Spring 5-3-2019

Autonomous Watercraft Simulation and Programming

Nicholas J. Savino
Lynchburg College, savino_n@lynchburg.edu

Autonomous Watercraft Simulation and Programming

Nicholas J. Savino

Senior Honors Project

Submitted in partial fulfillment of the graduation requirements

of the Westover Honors College

Westover Honors College

May 3, 2019

William Roach, PhD

Crystal Moorman, PhD

Nancy Cowden, PhD

Abstract

Automation of various modes of transportation is thought to make travel more safe and efficient [1]. Over the past several decades advances to semi-autonomous and autonomous vehicles have led to advanced autopilot systems on planes and boats and an increasing popularity of self-driving cars. We simulated the motion of an autonomous vehicle using computational models. The simulation models the motion of a small-scale watercraft, which can then be built and programmed using an Arduino Microcontroller. We examined different control methods for a simulated rescue craft to reach a target. We also examined the effects of different factors, such as various biases (which would be analogous to a current of water) and various initial separation distances, on the time it takes the simulated rescue craft to reach the target. The simulations suggested that it is most efficient to continually correct the direction of the simulated rescue craft for movement of the target when the object is moving at random. We predict that these simulations can model not only the small-scale watercraft but also full-size boats. Self-driving technology used here can be applicable in search-and-rescue missions where conditions may be too harsh for human-controlled watercraft and impractical for remote-controlled watercraft. This experiment also raises new questions in methods of control that can utilize machine learning to detect patterns of a moving target.

Introduction

An autonomous vehicle is a vehicle that is able to control itself without direct input from a human operator. Self-driving cars are becoming more mainstream. Aircraft use autopilot systems to fly themselves [2, 3]. Even technology such as Roomba vacuum cleaners operate autonomously [4]. The automation of vehicles allows, in a most fundamental sense, for self-driving vehicles to either seek out or avoid certain things. We sought to create a system that most efficiently reaches a target object.

Autonomous vehicles are capable of operating where it is either too dangerous or not feasible for humans to operate a vehicle. Military drones have been used throughout most of the 21st century in order to avoid putting human troops in danger [5]. Autonomous cars continue to be improved upon, allowing for safer operation. It is thought that self-driving cars could become safer than human operated vehicles since an autonomous car will not be prone to the same distractions that a human driver would experience [1].

In order to assess their surroundings, autonomous vehicles utilize various sensor inputs. For example, a self-driving car utilizes global positioning system (GPS) data [6] so that it knows where it is and how to get to a destination. Self-driving cars take in data from sonar and radar sensors, ensuring that they would avoid hitting other vehicles and pedestrians in the road. Other sensors within the car would also provide data to the control system of the car, so that the car operates efficiently and safely [7, 8].

Autonomous watercraft are not as mainstream as self-driving cars, however there are many potential uses. Self-driving boats are particularly useful in situations of search and rescue [8]. In a search and rescue situation, an autonomous watercraft seeks out an object, which may be a human being or a lifeboat in distress. We examined how an autonomous watercraft can seek out a target object and reach that object in the quickest time possible.

We sought to understand the behaviors of an autonomous watercraft when seeking an object using sonar. On water a craft is free to drift in the direction at which it had been previously moving, even if that direction is not directly straight ahead. On land, a vehicle is limited to travel in the direction of the tires, assuming the vehicle is traveling at a speed at which the tires are not skidding.

We simulated an autonomous watercraft in a computer model that would seek out a randomly moving target object. We looked for the most efficient and fastest way to reach an object, which would be critical should this be a search and rescue situation. This system did not depend upon GPS coordinates.

The general application for this experiment would be related to aquatic rescue [9]. There are cases where people require nautical rescue, but due to the dangers involved with poor weather, conditions are often quite dangerous for rescue personnel [7]. Thus, a robotic device can be useful in assisting in these rescues. The autonomous nature of the system we created allows it to operate without input from a human user, limiting the number of necessary people to participate in a rescue operation. This sort of system is not limited to rescue. Understanding the impacts of autonomous control on surface watercraft plays an important role in the future automation of civilian watercraft. Trash retrieval vehicles have also been utilizing autonomous technology [2].

There are already watercraft that operate with autonomous control systems [10]. However, these simulations and small-scale tests will allow us to better understand how different environmental factors can affect the behavior of the craft.

Methods

The Arduino UNO microcontroller (Arduino, Turin, Italy) allows for a relatively inexpensive and simple method of programming robotics. By utilizing the Arduino UNO, it is possible to create a small-scale autonomous watercraft that is capable of seeking out a floating object in the water [11, 12]. A sonar sensor on the craft can be used to sense the location of the object with respect to the watercraft, which would then be the direction the watercraft should orient itself as it travels. The utilization of an open-source microcontroller that can operate a small scale watercraft is far easier to design and build and is far less expensive to test.

We examined simulations in the Python programming language. These simulations were analogous to experimental tests, however they allowed us to predict how an experimental craft might behave. No experimental tests were run. Running simulations allowed us to examine the most efficient control method for the watercraft and assured we did not spend money on experimental tests that may not have produced useful results. The simulated model also allowed for us to control all variables in the experiment, which may otherwise be out of our control during field tests. We also had limited access to calm, open-water sources.

We tested how quickly a simulated watercraft can reach an object that is floating in the water using different methods of control. We also tested to see which type of control method allowed for the watercraft to take the shortest path to reach the target object. We attempted to minimize the path length the watercraft took to reach a target object, thus maximizing the efficiency of the watercraft. Efficiency is generally defined as benefit over cost; assuming it takes the same amount of energy to travel a given distance, efficiency is maximized if the path length that the watercraft travels is minimized.

We simulated an experimental field test, where a sonar sensor detects an object by scanning the surrounding environment. While we did not explicitly simulate a sonar detector, the principle of the simulation and the sonar detector are rooted in the same idea, where the vector pointing towards the target object is found. As the sonar detector rotates, it sends out ultrasonic pulses that bounce off solid objects. The angle, with respect to the boat, which the ultrasonic pulse is sent, is recorded, and the sonar sensors then read back the time it takes for the sonar pulse to return. Using the known speed of sound in air, we were able to use the microcontroller to convert this time into a distance. Thus, we have an array of angles and distances for each scan. The microcontroller pinpoints the angle at which the distance is shortest, thus detecting where an object is. The microcontroller now knows where the object is at a given moment. We tested how we can reach that object in the least amount of time.

The only control for the watercraft was based upon feedback from the sonar sensor. There were no other types of sensors on the watercraft. While target recognition has been previously examined [13], where target objects are distinguished from other objects, we focused solely on comparing different control methods. We tested different parameters to see if there is any difference in efficiency between various parameters.

We were able to simulate a randomly moving target object that the simulated watercraft would seek. It is very easy to generate a randomly moving target in a simulation. The target object is able to move in a random manner in the simulation, where there is no pattern associated with the movement of the object (the Python code generated pseudo random numbers, but when examining data sets of this size, the pseudo random numbers are effectively random). In an experimental test, there are environmental factors that can affect the ‘randomness’ of a randomly moving object. A simulation can examine how an autonomous vehicle tracks a moving target more easily. We examined how to efficiently track an object that is moving at random. We predicted that this type of movement is a good representation of the movement of an actual person in distress while treading water or a small lifeboat floating aimlessly in the water. These situations would be directly applicable to a full-scale, autonomous watercraft used for water rescue.

In the simulations we examined a moving target that travels in random directions, as if it were simply out of control and traveling with no sense of direction. We also introduced a directional bias into the simulations, as if there was a current pushing the target object. The movement of the target was generated with what is called a random walk, where a computer-generated number determines the direction of the object. This walk is confined to the four cardinal directions, but when scaled such that the movement steps of the object are small compared to the initial separation distance between the target and rescue craft, the movement appears to be smooth, thus more realistic.

Versions of the random walk and random number generators can be used for various other applications displaying random movement. We utilized Python's built in random number generator to generate our random number in the simulation. The random walk is essential to the movement of the target object, in order to model a realistic rescue situation, where it is hard to predict the movement of a target. The path an object would take on a random walk can be seen in Fig.1. After every step of the random walk, the bias is also added to the step that the target object takes (when we ran simulations that included a bias). For example, if we wanted to simulate a strong current pushing in the negative x direction, after every iteration, the x coordinate can be subtracted by a constant. By adding this bias in the simulation, we were able to recreate wind or water current that may be present in a field test. Vector fields can also be introduced, where a current may be changing strength and/or direction, thus changing the magnitude and direction of the constant added to the coordinates based upon the position of the target, although this was not tested.

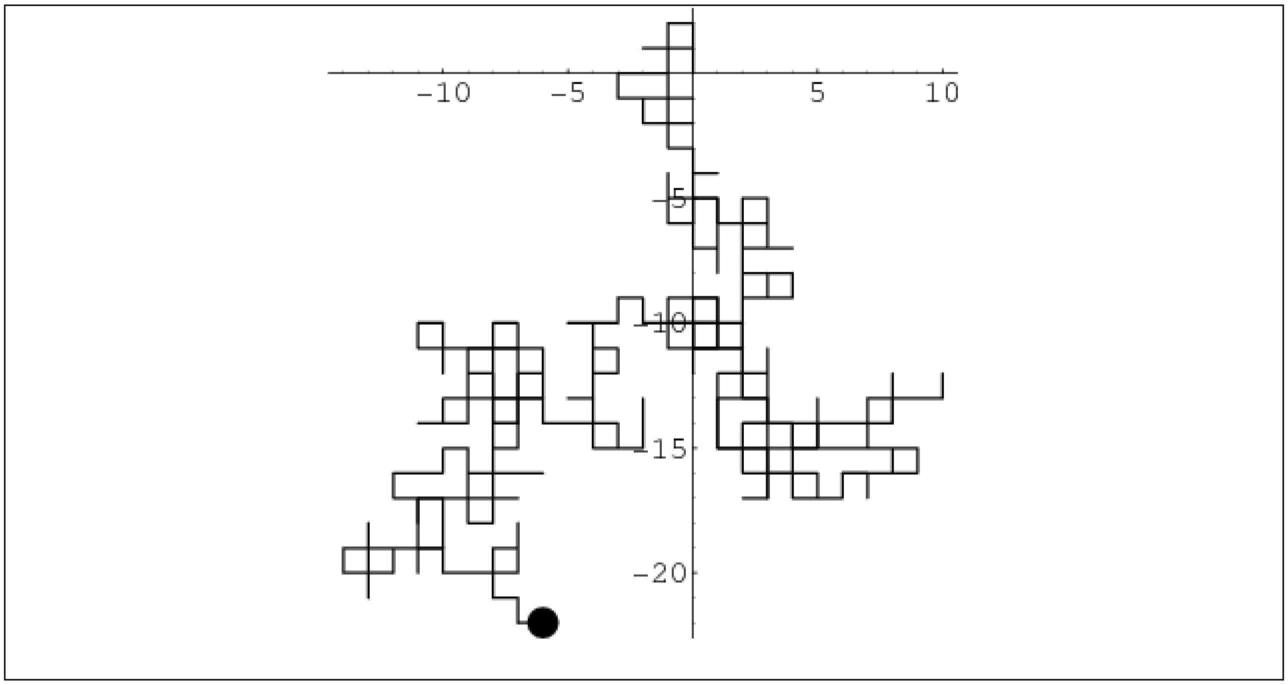

Figure 1: *This is an example of a path that an object on a random walk might take. We can see that the object randomly takes steps in one of the cardinal directions at every iteration. The object begins at the origin, follows the path of the solid line, and ends at the black dot. We predict that this models the movement of a randomly moving swimmer in distress, which we modeled in our simulations* [14].

As previously stated, the Python simulation models a watercraft that uses a sonar sensor to scan for a target object, find the target object's angle with respect to the orientation of the watercraft, and then takes a step towards the object. This is iterated until the watercraft reaches the target object. The simulation measures a vector between the coordinate of the watercraft and the coordinate of the target object, and then the watercraft takes a step toward the target object, along the line of this vector. This is iterated until the simulated watercraft has reached the simulated target object. So that we are able to relate the simulation to realistic situations, with either small or large-scale watercraft, we set up our parameters such that each step takes place over the course of 1 s. Therefore, if we are simulating a watercraft that has a maximum speed of 0.2 m/s, the maximum step size for the simulation would be 0.2 m. We use 0.2 m/s as the maximum speed of the simulated watercraft, since we expect that when an experimental water based test is done, the motors used would be able to be programmed to have a max speed of 0.2 m/s.

These simulations can be used to predict the actual path of watercraft and target object, as shown in Fig. 2, before we begin to utilize sensitive equipment. This simulation does not include confounding factors that could be present when conducting tests in open water, such as unexpected wind, waves, and engineering design issues that would cause the system to behave in ways that are not desirable.

In the Python simulation several different factors were tested. The initial separation distance between the rescue craft and the target object was varied for each set of parameters. One set of parameters consisted of speeds and distances that would be relevant to a small-scale watercraft seeking out another watercraft. Another set of parameters contain the speeds and distances that correspond to a Coast Guard rescue boat and a human swimming in the water. One of the most common rescue boats in the United States Coast Guard, the Fast Response Cutter (154 ft. class), travels at a top speed up to about 8 m/s [13]. This value is used for the speed of the watercraft in the large-scale simulations. As stated before, we take the small-scale watercraft to be 0.2 m/s. We also assume the speed of the randomly moving target object in the large-scale simulations to be 2 m/s, which is close to the speed of a swimmer [14]. We assigned a speed of 0.1 m/s for the target object in the small-scale tests, which is a speed that we expect we should be able to get an experimental target craft to travel with available motors. We also input a bias that only affected the target object (an example of this would be a current that is strong enough to affect a lifeboat in the water but was not strong enough to affect a larger rescue boat in the water) in some tests. The path a simulated watercraft takes when a bias is introduced to the target object is seen in Fig. 3. Table 1 and Table 2 include all of the parameters that we will be testing.

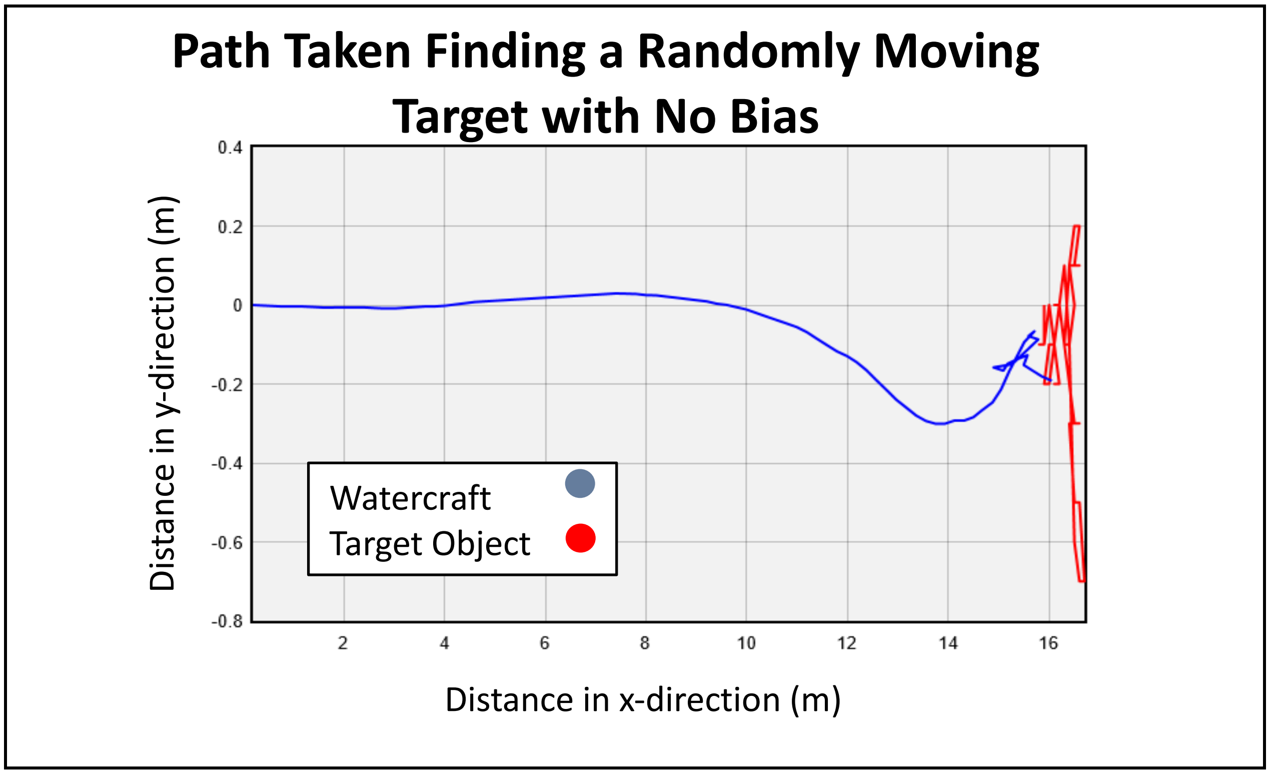

Figure 2: *The simulated path of a small-scale watercraft is shown here. The red shows the path of a target object on its random walk, the blue shows the path of the simulated watercraft as it seeks the object. The parameters correspond to small-scale experimental test; the simulated watercraft travels at 0.2 m/s, the target object at 0.1 m/s, where the target object begins 16 m away initially. Note the order of magnitude of the x and y-axes are different.*

Table 1: *This table contains a summary of small-scale testing parameters. Each parameter is modeled in the Python simulation.*

Small-Scale Testing Parameters		
	Constant Speed while Turning	Variable Speed while Turning (varied by cosine)
Randomly Moving Target with NO Bias	Parameter1	Parameter 4
Randomly Moving Target with Steady Bias	Parameter 2	Parameter 5
Randomly Moving Target with Randomly Changing Bias	Parameter 3	Parameter 6

Table 2: *This table contains a summary of large-scale testing parameters. Each parameter is modeled in the Python simulation.*

Large-Scale Testing Parameters		
	Constant Speed while Turning	Variable Speed while Turning (varied by cosine)
Randomly Moving Target with NO Bias	Parameter 7	Parameter 10
Randomly Moving Target with Steady Bias	Parameter 8	Parameter 11
Randomly Moving Target with Randomly Changing Bias	Parameter 9	Parameter 12

Different control methods were examined. We used a case where the rescue craft is able to make turns of any size instantaneously. This may not appear to be a realistic case, but when there is an initial separation distance that is much larger than the individual step sizes, we predicted that this does not play a significant role in the movement of the rescue craft. In this case the path the craft takes will be smooth (similar to the idea that a large, initial separation distance compared to the step size of the target object will make the path that appears rather smooth in reference to the total distance traveled).

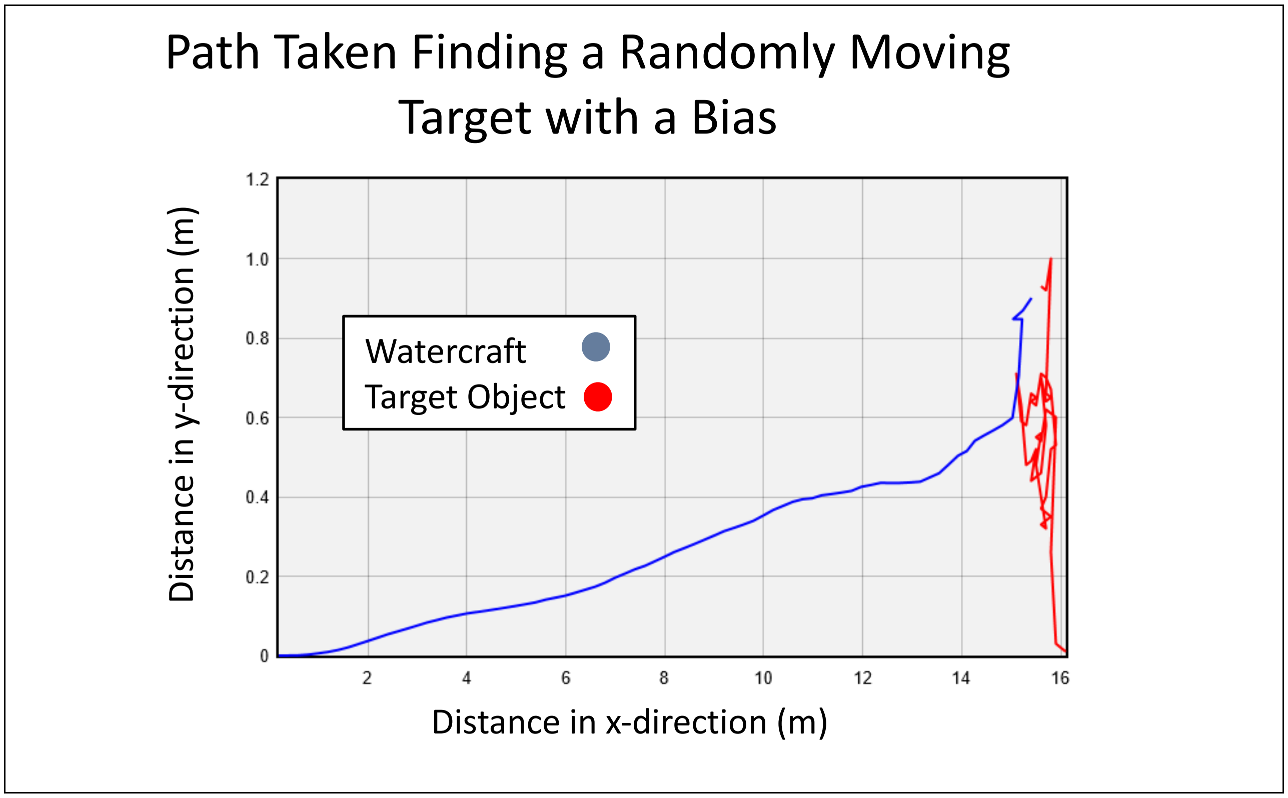

Figure 3: *The simulated path of a small-scale watercraft is shown here. However, in this figure, there is a bias given to the target object, pushing the target object upwards on the page. The red shows the path of target object on its random walk, the blue shows the path of the simulated watercraft as it seeks the object. The parameters correspond to small-scale experimental test; the simulated watercraft travels at 0.2 m/s, the target object at 0.1 m/s, where the target object begins 16 m away initially.*

We then ran simulations where the angle at which the craft turned directly corresponds to how fast it can travel. A vehicle must slowdown in order to make a sharper turn. We assume that there is a relationship between the speed at which the vehicle can take on a turn and the angle at which it is turning, such that the step size that simulation takes is the maximum step size multiplied by the cosine of the angle that the watercraft must turn.

Suppose the watercraft is initially oriented in the $\hat{d}$ direction, as shown in Fig. 4. Suppose the target object located some distance away, at some angle θ with respect to the $\hat{d}$ direction (this is analogous to an experimental watercraft using a radar sensor to find the location of some target object). The simulated watercraft will then take a step towards the target object. The max step would be the largest distance the watercraft can travel in a given step (if each step corresponds to 1 s, and the watercraft can travel 0.2 m/s, the max step size would be 0.2 m). However, if the watercraft must turn, it must slow down in order to negotiate that turn. The larger the turn, the more the watercraft must slowdown, thus if θ is larger, the smaller the step must be. Therefore, we find the step size is the max step multiplied by the cosine of angle θ. We are essentially taking a dot product between the max step vector and the $\hat{d}$ vector to generate the magnitude of the step vector, which will point towards the target object. This process is continually iterated, as the target object moves, allowing the watercraft to adjust its orientation and get closer, until the watercraft has reached the target object. In the future, we plan to examine different methods in which we utilize a linear relationship between the angle θ and the step size.

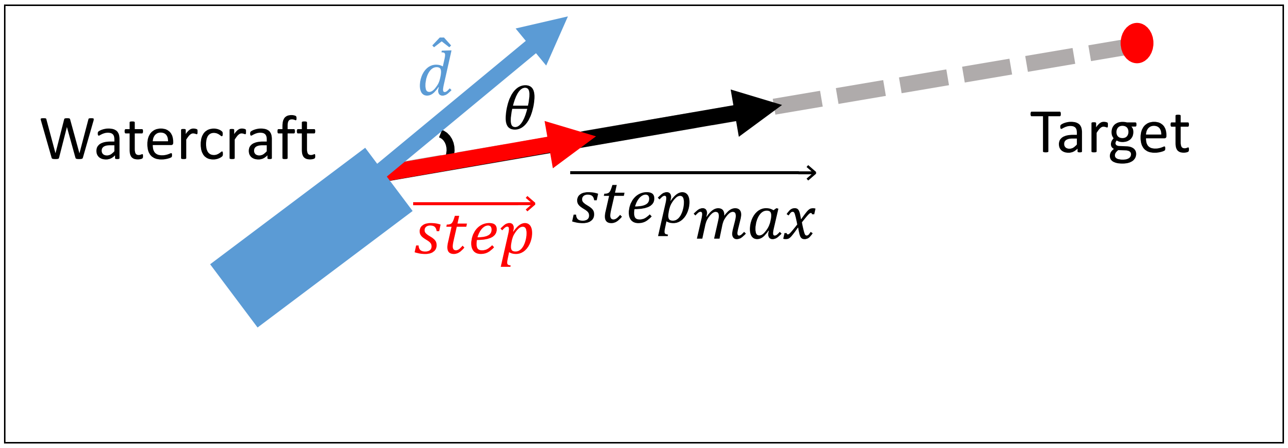

Figure 4: *This illustrates how the simulated watercraft executes a step towards a target, when the speed of the simulated watercraft is varied based upon the angle at which the watercraft must turn.*

This process is illustrated in Fig. 4. Again, initial separation distances and watercraft speed varied for those values which might be reasonable for the small-scale watercraft in a large pool or small pond. One hundred tests were run at each separation distance. At each distance, the number of steps needed to reach the target were averaged and then compared to the respective value of the average number of steps from the tests with the rescue craft that is able to rotate instantaneously. The paths of 100 runs can be seen in Fig. 5.

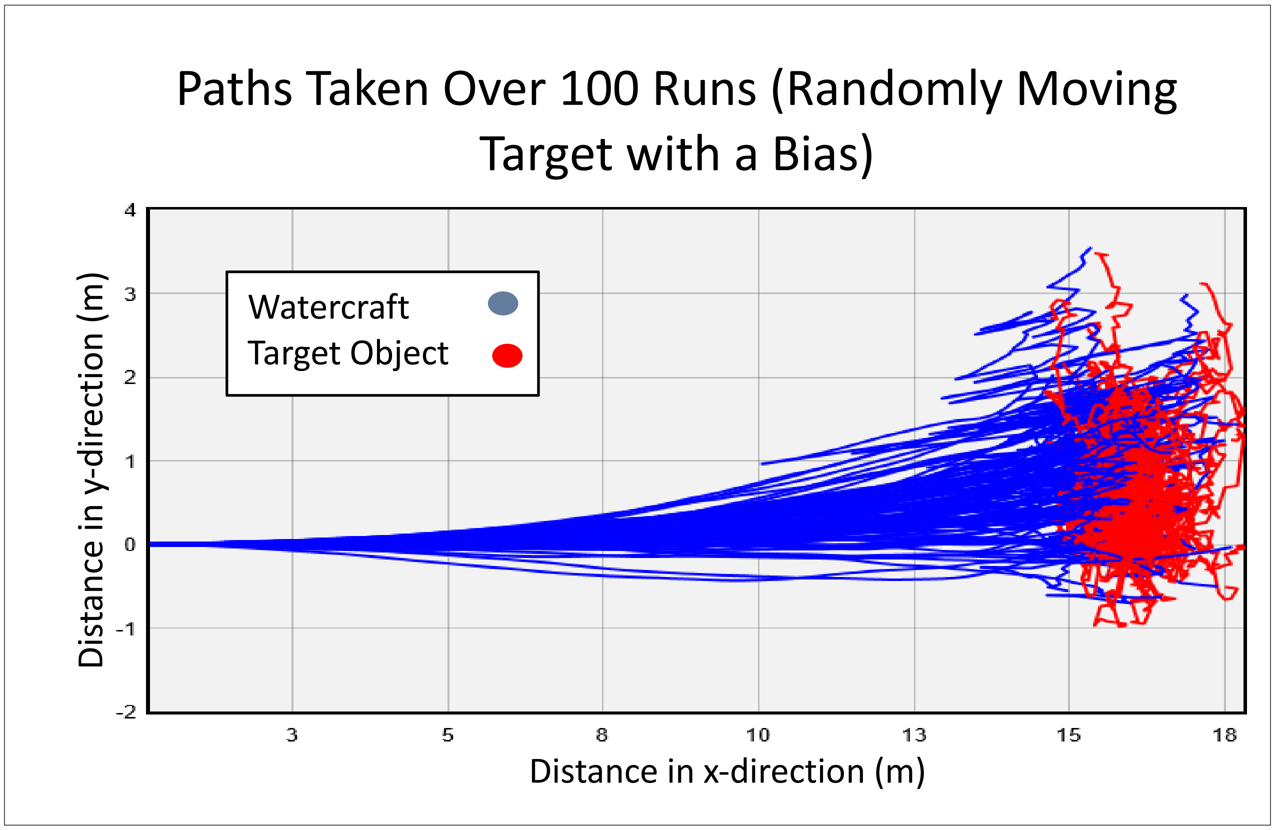

Figure 5: *We were able to run many trials in a very short amount of time and average the number of steps (which is analogous to time) it took for the simulated watercraft to reach the simulated target object. Here there is a bias given to the target object, pushing the target object upwards on the page. The red shows the path of target object on its random walk, the blue shows the path of the simulated watercraft as it seeks the object. The parameters correspond to small-scale experimental test; the simulated watercraft travels at 0.2 m/s, the target object at 0.1 m/s, where the target object begins 16 m away initially.*

Results

As expected, it takes a longer time for the simulated watercraft to reach a target object that begins farther away. However, we are interested in how the initial separation distance relates to the number of steps per initial separation distance.

For the simulated small-scale craft that makes instantaneous turns, we found that the steps per initial separation distance approached 1 as the initial separation increased. When the steps per initial separation distance is exactly 1, the watercraft effectively took a straight line path to the target at a constant speed. This is expected since, as the initial separation distance increases, the path of the watercraft smooths out and become more like a straight line, as seen in Fig.6.

Similar to the case with the small-scale simulation, for the simulated large-scale craft with constant speed, we found that the steps per initial separation distance approached 25 as the initial separation increased. This is expected, since, as the initial separation distance increases, the path of the watercraft smooths out and becomes more like a straight line, as seen in Fig. 7.

Now that we examined the ideal case, where the simulated watercraft does not need to slow down in order to turn towards the target object, we will now examine the more realistic case where the angle at which the watercraft turns affects its speed. The cosine of the angle that the watercraft must turn toward the target is multiplied by the maximum speed of the object.

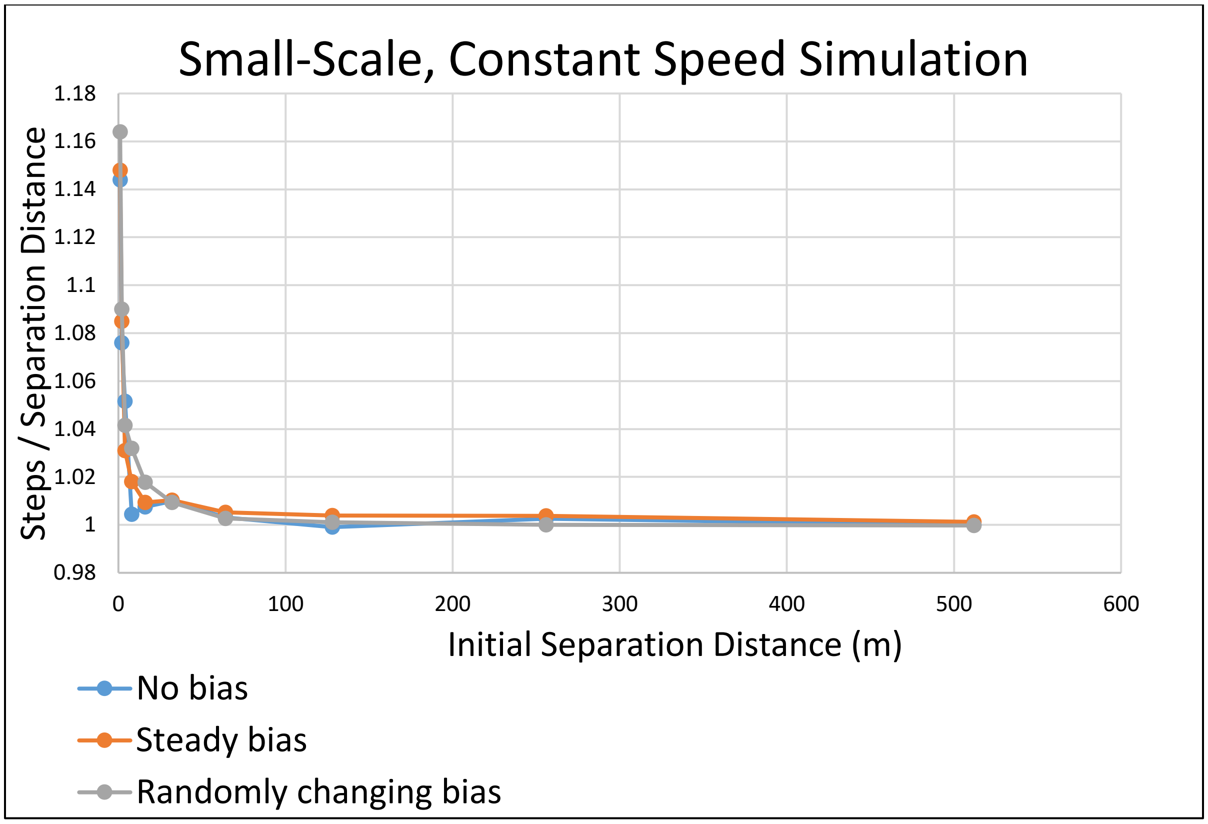

Figure 6: *Plot of the initial separation distance over the total distance traveled plotted versus the initial separation distance. This shows a plot of the relationship between initial separation distance and the number of steps (which is analogous to time) it takes the simulated, small-scale watercraft to reach a randomly moving target. The blue data points represent the runs with no bias on either object, the orange data points represent the runs with a constant bias throughout the runs, and the grey data points represent the runs with a bias that is randomly changing in bias, and direction and we assume the watercraft makes turns instantaneously, and does not need to slow down when changing direction. Each data point represents the average of 100 runs. As we expected, the time it takes the simulated small-scale watercraft to reach its target increases with distance. The points are connected with straight lines, which are in place to aid the eye.*

The simulated small-scale craft with variable speed (cosine function) behaves in a different manner than the constant speed simulation. As the initial separation distance increases, the steps per initial separation distance also increases, as seen in Fig. 8. This appears counter intuitive, and the simulation does not behave as the previous tests, done with the simulated watercraft traveling at a constant maximum speed. However, it makes sense when considering that more small steps need to be taken when the object starts farther away. While we did not examine this, it would be interesting to consider whether the path length behaves in this way as well.

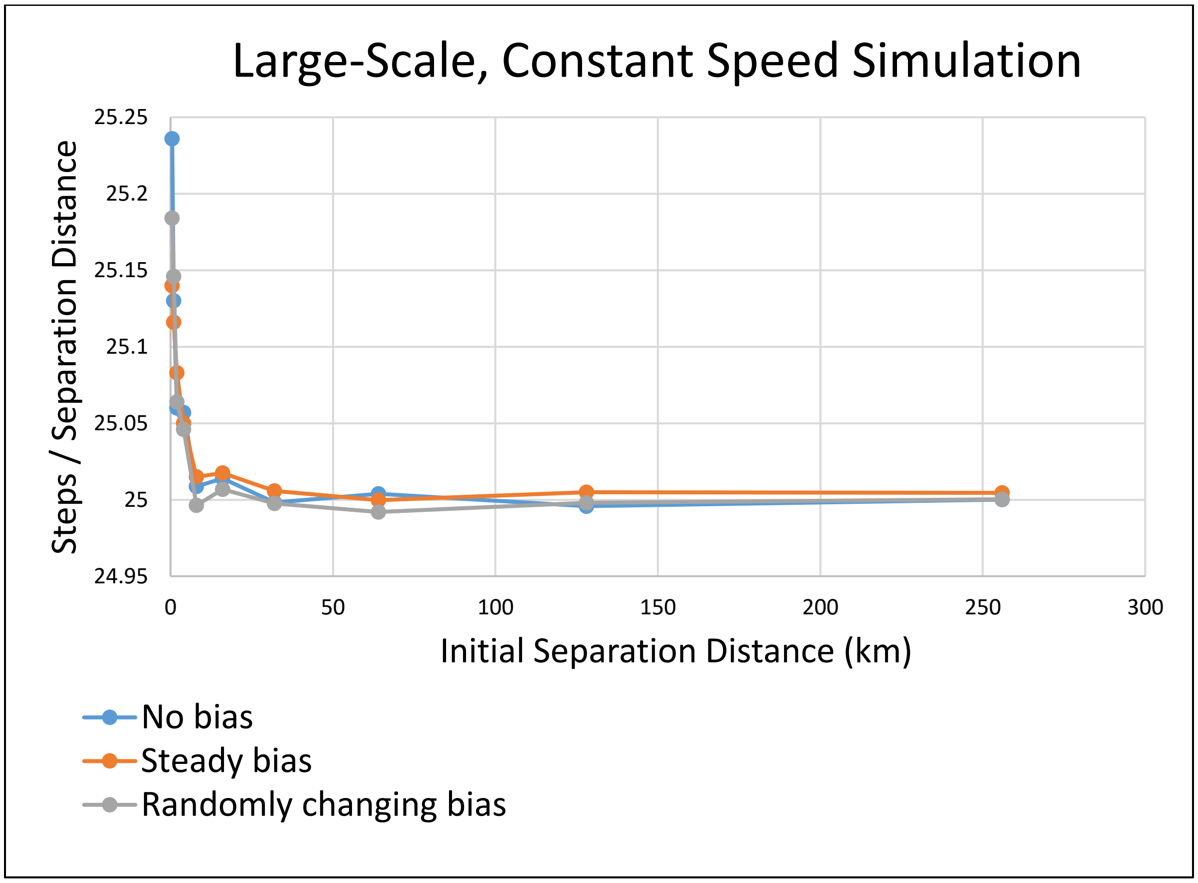

Figure 7: *Plot of the initial separation distance over the total distance traveled plotted versus the initial separation distance. Here the relationship between initial separation distance and the number of steps (which is analogous to time) it takes the simulated, large-scale watercraft to reach a randomly moving target. The blue data points represent the runs with no bias on either object, the orange data points represent the runs with a constant bias throughout the runs, and the grey data points represent the runs with a bias that is randomly changing in bias and direction, and we assume the watercraft makes turns instantaneously and does not need to slow down when changing direction. Each data point represents the average of 100 runs. This is very similar to the small-scale simulations, simply with speed and size scales that are appropriate to apply to full scale tests. As we expected, the time it takes the simulated small-scale watercraft to reach its target increases with distance. The points are connected with straight lines, which are in place to aid the eye.*

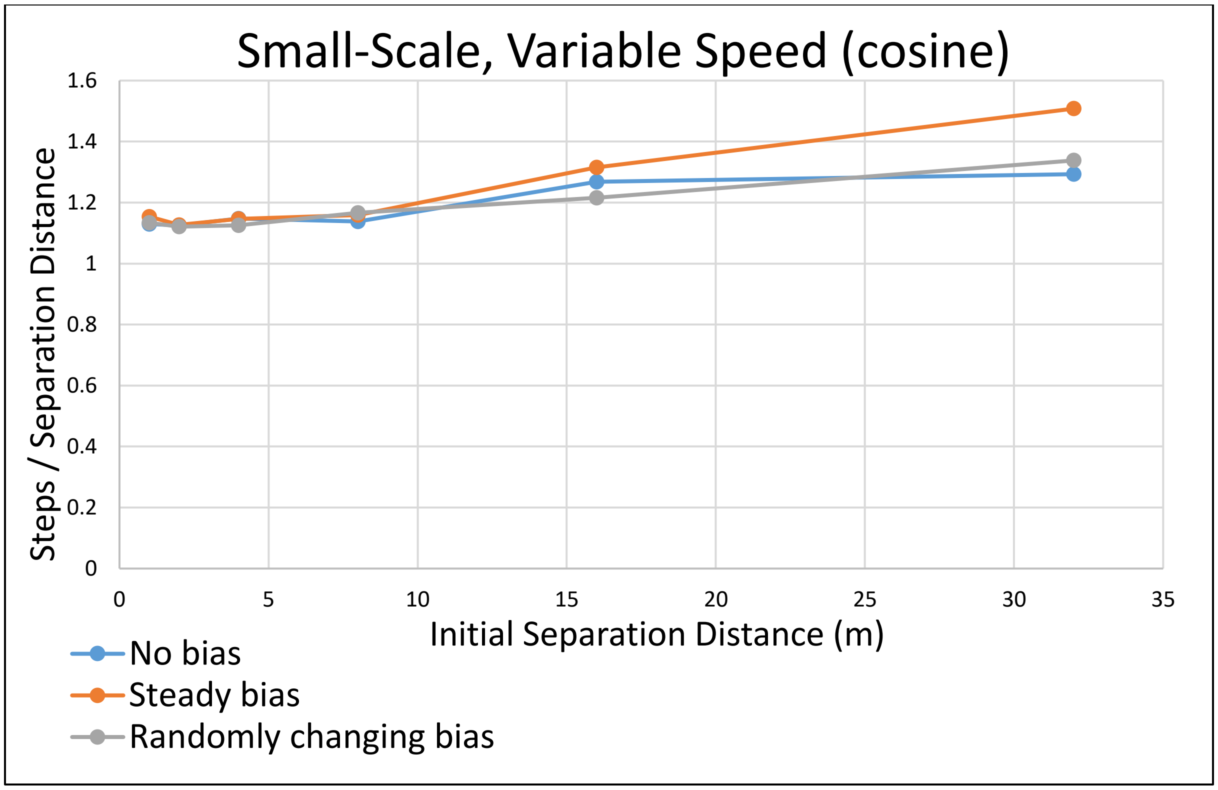

Figure 8: *Plot of the initial separation distance over the total distance traveled plotted versus the initial separation distance. This plot shows the relationship between initial separation distance and the time it takes the simulated, small-scale watercraft to reach a randomly moving target. The blue data points represent the runs with no bias on either object, the orange data points represent the runs with a constant bias throughout the runs, and the grey data points represent the runs with a bias that is randomly changing in bias and direction and we assume the watercraft makes turns instantaneously, and does not need to slow down when changing direction. Each data point represents the average of 100 runs. As we expected, the number of steps (which is analogous to time) it takes the simulated small-scale watercraft to reach its target increases with distance. However, it is not expected that these the number of steps per initial separation distance also grow. The points are connected with straight lines, which are in place to aid the eye.*

We again examine a more realistic case where we consider that the craft must slow down in order to make turns. The cosine of the angle that the watercraft must turn toward the target is multiplied by the speed of the object. Figure 9 shows a plot of the relationship between initial separation distance and the time it takes the simulated, small-scale watercraft to reach a randomly moving target. As we expected the time it takes the simulated small-scale watercraft to reach its target increases with distance.

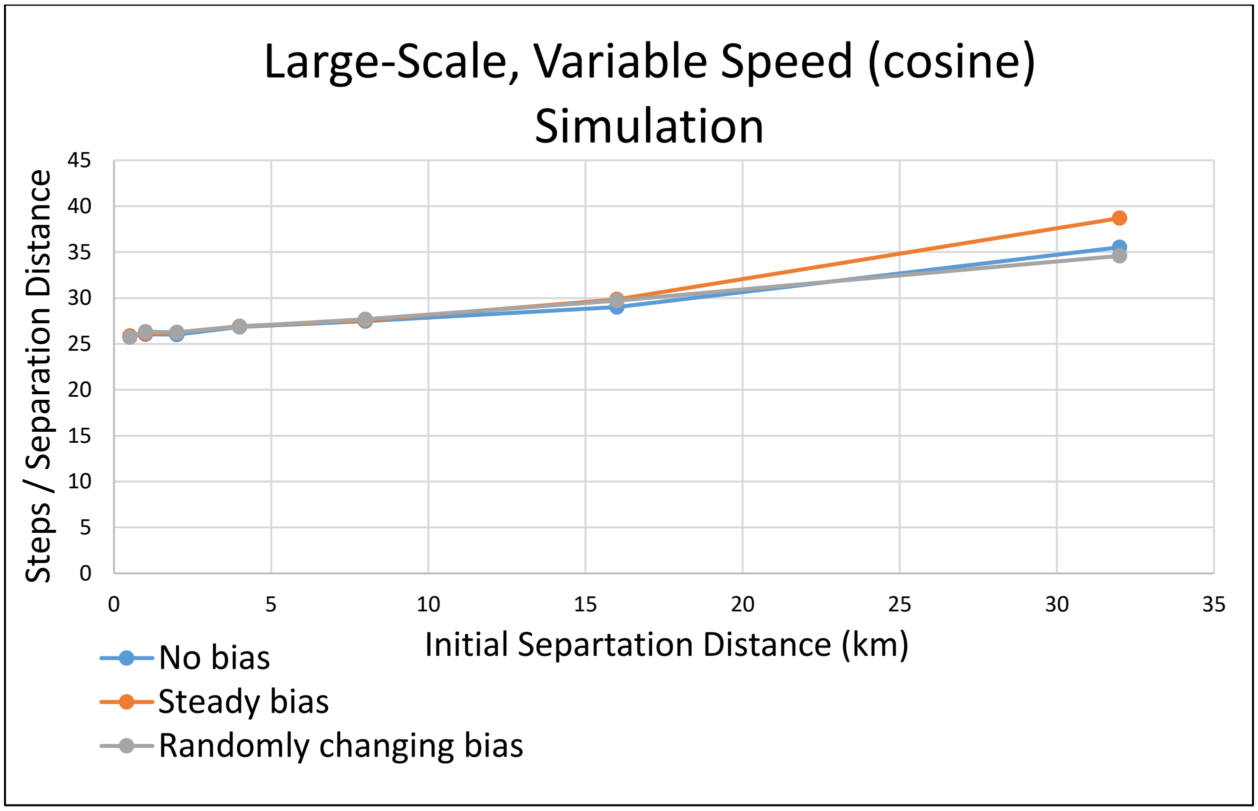

Figure 9: *Plot of the initial separation distance over the total distance traveled plotted versus the initial separation distance. This plot shows the relationship between initial separation distance and the time it takes the simulated, large-scale watercraft to reach a randomly moving target. The blue data points represent the runs with no bias on either object, the orange data points represent the runs with a constant bias throughout the runs, and the grey data points represent the runs with a bias that is randomly changing in bias and direction and we assume the watercraft makes turns instantaneously, and does not need to slow down when changing direction. Each data point represents the average of 100 runs. As we expected, the number of steps (which is analogous to time) it takes the simulated small-scale watercraft to reach its target increases with distance. However, it is not expected that these the number of steps per initial separation distance also grow. The points are connected with straight lines, which are in place to aid the eye.*

We compared the relationship between the numbers of steps needed for the simulated watercraft to reach the simulated target object when there is a bias applied to the target object versus when there is no bias applied. As expected, when a bias is applied, it takes the watercraft more steps to reach the target object, as seen in Fig. 10. We examined the situation where the speed of the watercraft was variable with the angle at which the watercraft must turn and with a bias that is at 90 degrees to the initial separation distance (for example, if the watercraft is traveling east to reach the target object, the bias is pointed north).

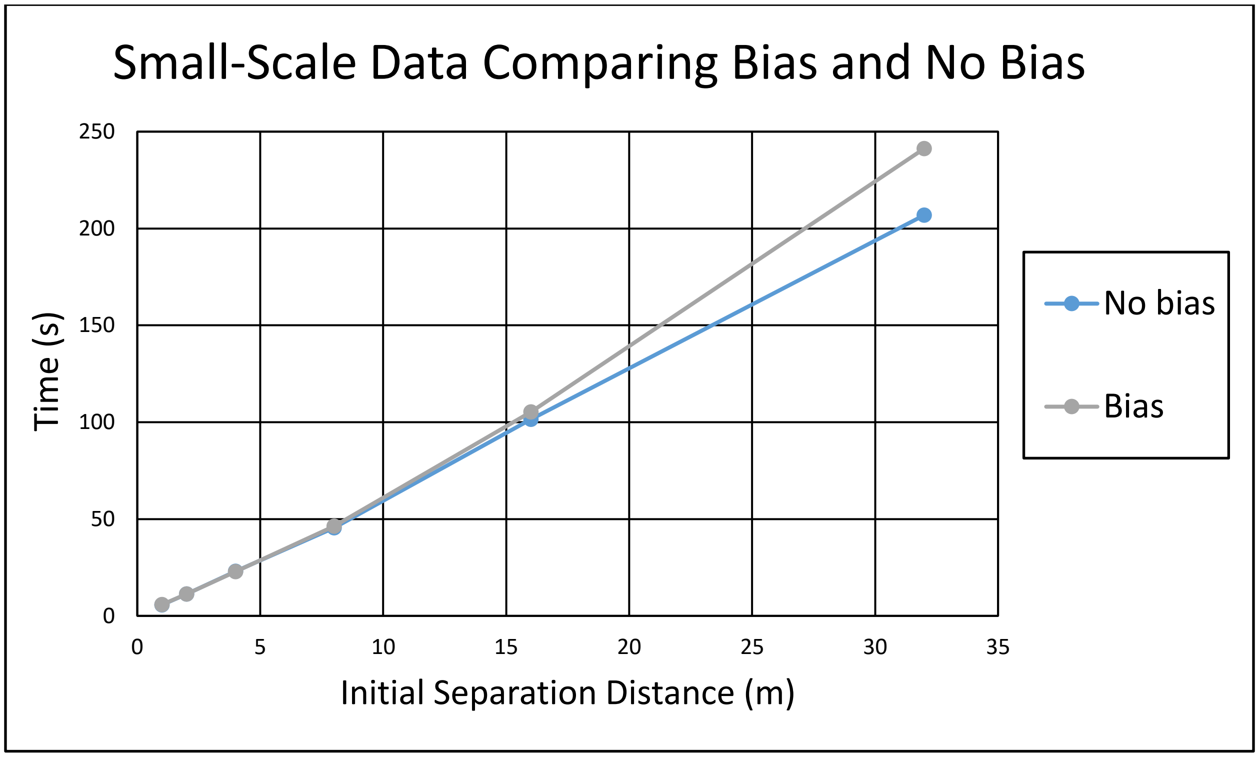

Figure 10: *The time it takes a simulated small-scale watercraft to reach the target object when a bias is applied to the target object and when there is no bias applied. When there is no bias applied, the data points are in blue, when there is a bias applied, the data points are grey. At shorter distances there is very little difference between the time it takes the watercraft to reach the target object with the two conditions, but as the initial separation increases, it becomes clear that it takes longer for the watercraft to reach the target when there is a bias applied to the target. Each data point represents the average of 100 runs. Please, note that the lines between data points are in place to aid the eye in following the data, and the lines do not represent trend lines or interpolations.*

Discussion

There is still a significant amount of research to be done to advance this project. More simulations need to be run in order to fully understand the possible paths that an autonomous watercraft might take and, thus, how quickly an autonomous watercraft can reach a target.

We have clearly determined that more steps and, thus, more time is needed to reach a target that is initially farther away, which is quite obvious. We have also found that many of the same trends that are seen in the simulations that model small-scale tests are also seen in simulations that model large-scale tests. We have also found that when there is a steady bias introduced to a system, more steps and more time are needed to reach the target object. This is true for both the large-scale simulation and the small-scale simulation. It does not appear as though the bias has any significant effect on the efficiency of the watercraft reaching the target. However, not statistical tests have examined this, and more work needs to be done to see how bias alters the simulation.

It is peculiar that the steps per initial separation distance increase exponentially as the initial separation increases in the simulations with watercraft step sizes that vary based upon the angle that the watercraft must turn. This could potentially be due to the increased number of corrections that the watercraft must make as the initial separation increases. Since the simulated watercraft needs to adjust more, it will need to make more steps at larger separation distances. The next step in understanding this phenomenon would be to determine if there is something that the code is doing that is not realistic and/or applicable to experimental tests.

We also look forward to testing different types of methods of varying speed as the simulated watercraft turns. We plan to run simulations where the step size is in a linear relationship with the angle that the watercraft must turn in order to move toward the target object. Using a linear varying speed would allow us to utilize different coefficients and, thus, potentially better model realistic watercraft.

After gathering data in simulations, land-based and water-based systems would be tested experimentally. A land-based system would be implemented largely to ensure the sonar system and the Arduino microcontroller are functioning properly. We will then run tests with the small-scale experimental watercraft where we will collect data at the same points as we did in the simulations. We can then examine if any of the simulated models accurately explain the behavior of the small-scale watercraft. If this is the case, we could attempt to extrapolate the simulations such that we can accurately model large-scale watercraft systems. Being able to accurately model large-scale autonomous watercraft systems would be beneficial as it has not been done often before.

When attempting to seek out a target with an experimental watercraft, we must consider how the craft will track a moving target versus a stationary object. We can also run tests of a land-based vehicle and see if the same results hold. We seek to understand if results were consistent across land vehicles and watercraft. This will allow for future experimenters to understand whether or not trends discovered in land-based tests can be applied to water vehicles. If land-based experiments can be substituted for water-based experiments, experimenters would be able to test water-based autonomous systems on land, which could potentially be more cost effective and easier.

In the future, we would also be interested in incorporating machine learning into our system. For example, the simulated watercraft could be programmed to detect trends in movement, thus seeking the target more efficiently.

Acknowledgments

I would like to thank my thesis committee, Dr. William Roach (committee chair), Dr. Crystal Moorman, and Dr. Nancy Cowden for their assistance and feedback. I would also like to thank the Westover Honors College, the Lynchburg College of Arts and Sciences, and the Department of Physics.

Appendix

This appendix contains the code used to run the simulations. Page 23 contains the code for the constant speed simulations. Page 24 contains the code for the variable speed simulation.

```
In [1]: %pylab
        import numpy as np # import necessary libraries
        import matplotlib.pyplot as plt
        from __future__ import division, print_function
        import random as r # random number generator
```

```
Using matplotlib backend: Qt4Agg
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: loop = 100 # number of loops being run
        j = 0 # loop counter
        count = 0 # count of total steps during simulation

        while j < loop:
            sep = 16 # initial separation distance between target and rescue craft (m)
            so = .1 # speed of target object (m/s) (fast swim or slow rowwing = 2 m/s, use .1 m/s for small scale)
            sw = .2 # speed of rescue craft (m/s) (Medium Endurance Cutter = 8 m/s, use .2 m/s for small scale)

            theta0 = 0 # initial orientation of the rescue craft (radians)
            dmo = .1 # magnitude of target bias (m/s) (use .01 m/s for small scale, .1 m/s for large scale)
            dao = np.pi/2 # angle of target bias (radians) (Set to pi/2)
            dmw = 0 # magnitude of rescue craft bias (m/s)
            daw = 0 # angle of rescue craft bias (radians)

            i = 1 # initialize the counter
            xo = sep # initial x coordinate of the target
            yo = 0 # initial y coordinate of the target
            xw = 0 # initial x coordinate of the rescue craft
            yw = 0 # initial y coordinate of the rescue craft

            position_o = [] # empty array of position of the target
            position_w = [] # empty array of the position of the rescue craft

            # each below iteration represents 1 second passing in time
            # each unit of distance represents 10 m
            while abs(xo - xw) >= sw or abs(yo - yw) >= sw: # while the target and rescue craft are seperated, run simulation
                i = i + 1 # counter
                z = r.randrange(0,4) # generate random number, z, either 0, 1, 2, 3
                dao = (2*np.pi*r.randrange(1,100))/100
                dmo = (.01*r.randrange(1,100))/100
                if z == 0: # if z = 0, move in x + bias
                    xo = xo + so + dmo*np.cos(dao)
                    yo = yo + dmo*np.sin(dao)
                if z == 1: # if z = 1, move in x + bias
                    xo = xo - so + dmo*np.cos(dao)
                    yo = yo + dmo*np.sin(dao)
                if z == 2: # if z = 2, move in y + bias
                    xo = xo + dmo*np.cos(dao)
                    yo = yo + so + dmo*np.sin(dao)
                if z == 3: # if z = 3, move in y + bias
                    xo = xo + dmo*np.cos(dao)
                    yo = yo - so + dmo*np.sin(dao)
                position_o.append([xo,yo]) # update array with new position of target

                if xo - xw == 0 and yo - yw > 0: # generates angle for the rescue craft to travel towards target
                    theta1 = np.pi # case where denominator in below equation is zero, target above
                if xo - xw == 0 and yo - yw < 0:
                    theta1 = -np.pi # case where denominator in below equation is zero, target below
                else:
                    theta1 = np.arctan((yo - yw)/(xo - xw)) # all other cases

                xw = xw + sw*np.cos(theta1) + dmw*np.cos(daw) # update x position of the rescue craft + bias
                yw = yw + sw*np.sin(theta1) + dmw*np.sin(daw) # update y position of the rescue craft + bias
                position_w.append([xw,yw]) # create array of position of the rescue craft

            position_o = np.array(position_o) # create lists of data from arrays
            position_w = np.array(position_w)

            #print(position_o) # print lists
            #print(position_w)
            #print(i) # print number of steps needed to reach target

            #figure() # create a figure to plot the paths taken
            #plot(position_o[:,0],position_o[:,1])
            #plot(position_w[:,0],position_w[:,1])
            #show()
            j = j + 1
            count = count + i
        avg_step = count/loop # calculate average number of steps needed to reach target
        avg_time = (avg_step*sw)/sep # calculate average time needed to reach target (s)
        print(avg_step)
        print(avg_time)
```

```
80.81
1.010125
```

```
In [1]: %pylab
        import numpy as np # import necessary libraries
        import matplotlib.pyplot as plt
        from __future__ import division, print_function
        import random as r # random number generator

        Using matplotlib backend: Qt4Agg
        Populating the interactive namespace from numpy and matplotlib

In [2]: loop = 100 # number of loops being run
        j = 0 # loop counter
        count = 0 # count of total steps during simulation

        while j < loop:
            sep = 2 # initial separation distance between target and rescue craft (m)
            so = .1 # speed of target object (m/s) (fast swim or slow rowwing = 2 m/s, use .1 m/s for small scale)
            sw = .2 # speed of rescue craft (m/s) (Medium Endurance Cutter = 8 m/s, use .2 m/s for small scale)

            theta0 = 0 # initial orientation of the rescue craft (radians)
            dmo = .1 # magnitude of target bias (m/s) (use .01 m/s for small scale, .1 m/s for large scale)
            dao = np.pi/2 # angle of target bias (radians) (set to pi/2)
            dmw = 0 # magnitude of rescue craft bias (m/s)
            daw = 0 # angle of rescue craft bias (radians)

            i = 1 # initialize the counter
            xo = sep # initial x coordinate of the target
            yo = 0 # initial y coordinate of the target
            xw = 0 # initial x coordinate of the rescue craft
            yw = 0 # initial y coordinate of the rescue craft

            thetadiff = 0 # define as the change in angles the watercraft makes
            speedmult = 1 # define some number between 0 and 1 that the step size will be multipied by when turning

            position_o = [] # empty array of position of the target
            position_w = [] # empty array of the position of the rescue craft

            # each below iteration represents 1 second passing in time
            # each unit of distance represents 10 m
            while abs(xo - xw) >= sw or abs(yo - yw) >= sw: # while the target and rescue craft are seperated, run simulation
                i = i + 1 # counter
                z = r.randrange(0,4) # generate random number, z, either 0, 1, 2, 3
                # dao = (2*np.pi*r.randrange(1,100))/100
                # dmo = (.01*r.randrange(1,100))/100
                if z == 0: # if z = 0, move in x + bias
                    xo = xo + so + dmo*np.cos(dao)
                    yo = yo + dmo*np.sin(dao)
                if z == 1: # if z = 1, move in x + bias
                    xo = xo - so + dmo*np.cos(dao)
                    yo = yo + dmo*np.sin(dao)
                if z == 2: # if z = 2, move in y + bias
                    xo = xo + dmo*np.cos(dao)
                    yo = yo + so + dmo*np.sin(dao)
                if z == 3: # if z = 3, move in y + bias
                    xo = xo + dmo*np.cos(dao)
                    yo = yo - so + dmo*np.sin(dao)
                position_o.append([xo,yo]) # update array with new position of target

                if xo - xw == 0 and yo - yw > 0: # generates angle for the rescue craft to travel towards target
                    theta1 = np.pi # case where denominator in below equation is zero, target above
                if xo - xw == 0 and yo - yw < 0:
                    theta1 = -np.pi # case where denominator in below equation is zero, target below
                else:
                    theta1 = np.arctan((yo - yw)/(xo - xw)) # all other cases

                thetadiff = abs(theta1 - theta0) # find the turning angle
                if thetadiff <= np.pi: # find speed multiplier
                    speedmult = np.cos(thetadiff)
                if thetadiff >= np.pi: # if the turning angle is greater than pi, the step will be used solely to turn, no movement
                    speedmult = 0

                xw = speedmult*xw + sw*np.cos(theta1) + dmw*np.cos(daw) # update x position of the rescue craft + bias
                yw = speedmult*yw + sw*np.sin(theta1) + dmw*np.sin(daw) # update y position of the rescue craft + bias
                position_w.append([xw,yw]) # create array of position of the rescue craft

                theta0 = theta1

            position_o = np.array(position_o) # create lists of data from arrays
            position_w = np.array(position_w)

            #print(position_o) # print lists
            #print(position_w)
            #print(i) # print number of steps needed to reach target

            #figure() # create a figure to plot the paths taken
            #plot(position_o[:,0],position_o[:,1])
            #plot(position_w[:,0],position_w[:,1])
            #show()

            j = j + 1
            count = count + i
        avg_step = count/loop # calculate average number of steps needed to reach target
        avg_time = (avg_step*sw)/sep # calculate average time needed to reach target (s)
        print(avg_step)
        print(avg_time)

        17.19
        1.719
```

References

1 Juan Pimentel and Jennifer Bastiaan, "Characterizing the Safety of Self-Driving Vehicles: A Fault Containment Protocol for Functionality Involving Vehicle Detection," Vehicular Electronics and Safety (ICVES) 2018 IEEE International Conference , 1-7 (2018).

2 Umberto Papa, *Embedded Platforms for UAS Landing Path and Obstacle Detection: Integration and Development of Unmanned Aircraft Systems (Studies in Systems, Decision and Control)*, edited by Anonymous 1st ed. (Springer, 2018), pp. 112.

3 Umberto Papa and Giuseppe Del Core, "Design of sonar sensor model for safe landing of an UAV," 2015 IEEE Metrology for Aerospace , (2015).

4 Wei Wang, Luis A. Mateos, Shinkyu Park, Pietro Leoni, Banti Gheneti, Fabio Duarte, Carlo Ratti, Daniela Rus, "Design, Modeling, and Nonlinear Model Predictive Tracking Control of a Novel Autonomous Surface Vehicle," SENSEABLE CITY LAB , (2018).

5 Noel Sharkey, "Automating warfare: lessons learned from drones," Journal of Law, Information and Science 21 (2), 116-23 (2011).

6 Mohammad Y. M. Ibrahim and Lukman Audah, "Real-time bus location monitoring using Arduino," AIP Conference Proceedings 1883, (2017).

7 S. Ilgin Guler, Monica Menendez and Linus Meier, "Using connected vehicle technology to improve the efficiency of intersections," Transportation Research C (46), 121-10 (2014).

8 Chuck Husick, "The button that will save your life the Coast Guard's new Rrscue 21 system brings 911 to the seas," Yachting 199 (3), 28 (2006).

9 Larry A. Young, "Small Autonomous Air/Sea System Concepts for Coast Guard Missions," U.S. Coast Guard Maritime Domain Awareness Requirements, Capabilities, & Technology Forum.

10 A. Pedro Aguiar and João P. Hespanha, "Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty," IEEE Transactions on Automatic Control 52 (8), 1362-1379 (2007).

11 Ricardo Silveira Rodrigues, Marcia Pasin and Renato Machado, "Indoor position tracking: An application using the Arduino mobile platform," IFIP Wireless and Mobile Networking Conference 10, (2017).

12 O. M. Amin, M. A. Karim and A. H. Saad, "Development of a highly maneuverable unmanned underwater vehicle on the basis of quad-copter dynamics," AIP Conference Proceedings (1919), (2017).

13 L. Du, H. Liu, Z. Bao, J. Zhang, "Radar automatic target recognition using complex high resolution range profiles," IET RADAR SONAR NAV 1 (1), (2007).

14 Eric W. Weisstein, "Random Walk--2-Dimensional," Wolfram MathWorld 2019 (May/2019), 1 (2019).

15 United States Coast Guard, "The Cutters, Boats and Aircraft of the U.S. Coast Guard," , 127 25 (2016).

16 Martin Truijens and Huub Toussaint, "Biomechanical aspects of peak performance in human swimming," BRILL 55 (1), 17–40 (2005).

17 Umberto Papa and Giuseppe Del Core , "Design of sonar sensor model for safe landing of an UAV," 2015 IEEE Metrology for Aerospace , 346-4 (2015).