Laser Range Finder Mapping of Floating Vehicle

By

Corinna Hui

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009

Author ....
Department of Mechanical Engineering
May 8, 2009

Certified by.. [illegible]
Professor Franz S. Hover
Assistant Professor
Thesis Supervisor

Accepted by .... [illegible]
Professor J. Lienhard V
Collins Professor of Mechanical Engineering
Chairman, Undergraduate Thesis Committee

Laser Range Finder Mapping of Floating Vehicle

by

Corinna Hui

Submitted to the Department of Mechanical Engineering
on May 8, 2009 in Partial Fulfillment of the
Requirements for the Degree of Bachelor of Science in
Mechanical Engineering

ABSTRACT

Using laser range finders as a method of navigation is popular with mobile land robots; however, there has been little research using it with water vehicles. Therefore, this thesis explores the usage and data flow of a laser range finder on a water raft. A unique algorithm for localization and mapping for the sensor is developed and tested both in simulation and in real-time with a vehicle. Both the localization of the vehicle and mapping of its environment are able to achieve precise locations, deviating only a few millimeters of their expected values. With this algorithm, a closed-loop control system is also developed and implemented on the vehicle. The vehicle is able to move to a predefined location and be within a very small range of acceptable values. The control loop is further explored with damping, gain variations, and different trajectories.

Thesis Supervisor: Franz S. Hover
Title: Assistant Professor of Mechanical Engineering

Acknowledgements

My undergraduate years at MIT have been an exciting rollercoaster ride, but I would not have had such a great experience without many people. I am grateful to my advisor, Professor Franz Hover, for both giving me an opportunity to do research with him and for his supervision during this work. Additionally, I would like to thank Brendan Englot for helping me get started with the thesis, understand how the vehicle works, and how to successfully control it.

Outside of thesis and undergraduate work, I participated in the MIT Sport Taekwondo club for four years. I am forever grateful to the people I have met and befriended through the club. I have grown to be the person I am through this club. I would not been able to survive the daily grind from MIT if I did not have a place that I could always run to and escape from stress. I am appreciative of the club instructors for watching over my growth in taekwondo and helping me whenever I need it. I am especially thankful to Master Dan Chuang for teaching me taekwondo, coaching me, and being an inspirational figure.

Lastly, I am always thankful to my family and friends. My parents have always encouraged me to do my best and gave me endless support when the coursework got tough. My friends have stood by me for the past four years and provided help whenever I need it. MIT has been both a wonderful and stressful experience, but it is something I do not regret doing. I look forward to using what I learned here at MIT, meeting new people, and learning more at Stanford University graduate school.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

This thesis focuses on the mapping and navigation functions of a small floating vehicle using a laser-range finder. The range finder, in general, is especially useful for robotic navigation and 3-D modeling because of its high precision scanning abilities. Previous work has been done on this type of sensor, mainly with using mobile robots in the interest of mapping their environment or navigating around a room. For instance, Hakan Telmeltas and Deniz Kavak released a paper, called "SLAM FOR Robot Navigation," about two different techniques for Simultaneous Localization and Mapping (SLAM) for mobile robot navigation. They compare Extended Kalman Filter-based with Compressed Extended Filter-based SLAM, which are both very good and famous algorithms. Although the aforementioned techniques are well-known, other researchers have developed other computational methods. Hee Jin Son and Byung Kook Kim, in their paper "An Efficient Localization Algorithm Based on Vector Matching for Mobile Robots Using Laser Range Finders," studied a new computational technique as the title explains. Although there are numerous theses and papers on this sort of navigation, using a laser range finder on water vehicles or rafts is not as widely researched. Michael A. Kokko, an MIT Master of Science graduate in 2007, has previously done research on this topic and has written about it in his thesis "Range-based Navigation of AUVs Operating Near Ship Hulls." The laser and vehicle combination used in this paper was actually developed by him. Therefore, the main objectives of this paper are to first understand the laser range finder and second, use it effectively as a method of navigation for an autonomous water vehicle.

Chapter 1 begins the thesis with background information of the vehicle and small laser sensor. Chapter 2 discusses the data flow from the laser and interprets each data point. Chapter 3 provides a platform on the range-based algorithm for sensor localization, specifically finding the sensor's X-Y trajectory during an experiment. Chapter 4 details the algorithm for environment mapping, in which the raft is able to feedback to the user where certain objects lie. Chapter 5 explores controlling the vehicle so that it remains in its position, despite an external force attempting to move it. In other words, although a user displaces the raft, it should be able to return to its original position automatically.

1.1 Vehicle Background

The floating vehicle is a simple raft that was developed by Kokko and contains four thrusters and a laser scanner on top. The thruster are positioned at each corner and orientated so that the thrust is directed off to the side, rather than straight out. The laser range finder is orientated along the 1-3 diagonal line. Because of thrusters' orientations, the surge direction (forward) is along that diagonal.

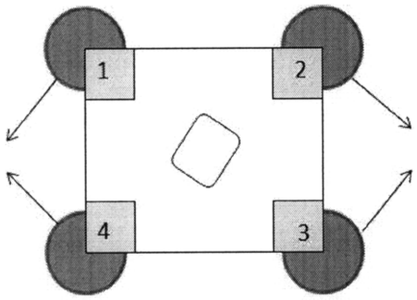


Figure 1-1. Representation of the raft. There are four thrusters, represented by the blue circles, at each of the corners. The laser sensor, the rounded circle, is located on the top. The arrows point to the directions of each thruster.

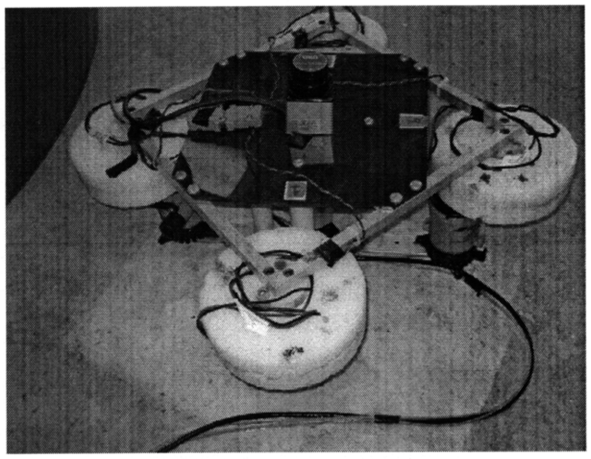


Figure 1-2. A photograph of the raft. The thrusters are located below the white foam, and the laser is the blue circular object on top.

To power the motors and make the vehicle move, a tether is connected to the motors. In order to control the motors in MATLAB, a series of steps and commands are needed to be performed as listed below in Table 1-1. The motors require at most twelve volts and approximately one-tenth of an amp to power. When commanded correctly, the raft can move forwards, backwards, left, and right without difficulty, in respect to its surge direction. In more complicated code, it can travel in curve-like trajectories, such as a sinusoid.

MATLAB Code	**Comment**
	Turn on power supply (12V, 0.1 A)
r = serial('COMN', 'BaudRate', 9600)	Creates object for laser scan N refers to the COM port on Computer
fopen(r)	Opens communication
fprintf(r,'k')	Sends this to the device
fscanf(r)	Checks if motor receives command !! means yes
	Plug in motor battery
fprintf(r, 'm1f300')	Sends command to motors "m1" is motor number "f" is forward ("r" is reverse) "300" is speed (max 500)
fprintf(r, 'k')	Stops motors
fprintf(r, ' ')	Hold and unhold

Table 1-1: MATLAB code in order to start the thrusters and move the raft

In the command line that determines the speed of the thrusters, "500" refers to 12 volts, while "0" refers to zero volts. All the speeds in between are proportional; the speed determines how much voltage is flowing to the motors. In order to command the vehicle to move forward in the surge direction, entering the commands "fprintf(r, ' ')","fprintf(r, 'm1f500')", "fprintf(r, 'm4f500')", and "fprintf(r, ' ') sequentially. With a manipulation of a pair of thrusters (such as #2 and #3), the vehicle can move to the right or left.

1.2 Laser Range Finder Background

The laser range finder for this raft is a Hokuyo URG laser scanner. Located at the center of the vehicle, it uses an infrared laser with a wavelength of 785 nm. Its physical dimensions are 50 mm wide, 50 mm long, and 70 mm tall. The laser finder is easily hooked up to the computer using a firewire with a USB port or a RS232C. In addition, the Hokuyo laser requires a power supply of five volts.

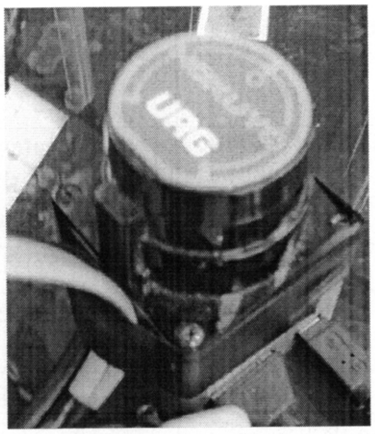


Figure 1-3. Photograph of Hokuyo laser range finder used for experiments

The range finder is able to collect distance measurements within its 270 degree scan area with a minimum of 20 mm and a maximum of 4000 mm radius. At 4000 mm, the laser beam has a maximum divergence of 40 mm. Furthermore, it outputs data taken in a total 768 "bins," where each bin is approximately 0.3516 degree. However, only between bins 44 and 725 does real data output exist; the ends are considered dead zones. Relative to the sensor's front axis, bin 0 represents the sensor angle at -135°, while bin 768 bin is at +135°. Bin 384 is the sensor's center line at 0°. Data is taken counterclockwise; therefore, bin #1 refers to the measurement made to the right of the sensor, while bin #683 is to the left. Each scan takes approximately 100 milliseconds.

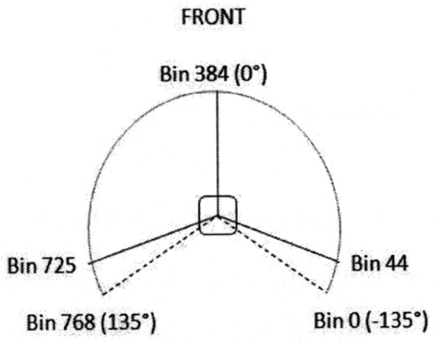


Figure 1-4. Scan area of the Hokuyo laser. The measureable area is between bins 44 to 725, and zero degree is represented at bin 384.

When the laser records the data obtained from its environment, its language is not in Cartesian coordinates that the user can easily read and understand. Its data is expressed in 12 bits and encoded to 2-characters in the 4095mm mode or in 18 bits and encoded to 3 characters in 5600 mm mode. Additionally, the 6-bits binary data is then converted to 1-byte character codes. Using two MATLAB m-files, this data is turned into Cartesian coordinates and kept into arrays. These arrays are easily accessible by the user, and MATLAB can plot the data.

To control the laser, a series of commands and steps must be performed in MATLAB. These commands are listed below in order of sequence. Once the series of code is entered, the laser continually takes data until it is told to be stopped.

MATLAB Code	**Comment**
s = serial('COMN', 'BaudRate', 115200)	Creates object for laser scan N refers to the COM port on Computer

set(s, 'InputBufferSize', 2049)
set(s, 'Terminator', 'LF')	

fopen(s)	Opens communication
	Turn on power supply (~5V, 0.5A)

fclosef(s);
delete(s);
	Turns laser off/
	Closes communication

Table 1-2: MATLAB code in order to start communication with the laser

Chapter 2: Data Flow from Laser

Experiments are first run in order to understand the data flow from the sensor better. The experiments are performed in a large tub filled with water. The raft moves around in the tube with a tether that hangs from the ceiling, which is positioned so that the raft is not dragged down by the weight of the wire or become tangled. Four poles stand at the edge of the tub, strapped against a horizontal L-shaped bar to keep them steady and stable; the middle two poles are linked together, and the unit is referred to as pole #2. These poles are ABS tubes with a diameter of approximately two inches.

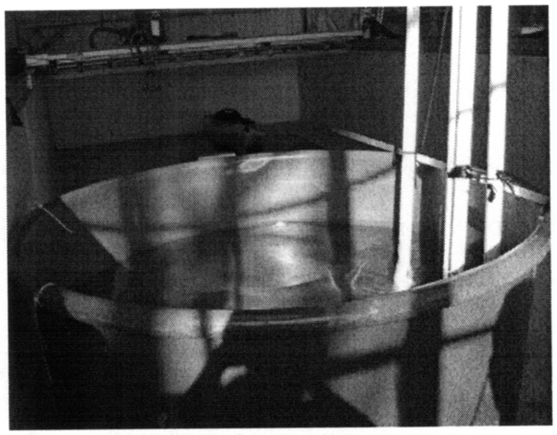


Figure 2-1. Experiments were performed in a large blue tub with white poles.

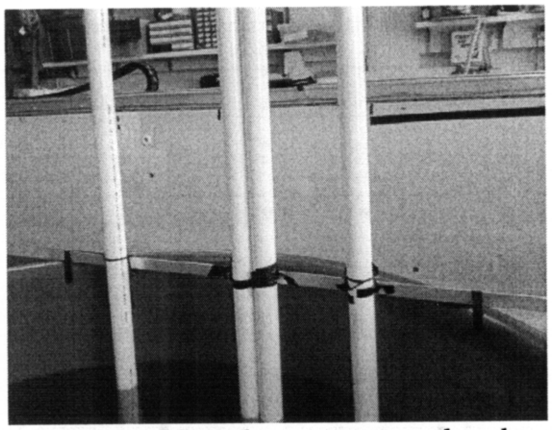


Figure 2-2. Poles strapped on in the tub. The poles are orientated so that the middle poles are closer to one side than other to help with localization.

Each experiment is set to run for fifty time steps, and each scan is taken every time step. The equivalent of a time step in real time is approximately between 0.5 to 1 second, depending on the thrusters' speed. The user designates a force vector that is sent to the thrusters, which drives the raft to move in a certain trajectory. The data obtained from the laser is translated into an array of Cartesian data points. Plotting the data array in MATLAB yields a graph that contains the bin number in the x-axis and distance in millimeters in the y-axis.

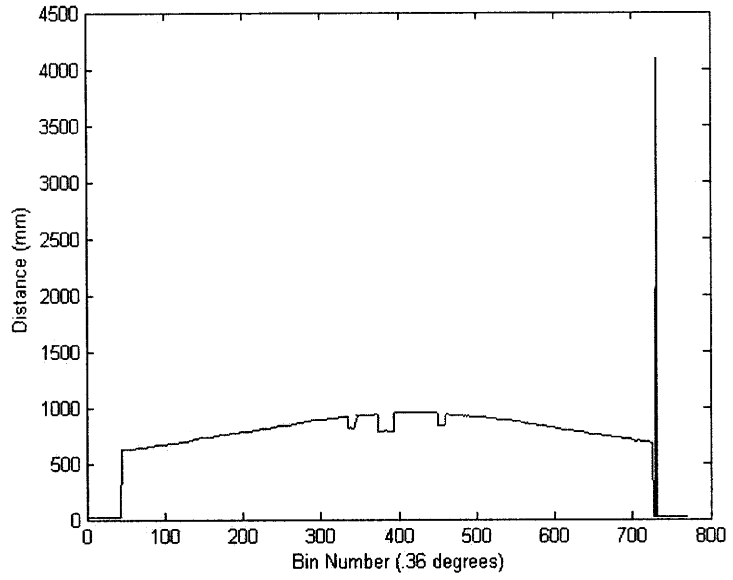


Figure 2-3. Sample graph of laser scan. The rectangular dips are the poles located in the tub. The curved slope is the tub's wall; the ends are considered dead zones.

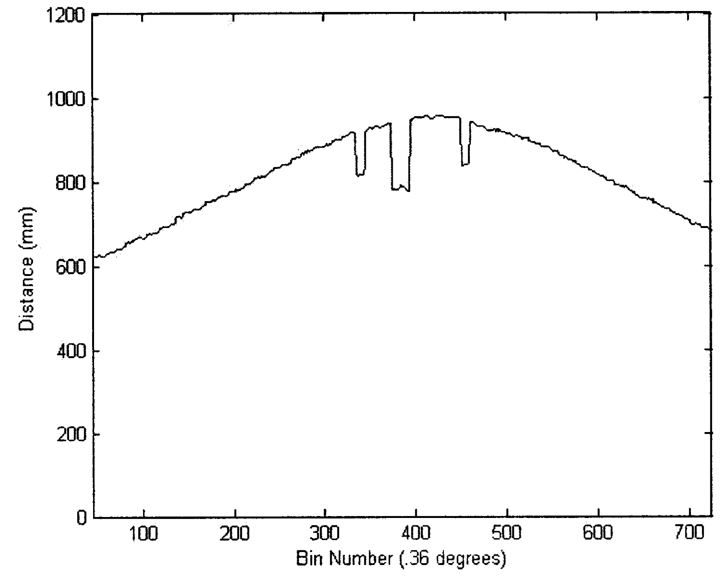


Figure 2-4. Sample graph of same laser scan in Figure 2-3. The plot has been modified to remove the dead zone. This is the range of data used for calculations.

Although the laser is able to display the tub and poles, the scans are relatively noisy, especially those of the tub walls. Despite the even appearance of the walls, the laser detects small surface irregularities, which result in slight fluctuations seen along the curve of the graph. Another example of the laser's sensitivity to subtle changes in contour can be seen in the scan of the two poles joined together in the middle. At first, the laser seems to register the tubes as a single unit, however, a closer inspection of the graph reveals a tiny elevation, indicating the presence of two tubes, rather than one. Nevertheless, the laser's ability to differentiate between the two poles is poor. Therefore, while the laser has the capability to recognize the existence of the tubes, its lack of precision limits its ability to generate a map representative of its surroundings.

When all fifty time steps curves are compiled together into a mesh plot, the real-time trajectory of the raft can be discerned through the undulations of the graphs.

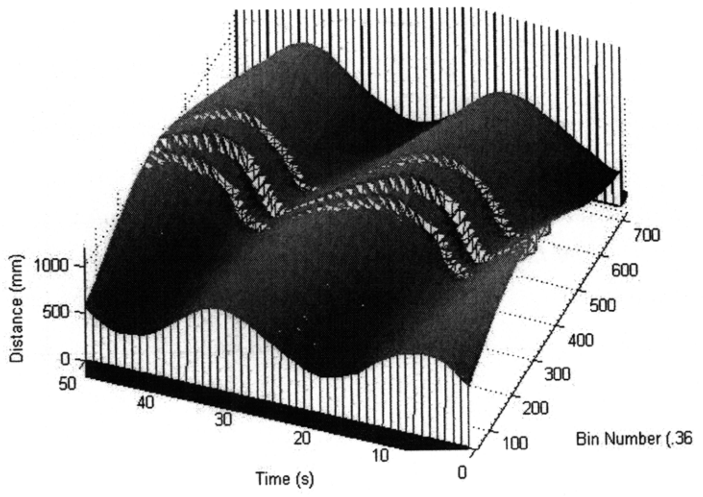


Figure 2-5. Mesh plot of laser scan during an experiment. The three stripped lines are where the laser sees the poles.

Chapter 3: Localization

In order to calculate the vehicle position of the raft in the X-Y plane, a series of steps are implemented in MATLAB. Sections 3.1 and 3.2 present algorithms for the Hokuyo sensor on how to determine the vehicle's location and yaw in its environment. Accuracy of the vehicle's position is greater when comparing poles far apart; therefore, the calculations are made using poles #1 and #3.

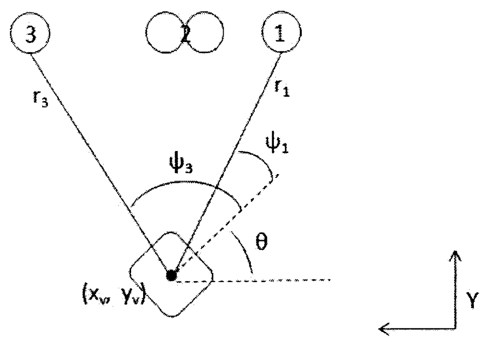


Figure 3-1. Representation of the poles' locations. Because the laser collects data counterclockwise, the X-Y origin is arbitrarily set at the bottom right of the tub.

3.1 Vehicle Position Calculation

1. For each time step *k*, read the vector *r*, which contains all the bins' measurements.

2. From bin number 47 to 723, for each bin number *n*, find the slope from *n* to (*n*+2). The range of data is limited between 47 and 723 is confirm that the only points obtained are within the laser's scan area.

3. The slope of distance and bin number is usually relatively small; therefore, if the slope is to ever exceed a certain threshold, the sensor reads an object in its scan area. The slope threshold is set to 25 mm/bin number, which signals a pole present. Since the threshold is very small, any extraneous objects in the scan area will be registered as a pole. Therefore, in the best case, the laser should only see poles and nothing else.

4. For each *n* that has an absolute slope greater or equaled to 25 mm/bin number, add *n* to single column matrix [*poles*]. The positive slope represents a change in distance where surface moves farther away, while a negative slope is when the surface becomes closer.

5. Due to the way the poles are found mathematically, [*poles*] may include consecutive bin numbers. Therefore, loop through the array and discern whether the array index *i* is consecutive with *i*±1 and *i*±2. If so, replace *i* with zero, and after the loop is completed, remove all zeros from the array.

6. Store the average of each pair in [*poles*] (1 & 2, 3 & 4, and 5 & 6) as b. Each b is the bin number where the poles exist.

7. Calculate for the angle (φ) of each pole relative to the laser. Equation below yields φ in degrees.

$$\varphi = b * \frac{270}{768} - 135 \tag{3.1.1}$$

8. Compute for distance measurement for each pole through $R = r[b]$.

9. Solve the vehicle's position (x_v, y_v) using Pythagorean Theorem and the poles 1 and 3's distance measurements (R_1 and R_3, respectively). Additionally, the origin where the poles' real locations are taken from is arbitrarily set by the user. Using systems of equations, x_v and y_v are calculated. When computing in MATLAB, the solve function yields two values for y_v. The equations do not determine whether the vehicle is in front or behind the poles. Therefore, in order to retrieve the correct position, choose the smaller value.

$$R_1{}^2 = (x_v - x_1)^2 + (y_v - y_1)^2 \tag{3.1.2}$$

$$R_3{}^2 = (x_v - x_3)^2 + (y_v - y_3)^2 \tag{3.1.3}$$

3.2 Yaw (Heading) Calculation

1. For each time step k, obtain the respective x_v and y_v values.

2. Determine the yaw angle (in radians) by choosing one of the pole's coordinates and angle. Because the calculations are used for absolute values, there is a conditional sequence for which equation to use. If vehicle's x-position is greater than or equaled to pole #1's x-value, proceed with Equation 3.2.1. If it is of lesser value, proceed with Equation 3.2.2. Similar conclusions can be drawn for pole #3.

$$\theta = atan\left(\left|\frac{y_1 - y_v}{x_1 - x_v}\right|\right) - \varphi_1 \tag{3.2.1}$$

$$\theta = \pi - atan\left(\left|\frac{y_1 - y_v}{x_1 - x_v}\right|\right) - \varphi_1 \tag{3.2.2}$$

3.3 Localization Results

Using the calculations listed in Sections 3.1 and 3.2, the vehicle's position in a complete experiment run can be graphed as a function of time step with rotation of the vehicle icon, representing the heading. Using the same format as previous experiments for data flow were conducted, the raft was made to travel in a circle. The raft itself cannot run in a perfect circle by itself in open loop, therefore it was pushed around by hand. No torque was provided, so the sensor faced forward for the duration of the experiment.

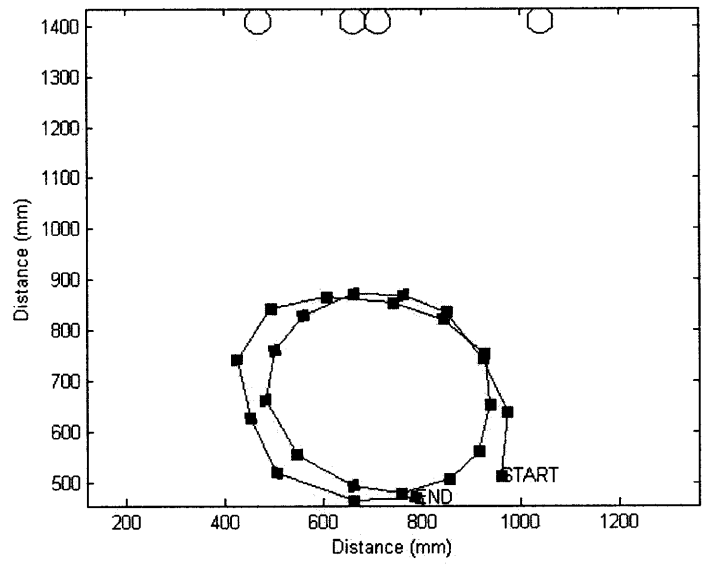


Figure 3-2. Laser sensor X-Y trajectory as the rafts moves in a circle at every other time step. The four poles at the top represent the locations of the poles in the actual environment. The far left circle is the position of the first pole. The blue squares designate the location and direction of the sensor.

To an extent, the sensor is able to determine its location in the tub. The motion of the laser resembles a circle with almost smooth curves. The curvature is entirely smooth because the laser scan itself is noisy and out of fifty scans, only twenty-five are presented. Additionally, it is unknown whether the vehicle traveled in a perfect circle since it was moved by hand, which may affect the shape of the plot. As a consequence of both noise and human error, the accuracy of the localization map is decreased. The rotation of each of the blue square is the heading of the vehicle. The rotation of the icons seems to be in parallel with the direction of the sensor. Some are more rotated than others due to, again, noise and human error. As mentioned before, the vehicle was translated around in a circle with the sensor facing forward (ninety degrees above the +x axis) during the experiment run. Because it was moved by hand, the angle of the sensor may not stay exactly be ninety degrees at each scan, as seen below in Figure 3-3. Nevertheless, the heading stayed within 10 degrees of what was expected.

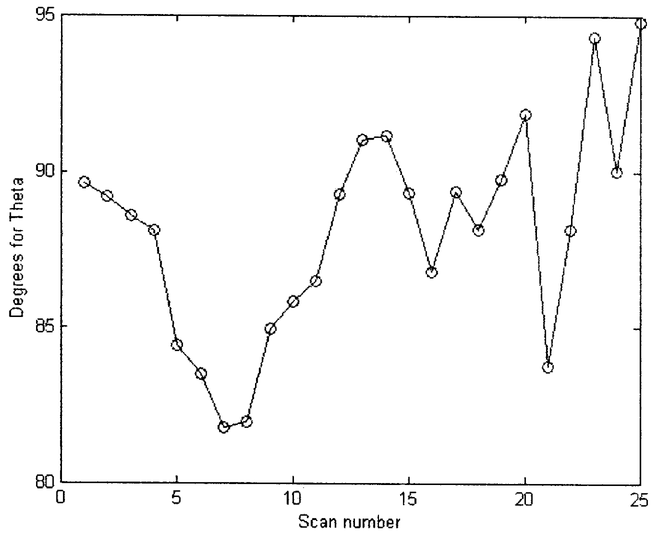


Figure 3-3. Variations of the vehicle's heading as a function of every other scan number, using Equation 3.2.1

Chapter 4: Mapping of Environment

After determining where the raft is located in its environment and using simple geometry, it should be able to use that information to map out where its surroundings are.

4.1 Mapping Algorithm

1. Obtain the values for x_v, y_v, θ, and respective R and φ in each time step k.
2. For k, calculate the position of pole 1 (x_1, y_1). These equations are applicable regardless of whether $\varphi_1$is negative or positive.

$$x_1 = x_v - R_1 * \cos(\varphi_1 + \theta) \qquad (4.1.1)$$

$$y_1 = x_v + R_1 * \sin(\varphi_1 + \theta) \qquad (4.1.2)$$

3. To determine the position of pole 3 (x_3, y_3), similar equations are used, as follows.

$$x_3 = x_v - R_3 * \cos(\varphi_3 + \theta) \tag{4.1.3}$$

$$y_3 = x_v + R_3 * \sin(\varphi_3 + \theta) \tag{4.1.4}$$

4.2 Mapping Results

Using the same format and experiment conditions as before, the algorithms in Chapters 3 and 4 yielded the vehicle's position within the tub and then mapped out the poles in MATLAB. The calculations in Chapter 3 use the center of the poles' surfaces to calculation the vehicle's path; therefore, the mapping computations in this section only map out where the vehicle sees the center of the poles is. More in-depth calculations can provide a realistic 2-D view of the poles, as seen in the laser scans.

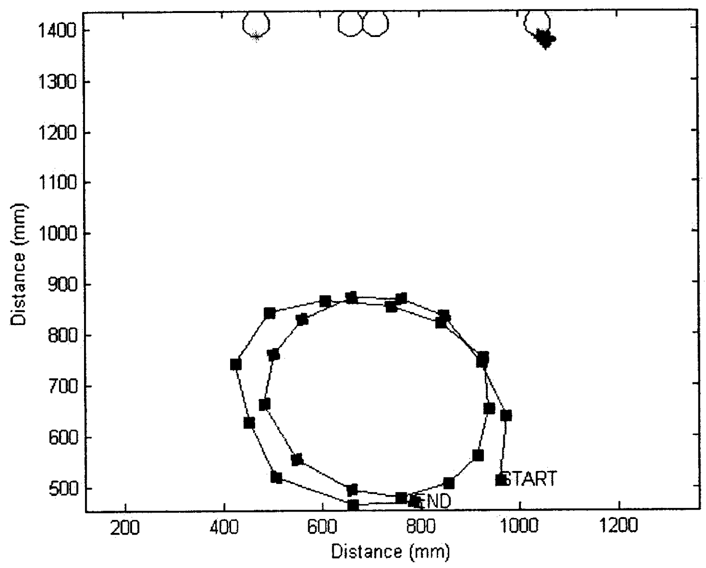


Figure 4-1. Mapping of the poles using pole 1 as a reference point. The green asterisks are pole #1's locations, while the red pluses are pole #3's. START designates the beginning of the run, and END the end.

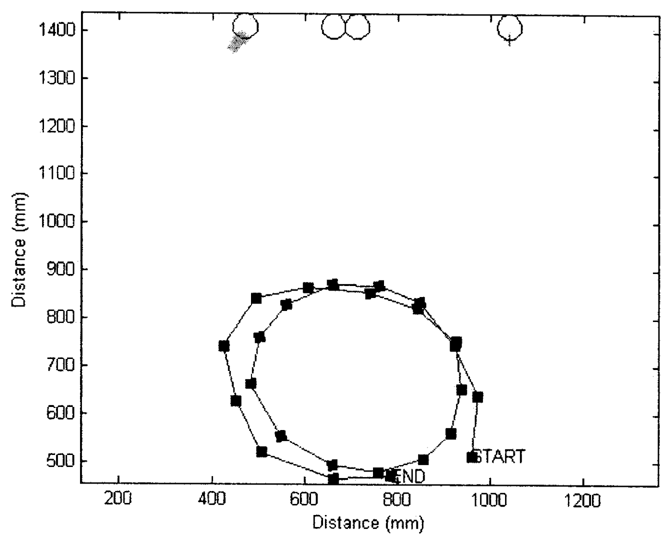


Figure 4-2. Mapping of the poles using pole 3 as a reference point.

Both graphs display a similar vehicle trajectory; there are no major discrepancies between the locations of the data points. Despite whichever reference point the algorithm uses, the mapping of the two poles are within an acceptable range. For example, in Figure 4-2, the range of the pole's x-values is 445 mm to 470 mm when the actual x-location is 469.9 mm; y-range is 1366 mm to 1385 mm while the real y is 1384.3 mm. In both axes, the deviation is at most 20 to 25 mm. Depending on which pole was used for the calculations, that particular pole's mapping is very precise, which is why one can see one plus or asterisk in either figures. Therefore, there is no favoritism towards using one particular side over the other. Like in Chapter 3.3, the mapping of the vehicle's position is not a perfect circle, which results not completely precise poles' locations. The noise from the laser scan also makes it difficult to plot perfectly. However, in the instance that the vehicle is sitting mostly still, the points pinpoint the center of the first tube. Moreover, the precision is even higher than while the vehicle is in motion.

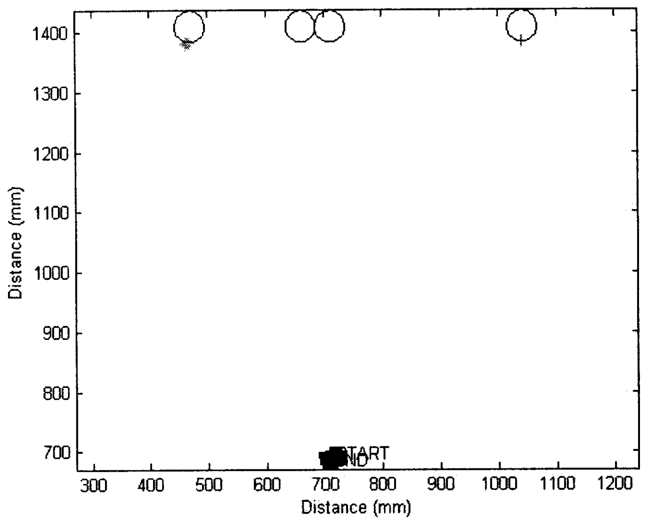


Figure 4-3. Plot of the poles with the vehicle is standing still during an experiment run.

Chapter 5: Controlling the Raft

Controlling the raft is commanding it to return to a specific location in the tub. In the first few control experiments, the closed loop was broken up into three parts – yaw, sway, and surge. Rather than making the vehicle move diagonally, it first orientates itself in the proper heading, move towards the desired position vertically, and then horizontally. In this case, the proper heading is when the vehicle faces forward; θ is ninety degrees (facing the poles). Because it is difficult to pinpoint exactly ninety degrees or precisely the (x, y) position, there are buffers for all three directions. In particular, there is about ± 10 degrees for the heading and ± 15 mm in the x-y direction. These ranges were chosen after experiments show that the vehicle is able to move to those numbers. When the vehicle is within a range, the control loop for that direction is completed until it moves away from the desired coordinates. Afterwards, all three loops are combined into one closed-loop, which both moves and rotates at the same time. In another control system, a damping function is implemented on the heading vector. In the last section of this chapter, the sway direction is controlled in a way that the vehicle travels a square-wave trajectory.

5.1 Three Individual Control Loops

Each loop was first developed individually through experimentation, such as what kind of input causes the thrusters to move in a certain direction. After a better understanding of the force input, the closed loops were performed sequentially.

5.1.1 Implementation of Each Loop

In order for the vehicle to orientate itself, it first needs to scan its surrounding environment to determine what are its heading and local position. Afterwards, it goes through three different while-loops, each for its respective direction. It determines its location and the difference between where it is and where it wants to be. Each time, it will increment a little towards its desired destination until it reaches there and the loops end.

To first start off the loops, the x, y differences, and heading of the desired location are arbitrarily set. The MATLAB program proceeds to continue through the code like during previous experiments – starting the laser and motors. The motors of the vehicles are set to a low speed in the code so that the vehicle does not overshoot its goal as it slowly increments. The number of time steps is one so at each scan, the vehicle can determine its location and move appropriately.

Within this code, a closed loop command is sent to the thrusters with specific gains regarding which direction should they move in. The closed-loop control is designated by the matrix below.

$$\begin{bmatrix} F_x \\ F_y \\ T_\theta \end{bmatrix} = \begin{bmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_\theta \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{bmatrix}$$

Figure 5-1. Force/Torque Input Matrix

The force and torque input matrix is the product of a 3x3 gain matrix and a 3x1 horizontal vector matrix. The former represents the gain values for the control system, whereas the latter matrix contains the vectors (x, y, θ). Each value in the horizontal matrix represents the force in a particular direction; the matrix is as follows: [surge sway yaw]. As previously stated, surge refers to the direction of which the sensor is facing (the 1-4 diagonal). Sway is perpendicular to surge, and yaw refers to the angular displacement from the surge direction. If the horizontal matrix is set to [1 0 0], there is now a step input of 1 N in the surge direction; the vehicle would move forward until the number of time steps runs out.

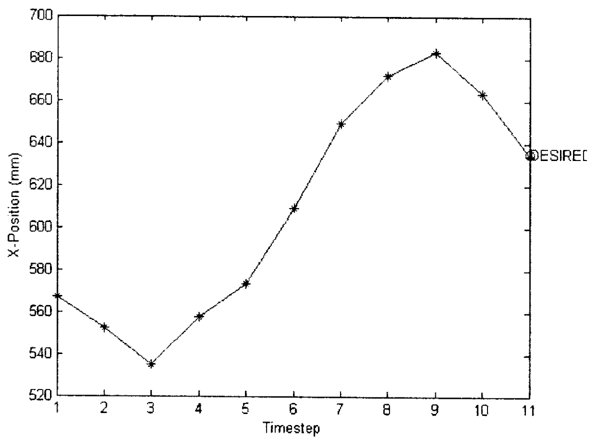
In the subsequent experiments, the forces or torques sent to the thrusters are determined by signs. The gain matrix is set to an identity matrix and the force/torque matrix determines whether the force is positive or negative. If the x-y difference is less than 15 mm, the force input is a negative step, while if it is greater, it is positive. In the yaw direction, it is opposite: if it is less than 80°, the torque is a positive step to rotate it more counterclockwise, and vice versa. In the scenario where the difference or the heading is near the preferred location, the vector is set to zero.

To effectively move to its desired spot, the raft must first rotate itself so that it can see all three sets of poles easily. In the algorithms above in locating vehicle's position, the matrix [*poles*] contains six values. If one or more are missing (for instance, the vehicle only sees two poles), the algorithm cannot run properly. Therefore, the code explicitly shows that if the vector length is less than six, set theta to an arbitrary number that does not fall within the acceptance range, such as zero. In this way, the program will loop back to the start and keep the vehicle rotating until it can see all poles.

After each scan, the program runs through the calculations for vehicle position. The values for x, y, and theta differences (between actual and desired) are updated, and the loop reroutes back to the top, respectively, if they do not fall within the range. If it does, the program ends the loop and continues on to the next loop or finishes.

5.1.2 Individual Control Results

All three control loops are able to perform either individually or in a sequence; most fall very close to the user-defined location. When implemented, there is oscillation around the predefined value because depending on where it is initially and the external factors, the vehicle is usually not able to reach it within a few time steps.



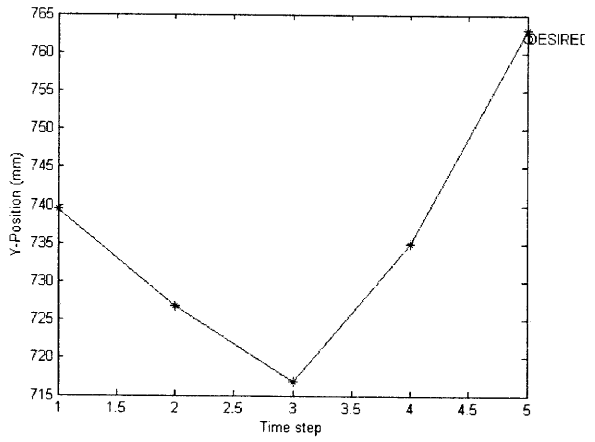


Figure 5.2: X and Y Position as a function of time when the respective control is implemented. The DESIRED text refers to where the vehicle's goal is.

As mentioned in Section 5.1.1, the x-y and heading differences are initially defined, so based on those numbers, the vehicle moved accordingly before taking another scan. Because of that initial move, the vehicle moves away from the user-defined location before changing directions. In this experiment, the vehicle was able to reach the y-value in five time steps, whereas it took approximately 11 steps for the x-direction. In the x-direction, as the raft is moving towards the x-coordinate, it stops just outside the range at time step 7, causing it to overshoot in the subsequent time steps. Its maximum overshoot is about 50 mm, while in y-direction, there was no overshoot since it was able to fall within the range on its way up. The controls for x and y are the same; therefore, the raft was positioned where it was able to fall within the buffer zone in the y-direction more easily than x. As the x-position graph has shown, until the raft is within that buffer zone, it continues to oscillate.

While implementing the controls sequentially proved that the raft was able to reach the x and y coordinates, it does not travel as expected. When implementing these loops, it was assumed that while the vehicle was moving towards its x or y value, it would stay in a straight line. For example, if it reached its y-destination already (y=12) and the x-loop begins, it was anticipated that it would stay on that y value when it arrived at its x-location. Due to external factors, it behaved differently than expected.

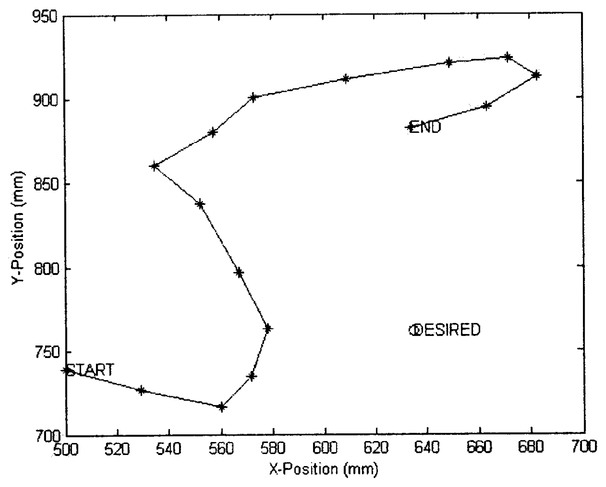


Figure 5-3: Vehicle Trajectory during the implementation of individual loops.

The expected trajectory of the vehicle is more like an L-shaped curve, where it moves along one axis and then another. In Figure 5-2, the vehicle, as shown in Figure 5-1, reaches the desired y-coordinate on the fifth time step (not taking in account the starting position). During its route, it does not stay along a specific x-value. The same behavior occurs when the raft orientates itself in the horizontal direction.

There are several factors that explain why this trajectory is different. During the experiment, the tether that is connected to the vehicle provides some external force. Because it is fixed at the ceiling, if the vehicle is too far displaced, the wire starts to pull it back slightly. Additionally, the thrusters are not completely perfect. Because the directions share the thrusters, even though the motors are outputting force in the sway direction, it moves a little in the surge direction. This behavior occurs in the yaw direction as well; while it rotates, it drifts to the side. Additionally, if the thrusters' speed is set too high, the moving raft causes small waves which can displace it.

5.2 Single Combined Loop

5.2.1 Implementation of One Loop

Combining the three loops into one loop is very similar to the implementation in Section 5.1.1. Everything, such as ranges and vectors, remain the same. The change mainly involves removing the code for each loop and putting them all together into one large loop. Additionally, the overarching while-loop contains all the constraints of each loop.

5.2.2 Single Loop Results

With three different directions to control, the vehicle takes a significantly longer time to reach its destination. Because there are some drifting involved and other miscellaneous factors, the vehicle can never remain in a particular x, y, or heading value. When it falls within the range of x, it tries to rotate or move towards the other two goals, and by doing so, it falls out of the x-range.

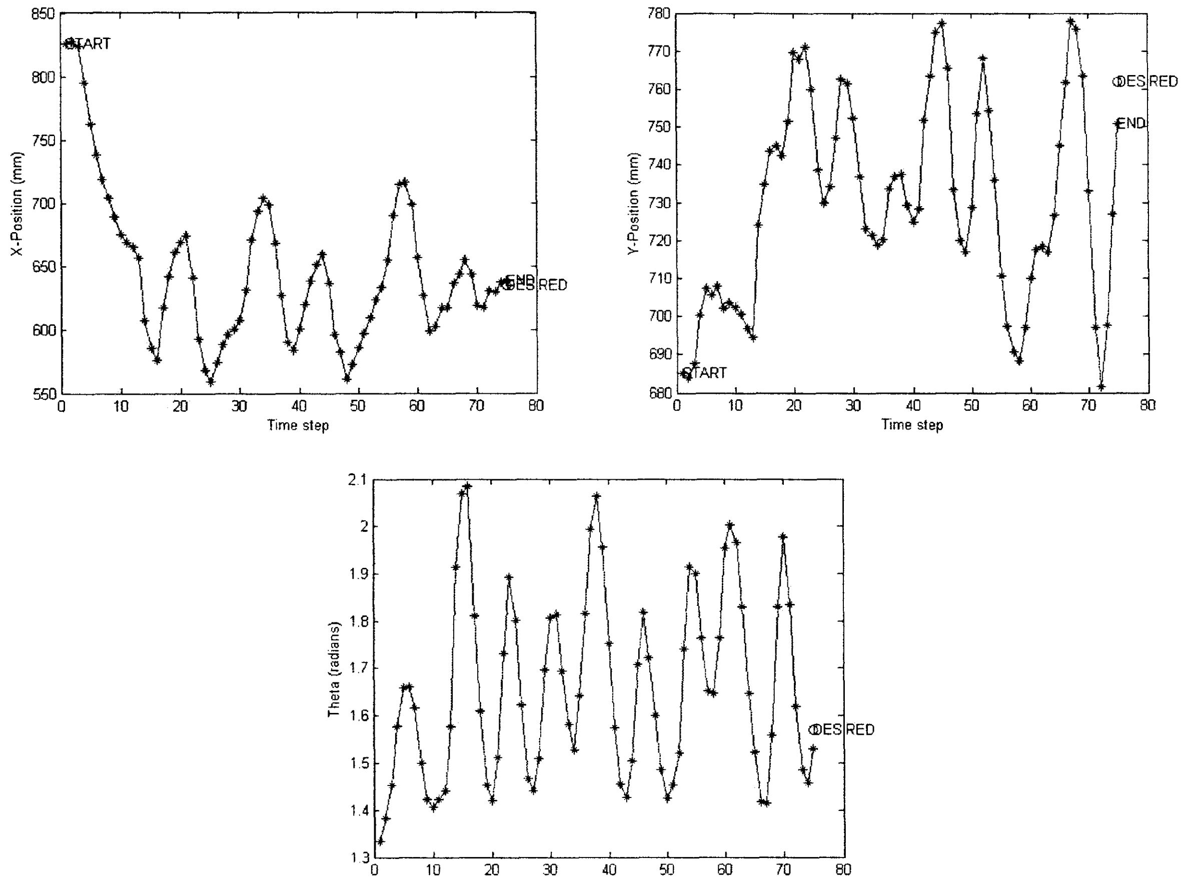


Figure 5-4. X-Y-θ Trajectory during Single Closed Loop.

In this experiment, the number of time steps for the vehicle to fall within the acceptable range of the desired location is seventy-five, which is significantly higher than when each control loop was implemented independently. The oscillations seen in by the x-trajectory in Figure 5-1 are more prominent and more in numbers in the above figure. Because there is much movement in the individual pathways, the vehicle's x-y trajectory is very chaotic; it moves around the desired location for a long time, as seen in the figure below.

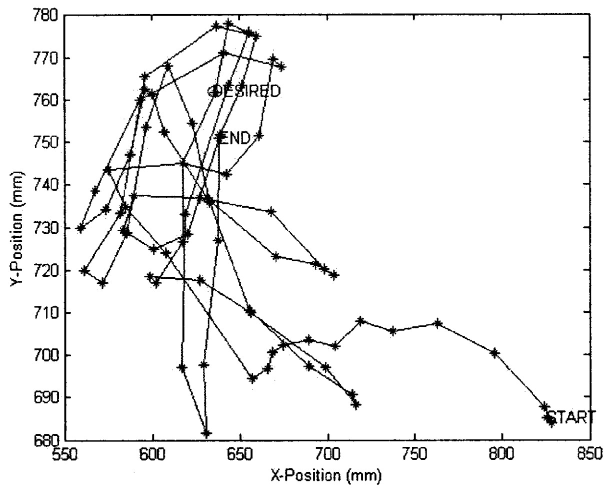


Figure 5-5. X-Y Trajectory during Single Closed Loop Control

5.2.3 Gain

The heading oscillation in the Figure 5-4 has approximately 0.65 radians peak-to-peak amplitude. Because that is a significant amount of variation, the gain of the heading, which was initially set to 1, is changed to 2. By changing the gain, the oscillation pattern is still similar, but the main difference is the peak-to-peak amplitude. With a higher gain, the heading has smaller amplitude when oscillating. The new amplitude is roughly 0.5 radians, which is significantly less than before.

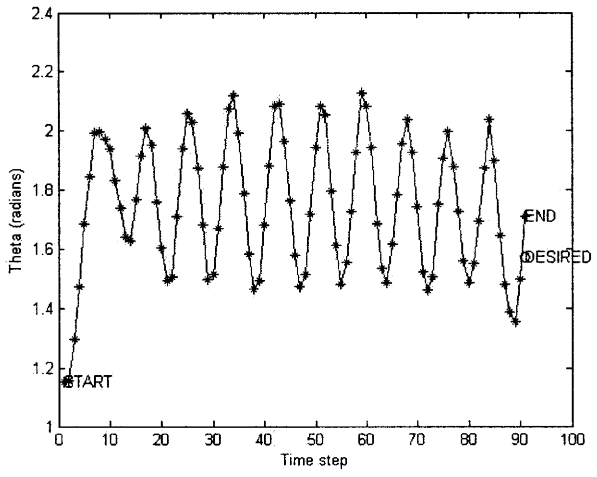


Figure 5-6. Heading as a function of time step with gain of 2.

5.3 Damping

After the gain was implemented, experiments were also performed to determine whether damping has any effect on the control system and vehicle.

5.3.1 Damping Function

As mentioned and shown in Section 5.2.2, there are oscillations in all directions of varying amplitude and frequency. A damping function is executed onto the control system, more specifically, on the heading. This function is entered as part of the force vector and is dependent of the difference of θ and θ_{last} (the current heading and the previous heading reading). If the difference is positive, there is a higher input into the control loop, adding more torque to the thrusters; if negative, there is less. The vector equation is modified as shown below.

$$\begin{bmatrix} F_x \\ F_y \\ T_\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{bmatrix} \text{ where}$$

$$\tilde{\theta} = \begin{cases} 2 + 3 * (\theta - \theta_{last}); & \theta < (\frac{\pi}{2} - 0.17) \\ 0; & \left(\frac{\pi}{2} - 0.17\right) \le \theta \le \left(\frac{\pi}{2} + 0.17\right) \\ -2 + 3 * (\theta - \theta_{last}); & \theta < (\frac{\pi}{2} + 0.17) \end{cases}$$

Figure 5-7. Damping on Heading. Depending on what the heading is, the torque input has three different possibilities. The gain on the damping function was arbitrarily set, and the gain of 2 was increased from before.

5.3.2 Damping Results

With this damping input on the heading of the vehicle, the oscillations are small and show some decreasing behavior as it reaches towards its destination. The decay of the oscillations mimics an exponential function. Additionally, the time to reach the desired spot decreases significantly. The decay function causes the heading to not fluctuate as much as it used to and thus, lowering the number of time steps. When the vehicle is in the right position, its heading is also close within range. This behavior contrasts with the one shown in Figure 5-6 or Figure 5-4, where the peak-to-peak amplitude was consistently the same throughout the experiment run. In a particular run, the addition of damping changed the oscillation from about 0.5 radians peak-to-peak to roughly 0.3 radians, and the oscillations were still decaying. The damping function is useful and effective in developing a good control system for the vehicle. The damped trajectory of the raft can be seen below in Figure 5-8.

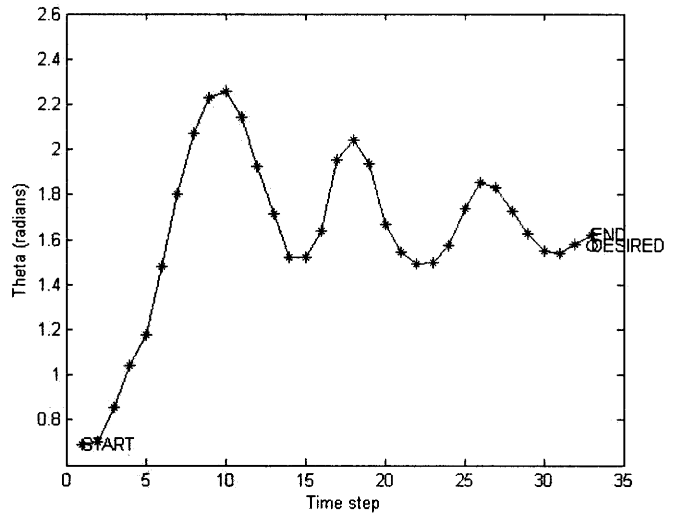


Figure 5-8. Heading with Damping as a function of time step.

While damping to the control system shows that both the numbers of oscillations and time steps decrease, it can do harm if it is not set correctly. For instance, if the gain is set too high, the vehicle will not be able to reach its destination. It will overshoot or undershoot whenever it tries to get close; the vehicle will keep oscillating until the motors or laser shut down or user stops the program.

5.4 Controlling Square-Wave Trajectory

After the damping function is proved to be effective, the final experiment focused on controlling the vehicle to move in a square wave-like trajectory. Because of the tether's constraint and previous experiments, the sway direction is selected to be the path which the vehicle would travel.

5.4.1 Square Wave on X-Position

To control the vehicle to move in a square wave-like trajectory requires slightly more code than the previous experiments. The basic design of this program includes one overarching while-loop with two smaller loops that control the vehicle (one for each end). Because the y-position is not necessary for the function, those constraints and its closed-loop can be removed. The large while-loop is set so that it can never be fulfilled; the loop or any vehicle motion needs to be stopped the user or until the number of time steps runs out.

Each of the smaller loops contains the same control loops and contents as before. Additionally, there is an extra variable that is initially set to zero (prior to the start of the big loop); this variable counts how many times the vehicle is within an acceptable range. The while-loop constraint is different than before in that the vehicle runs through this control system until the extra variable is less than six. Six is an arbitrarily set number and can be changed depending on preference. There is another small addition to the loops, which is when the vehicle near the desired location, the variable is incremented by one. When the raft reaches its destination six times or stays still for six time steps, the while-loop ends. After the loop ends, the variable (the counter) is set to zero.

After the first loop ends, the second control, with exactly the same information, a different x-destination, and counter variable name, starts. This second control causes the raft to move to a different location and stay there for six time steps. When this constraint is achieved, the larger while-loop brings the vehicle back to the first loop and back to its first position.

5.4.2 Square Wave Results

Going through this control system yields the expected square wave trajectory; however, rather than having smooth high and low peaks, the peaks are "noisy." The noise is the oscillations that the vehicle has when moving towards its goal. Whenever the raft reached its destination, it would not input force to its motors; therefore, due to drifting and the tether, the vehicle is slightly displaced enough to fall out of the buffer range. It proceeds to try to orientate itself correctly, creating the oscillations seen below in Figure 5-9. By being able to perform a square wave trajectory shows that the closed-loops implemented are working consistently and accurately.

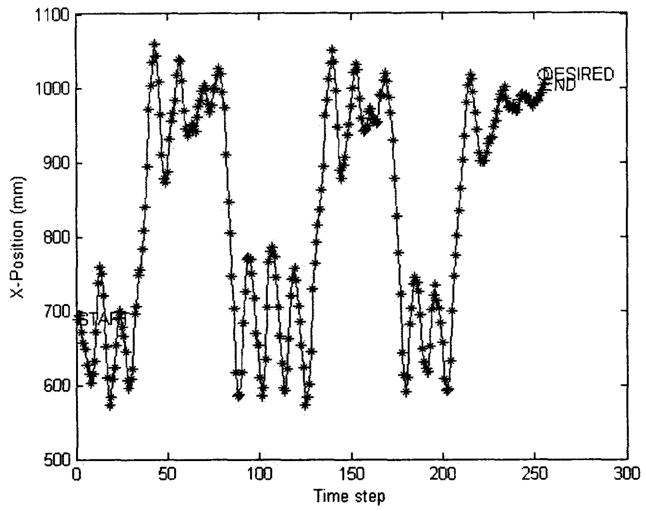


Figure 5-9: Vehicle's X-Position in a Square Wave Trajectory. The lower desired position is approximately 15 inches away from the higher location.

Chapter 6: Conclusions

6.1 Contributions

This thesis provides three main contributions to navigation for water vehicles and laser range scanning: development of the localization algorithm, creation of the mapping calculations for any position in the tank, and control loops for water vehicle to return to a desired position. Chapter 3 provides a detailed algorithm one needs to implement for localization of the vehicle; by implementing the equations in a technical computing software, such as MATLAB, one can deduce where the robotic raft's location and what its trajectory is in real-time. Based on the values obtained from Chapter 3, Chapter 4 develops another set of algorithms to map what the Hokuyo laser is seeing. Lastly, Chapter 5 documents and implements the closed loop controls on the vehicle for positioning precision. These controls allow it to return to a desired spot precisely despite from anywhere else. It further explores different aspects of the closed-loop system, such as gain, damping, and square-wave trajectory.

6.2 Conclusions

Based on the experiments and results from this research, there is a lot to take away about map-based control in general. Localization and mapping of the environment can be performed easily with knowledge about geometry; they sound more daunting than they truly are. However, it is important that the geometric constraints are accurate and correct; even the slightest error can output the wrong vehicle trajectory or display the pole locations completely off. Additionally, controlling a vehicle in real time is difficult. There are many different small issues that can cause the wrong outcome, such as the thrusters and tether issues. Even within the control loop, increasing the gain too high may cause endless oscillations.

Although the laser sensor cannot scan its environment in 3-D, there are a few good ideas obtained from using a 2-D view that are applicable for 3-D, space or underwater. More specifically, the use of pinpointing the middle of the objects in view is very useful, especially for 3-D, where objects are more complicated since they have depth. If the vehicle can pinpoint the middle of the object's surface area and knowing the depth, the vehicle can map out the center of the volume of the objects. Additionally, using slope of the scan image as the edge of an object is helpful. When scanning in 3-D or in space, if the laser registers a slope greater than a number, the algorithm can determine that it is an edge. This useful for scenarios where entities are dispersed. It is difficult to determine the boundary of an object if it is clustered together with other objects, especially if the slope from one to another is small.

The mapping ability of the vehicle is powerful; it is able to map out where the poles are with given a limited amount of information. Based on these results, if a vehicle operates in an unknown area and given measurements, it is possible for the vehicle map out its surroundings. It is unclear how the mapping algorithm would work with an environment that is not a circular tub, but the computation can be further developed for varying surroundings. The only concern is when the objects in the area are small and clustered together. As noted before, the laser is not very precise; it barely noticed that there was a dip between the pair of poles. Therefore, if there are small objects in the view, it would difficult for the sensor to determine whether it is an object or noise. Furthermore, the map that the vehicle outputs consists of single points; therefore, the most the vehicle knows where the center of an object is. Further development is necessary to display an accurate 2-D map.

6.3 Future Work

There are several different pathways for future work on this thesis. One of the major problems during this research is that the thrusters and tether are major issues with the raft. The tether restricted the raft's range of position; it cannot stray too far away before the tether would start pulling it back. Because the raft is partially submerged in water, it does not take much force to start moving it back. Additionally, the thrusters are not very robust; when it is supposed to be moving in sway or surge, sometimes they cause the raft to drift off to the other direction. The combination of these two external constraints limits what is physically achievable in positioning precision. Figuring a solution to these two issues is important for future work, such as the usage of different thrusters or a better method to send power and commands to the motors.

As mentioned in the Conclusion section, the mapping of the environment can be more accurate than using points as representation. If the vehicle can map out the surface of the poles, it would give a better representation and be more accurate for mapping purposes. In order to include this type of feature, more extensive computation is necessary in the mapping algorithm.

One possibility is to further develop both localization and mapping computations for different environments, such as a rectangular room or an environment that has sharp turns or edges. All of the previously mentioned experiments been performed in a medium-sized circular tub. Therefore, because the walls lacked edges or sharp turns, the algorithm is simple and not useful for more complex environments. In particular, experiment should be performed in an environment that has four or more sides, and to successfully progress the algorithm, the vehicle needs to be positioned so that it can see at least two corners that any given moment. With a change in setting, both mapping and localization will be more robust. The control loop should also be further explored so that it is applicable in any kind of environment.

Additionally, in the current environment, the control loop is not very robust. There are still many oscillations about the user-defined position and the vehicle takes too many steps to reach the location. The code and control loop should be modified so that they minimize overshoot or undershoot. By doing so, the number of oscillations and time steps in an experiment run are likely to decrease. In an ideal scenario, the raft can move from one point to the next in a straight line without deviation. Another possibility is to determine whether the square-wave trajectory is applicable to the y-axis.

APPENDIX A

MATLAB Code for Vehicle Position and the Plotting of Poles

```
%plot step
a = 2;
figure;

% Loop that goes through each of the frames
for k = 2:a:50

%% Code finds the poles
    p=0;
    for n = 100:1:700
        %% Using the slope between two frames to determine whether
there was a pole reading
        if ((Karray(k,n)-Karray(k,n+2))/2 >= 22 || (Karray(k,n)-
Karray(k,n+2))/2 <= -22)
            p=p+1;
            polesx(p) = [n];
        end
    end

%% Fixing the x poles, in case there was more than six readings
for q = 1:1:(length(polesx)-1)
    if ((polesx(q) > polesx(q+1)-3) && (polesx(q) < polesx(q+1)+3))
        polesx(q)=[0];
    end
end
index = find(polesx == 0 );
polesx(index) = [];

%% Actual Pole coordinates
x1 = 18.5 * 25.4; %17.5+1 inch to mm
x2 = 27 * 25.4; %17.5+2+5.5+2 inch to mm
x3 = 41 * 25.4; %17.5+2+5.5+4+11+1 inch to mm
y = 54.5 * 25.4; %54.5 inch to mm

%% Angle (Degrees) of the poles
b1d = ( polesx(2)+polesx(1) )/2 * 270/768 - 135;
b2d = ( polesx(4)+polesx(3) )/2 * 270/768 - 135;
b3d = ( polesx(6)+polesx(5) )/2 * 270/768 - 135;
%% Angle (Radians)
b1 = b1d/360*2*pi;
b2 = b2d/360*2*pi;
b3 = b3d/360*2*pi;

b1dm(k/a)=[b1d];
b2dm(k/a)=[b2d];
b3dm(k/a)=[b3d];

%% Radius from the vehicle
```

```
r1 = Karray(k,round((polesx(2)+polesx(1))/2));
r2 = Karray(k,round((polesx(4)+polesx(3))/2));
r3 = Karray(k,round((polesx(6)+polesx(5))/2));

r1m(k/a)=[r1];
r2m(k/a)=[r2];
r3m(k/a)=[r3];

%% Vehicle
syms xv yv;

%% Equations (X-Y Position)
eq1 = (xv-x1)^2 + (yv-y)^2 - r1^2;
eq2 = (xv-x2)^2 + (yv-y)^2 - r2^2;
eq3 = (xv-x3)^2 + (yv-y)^2 - r3^2;

%% Solving
[xv, yv] = solve(eq1, eq3, xv, yv);

%% Putting the data into a matrix
A = double(xv(1));
B = double(yv(2));
xvec(k/a) = [A];
yvec(k/a) = [B];

%% Finding theta (heading)
if A >= x1
    theta= atan(abs((y-B)/(x1-A))) - double(b1);
else
    theta=pi- atan(abs((y-B)/(x1-A))) - double(b1);
end

if B <= x3
    theta3= pi - atan(abs((y-B)/(x3-A))) - double(b3);
else
    theta3=atan(abs((y-B)/(x3-A))) - double(b3);
end
tvec(k/a)=[theta];

%% Replotting the poles
x1p = A - r1 * cos(double(b1)+theta);
y1p = B + r1 * sin(double(b1)+theta);
x2p = A - r2 * cos(double(b2)+theta);
y2p = B + r2 * sin(double(b2)+theta);
y3p = B + r3 * sin(double(b3)+theta);
x3p = A - r3 * cos(double(b3)+theta);

C = double(x1p);
D = double(y1p);
E = double(x2p);
F = double(y2p);
G = double(x3p);
```

```
H = double(y3p);

x1pm(k/a) = [C];
y1pm(k/a) = [D];
x2pm(k/a) = [E];
y2pm(k/a) = [F];
x3pm(k/a) = [G];
y3pm(k/a) = [H];

end

%% Plotting the variables
plot(xvec,yvec,'-')
hold on
%% Plotting the poles
h=2*25.4;
w=2*25.4;
rectangle('Position',[x1-25.4, y, w, h],'Curvature',[1,1])
hold on
rectangle('Position',[x2-25.4*2, y, w, h],'Curvature',[1,1])
hold on
rectangle('Position',[x2, y, w, h],'Curvature',[1,1])
hold on
rectangle('Position',[x3-25.4, y, w, h],'Curvature',[1,1])
hold on

%% Plot Loop (to make the Rectangles)
for m = 1:1:50/a
    xvm = double(xvec(m));
    yvm = double(yvec(m));
    tvm = double(tvec(m));

    recx1 = -10;
    recy1 = -10;
    recx2 = 10;
    recy2 = -10;
    recx3 = 10;
    recy3 = 10;
    recx4 = -10;
    recy4 = 10;

    %% Rotating the vertices
    xvm1 = recx1*cos(tvm) + recy1*sin(tvm) + xvm;
    yvm1 = -recx1*sin(tvm) + recy1*cos(tvm) + yvm;
    xvm2 = recx2*cos(tvm) + recy2*sin(tvm) + xvm;
    yvm2 = -recx2*sin(tvm) + recy2*cos(tvm) + yvm;
    xvm3 = recx3*cos(tvm) + recy3*sin(tvm) + xvm;
    yvm3 = -recx3*sin(tvm) + recy3*cos(tvm) + yvm;
    xvm4 = recx4*cos(tvm) + recy4*sin(tvm) + xvm;
    yvm4 = -recx4*sin(tvm) + recy4*cos(tvm) + yvm;

    rect.vertices = [xvm1 yvm1; xvm2 yvm2; xvm3 yvm3; xvm4 yvm4];
    rect.faces =[1 2 3 4];
    patch(rect, 'Vertices', rect.vertices,'FaceColor',[0 0 1]);
end
hold on
text(xvec(1), yvec(1), 'START');
hold on
text(xvec(50/a), yvec(50/a), 'END');
hold on

%% Plotting the poles
plot(x1pm, y1pm, 'g*');
hold on
plot(x3pm, y3pm, 'r+');
hold off
axis('equal')
ylabel('Distance (mm)')
xlabel('Distance (mm)')

% Plots how much the sensor's heading changes with scan number
figure;
tvecx = 1:1:50/a;
plot(tvecx, tvec/(2*pi)*360,'o-')
ylabel('Degrees for Theta')
xlabel('Scan number')
```

APPENDIX B

MATLAB Code for Vehicle Control (Single Closed-Loop with Gain and Damp)

```
%% Insert in code to start motors

%% Predetermined differences: to jump start the vehicle's motion
thetame = pi/4; %%Heading in radians
yme = 100; %% Difference between wanted and current y-location
ywant = 30*25.4; %% Where the vehicle want to be
xme = 100; %%Difference between wanted and current x-location
xwant = 25*25.4; %%Where the vehicle want to be
%% thetadelta = 0; %% Setting the difference between current and
previous heading to zero

%% Control loop: until all six constraints are met, the vehicle will
keep cycling through
while thetame > (pi/2 + .17) || thetame < (pi/2 - .17) || yme > 15 ||
yme < -15 || xme > 15 || xme < -15

%% Insert code that starts laser, converts laser images to Cartesian
coordinates

%% Force/Torque Command
%% Determines the sign of the inputs
if yme < -15
   u1 = -1;
elseif yme > 15
   u1 = 1;
else u1=0;
end

if xme < -15
   u2 = -1;
elseif xme > 15
   u2 = 1;
else u2=0;
end

%% If want to increase or decrease the gain, it can changed in Rmat or
change what u3 is equaled to
%% If want to add a damp, modify u3 to equal to u3=±2+3*(thetadelta)
where thetadelta is modified later on
if thetame < (pi/2 - .17)
    u3 = 2;
elseif thetame > (pi/2 + .17)
    u3 = -2;
else u3=0;
end

Rmat=eye(3);
```

```
%% Force/Torque Matrix
Ucom = Rmat*[u1;u2;u3];
Ul = Ucom(1,1);
Vl = Ucom(2,1);
Tl = Ucom(3,1);
%% Enter in code that controls the motors.

%% Sets the speed of the thrusters
    fprintf(r,'m1f175\n');    %sets motor 1
    fprintf(r, char([109 49 102 Aa Bb Cc 92 110]));
    fprintf(r,'m2r175\n');  %sets motor 2
    fprintf(r, char([109 50 102 Aa Bb Cc 92 110]));
    fprintf(r,'m3f175\n');    %sets motor 3
    fprintf(r, char([109 51 102 Aa Bb Cc 92 110]));
    fprintf(r,'m4r175\n');    %sets motor 4
    fprintf(r, char([109 52 102 Aa Bb Cc 92 110]));

%% Insert code that sends above info to thrusters
%% Insert Localization code

%% Updates the values of thetame, yme, xme, and other variables
%% thetalast = thetame;  %% sets the last heading before the next line
updates it
thetame= atan(abs((y-B)/(x1-A))) - double(b1);
yme = B-ywant;
xme = A-xwant;
%% thetadelta = thetame-thetalast;  %% sets the difference


end
```

APPENDIX C

MATLAB Code for Square Wave

```
%% Insert in code to start motors

%% Predetermined differences: to jump start the vehicle's motion
thetame = pi/4;
yme = 100;
xme = 100;
thetadelta = 0;
ttime = 0;  %% Counters for loop 1
tttime = 0;  %% Counters for loop 2

%% Overarching While loop: until yme <= 0, the vehicle will keep
cycling through; since we do not deal with the y-coordinate, this will
never happen
while yme>0

%% First control loop
while ttime<6
xwant = 25*25.4;

%% Insert code that starts laser, converts laser images to Cartesian
coordinates

%% Force/Torque Command
%% Determines the sign of the inputs

if xme < -15
   u2 = -1;
elseif xme > 15
   u2 = 1;
else u2=0;
end

if thetame < (pi/2 - .17)
    u3=2+3*thetadelta;
elseif thetame > (pi/2 + .17)
    u3 = -2;
else u3=0;
end

Rmat=eye(3);

%% Force/Torque Matrix
Ucom = Rmat*[0;u2;u3];
Ul = Ucom(1,1);
Vl = Ucom(2,1);
Tl = Ucom(3,1);
%% Enter in code that controls the motors.
%% Set the speed of the thrusters
%% Insert Localization code

%% If the vehicle falls within the range, the counter increments by
one
if (xme < 15 && xme > -15)
    ttime=ttime+1
end

%% Updates the values of thetame, xme, and other variables
thetalast = thetame;
thetame= atan(abs((y-B)/(x1-A))) - double(b1);
xme = A-xwant;
thetadelta = thetame-thetalast;

%% End of the first loop
end


%%Sets the first loop counter to zero
ttime=0;

%% Second control loop
while ttime<6
xwant = 25*25.4; %%New x-value

%% Insert code that starts laser, converts laser images to Cartesian
coordinates

%% Force/Torque Command
%% Determines the sign of the inputs

if xme < -15
   u2 = -1;
elseif xme > 15
   u2 = 1;
else u2=0;
end

if thetame < (pi/2 - .17)
    u3=2+3*thetadelta;
elseif thetame > (pi/2 + .17)
    u3 = -2;
else u3=0;
end

Rmat=eye(3);

%% Force/Torque Matrix
Ucom = Rmat*[0;u2;u3];
Ul = Ucom(1,1);
Vl = Ucom(2,1);
```

```
Tl = Ucom(3,1);
%% Enter in code that controls the motors.

%% Set the speed of the thrusters
%% Insert code that sends above info to thrusters
%% Insert Localization code

%% If the vehicle falls within the range, the counter increments by
one
if (xme < 15 && xme > -15)
    tttime=tttime+1
end

%% Updates the values of thetame, xme, and other variables
thetalast = thetame;
thetame= atan(abs((y-B)/(x1-A))) - double(b1);
xme = A-xwant;
thetadelta = thetame-thetalast;

%% End of the second loop
end

%%Sets the first loop counter to zero
tttime=0;

%%End of while-loop
%% User should hit ctrl+c to stop the loop
end
```

Bibliography

Kokko, Michael A. Range-based navigation of AUVs operating near ship hulls. Thesis. Massachusetts Institute of Technology, 2007. Cambridge: Massachusetts Institute of Technology, 2007.

Sohn, Hee Jin, and Byung Kook Kim. "An Efficient Localization Algorithm Based on Vector Matching for Mobile Robots Using Laser Range Finders." Journel of Intelligent and Robotic Systems 51 (2008): 461-88.

Temeltas, Hakan, and Deniz Kavak. "SLAM for Robot Navigation." IEEE Aerospace and Electronic Systems Magazine Dec. 2008: 16-19.