST REU Program for this opportunity. She would also like to thank the National Science Foundation for supporting SUNFEST through NSF REU grant no. 1950720.

References

[1] [1] C. J. Vörösmarty, P. B. McIntyre, M. O. Gessner, D. Dudgeon, A. Prusevich, P. Green, S. Glidden, S. E. Bunn, C. A. Sullivan, C. R. Liermann, and P. M. Davies, "Global threats to human water security and River Biodiversity," *Nature*, vol. 467, no. 7315, pp. 555–561, 2010.

[2] [2] Alice K. Li et al., "Towards Understanding Underwater Weather Events in Rivers Using Autonomous Surface Vehicles," GRASP Lab, Philadelphia, PA, USA.

[3] [3] Z. Yang, T. Wang, N. Voisin, and A. Copping, "Estuarine response to river flow and sea-level rise under future climate change and human development," *Estuarine, Coastal and Shelf Science*, vol. 156, pp. 19–30, 2015.

[4] [4] R. Monroe, "Underwater Weather," *Weatherwise*, vol. 55, no. 3, pp. 14–21, 2002.

[5] [5] Z. Peng, J. Wang, D. Wang, and Q.-L. Han, "An overview of recent advances in coordinated control of multiple autonomous surface vehicles," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp.
[6] A. J. Sørensen, "A survey of dynamic positioning control systems," *Annual Reviews in Control*, vol. 35, no. 1, pp. 123–136, Apr. 2011.