

# Navigation and Control of an Autonomous Vehicle

Ian Schworer

Thesis submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

Master of Science  
in  
Electrical Engineering

Dr. Pushkin Kachroo, Chair  
Dr. A Lynn Abbott  
Dr. William Baumann

April 29, 2005  
Blacksburg, Virginia

Keywords: autonomous vehicle, mobile robot navigation, path following, trajectory tracking, optimal control

Copyright 2005, Ian Schworer

# Navigation and Control of an Autonomous Vehicle

Ian Schworer

(ABSTRACT)

The navigation and control of an autonomous vehicle is a highly complex task. Making a vehicle intelligent and able to operate “unmanned” requires extensive theoretical as well as practical knowledge. An autonomous vehicle must be able to make decisions and respond to situations completely on its own. Navigation and control serves as the major limitation of the overall performance, accuracy and robustness of an autonomous vehicle. This thesis will address this problem and propose a unique navigation and control scheme for an autonomous lawn mower (ALM).

Navigation is a key aspect when designing an autonomous vehicle. An autonomous vehicle must be able to sense its location, navigate its way toward its destination, and avoid obstacles it encounters. Since this thesis attempts to automate the lawn mowing process, it will present a navigational algorithm that covers a bounded region in a systematic way, while avoiding obstacles. This algorithm has many applications including search and rescue, floor cleaning, and lawn mowing. Furthermore, the robustness and utility of this algorithm is demonstrated in a 3D simulation.

This thesis will specifically study the dynamics of a two-wheeled differential drive vehicle. Using this dynamic model, various control techniques can then be applied to control the movement of the vehicle. This thesis will consider both open loop and closed loop control schemes. Optimal control, path following, and trajectory tracking are all considered, simulated, and evaluated as practical solutions for control of an ALM.

To design and build an autonomous vehicle requires the integration of many sensors, actuators, and controllers. Software serves as the glue to fuse all these devices together. This thesis will suggest various sensors and actuators that could be used to physically implement an ALM. This thesis will also describe the operation of each sensor and actuator, present the software used to control the system, and discuss physical limitations and constraints that might be encountered while building an ALM.

# Acknowledgements

I would like to thank several individuals for their support, motivation, and inspiration while writing this thesis. Only through their constant financial, intellectual and emotional support could this work have been completed.

First and foremost I would like to thank Virginia Tech's Bradley Department of Computer and Electrical Engineering. By awarding me a Bradley Fellowship you have made my graduate studies a tremendously rewarding and enlightening experience. The funding has not only provided financial support but has also allowed me to follow my own interests and dreams. I would like to personally thank John Rocovich and the Via family for endowing this fellowship and providing me and the other Bradley Fellows with this amazing funding opportunity. This fellowship is proof that hard work during undergraduate studies does pay off.

I also like to thank Dr. Pushkin Kachroo, my graduate advisor and professor, for his constant intellectual support. It is rare to meet someone so smart yet so personable. His classes were inspiring and he always seemed to make the material relevant. He always listened to all my questions and concerns and seemed as if he had the answers to everything. He is truly one of the smartest people I have ever met and I feel grateful to have met and known him.

Also, I would like to thank my family for all their love and support. My family has always been there for me, to help me through personal hardships and push me to succeed. They have taught me the value of an education and the benefit of hard work. I am where I am today only because of their love and support. Mom, Dad, Page, Kyle, Cameron, Isaac, and Grandma, thank you.

I would also like to thank my girlfriend, Adele, for being such a sweet, caring, and understanding person. She has been my motivation to finish this work and pushed me to keep going. Her love has been inspiring and I cannot wait until we can be close again.

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction.....</b>                                 | <b>1</b>  |
| 1.1      | Terminology.....   | 1         |
| 1.2      | Motivation.....  | 2         |
| 1.3      | Existing Autonomous Vehicles.....                        | 3         |
| 1.3.1    | Military Applications.....                               | 3         |
| 1.3.2    | Consumer Applications.....                               | 4         |
| 1.4      | Contributions of this Thesis.....                        | 5         |
| 1.5      | Organization of this Thesis.....                         | 6         |
| <b>2</b> | <b>Autonomous Vehicle Research at Virginia Tech.....</b> | <b>7</b>  |
| 2.1      | The Complexity of Autonomous Vehicle Design.....         | 7         |
| 2.2      | Research and Accomplishments at Virginia Tech.....       | 8         |
| 2.3      | ION Navigation Competition.....                          | 9         |
| <b>3</b> | <b>Navigation.....</b>                                   | <b>10</b> |
| 3.1      | Overview.....  | 10        |
| 3.2      | Autonomous Lawn Mower Navigation Problem.....            | 11        |
| 3.3      | Region Filling with Random Obstacle Avoidance.....       | 13        |
| 3.3.1    | Behaviors and Data Structures.....                       | 13        |
| 3.3.2    | Autonomous Lawn Mowing.....                              | 18        |
| 3.4      | Simulation.....  | 19        |
| 3.4.1    | 3DState’s 3D Graphics Engine.....                        | 19        |
| 3.4.2    | Simulation Environment and Results.....                  | 19        |
| 3.5      | Efficiency of the ALM’s Navigation System.....           | 22        |
| 3.5.1    | Exact Cellular decomposition.....                        | 25        |
| 3.5.1.1  | The Boustrophedon Decomposition.....                     | 25        |
| 3.5.1.2  | The Traveling salesman problem (TSP).....                | 26        |

|            |   |           |
|------------|---|-----------|
| <b>4</b>   | <b>Mathematical Modeling and Control.....</b>       | <b>28</b> |
| <b>4.1</b> | <b>Mathematical Modeling.....</b>                   | <b>28</b> |
| 4.1.1      | Overview.....                                       | 28        |
| 4.1.2      | Nonholonomic Constraints .....                      | 29        |
| 4.1.3      | Global Coordinate Model .....                       | 30        |
| <b>4.2</b> | <b>Open Loop Control.....</b>                       | <b>32</b> |
| 4.2.1      | Inverse Kinematics.....                             | 33        |
| 4.2.2      | Optimal Control .....                               | 35        |
| 4.2.2.1    | Overview.....                                       | 35        |
| 4.2.2.2    | Physical Constraints and Performance Criteria ..... | 36        |
| 4.2.2.3    | Obtaining a Solution – Numerical Techniques.....    | 37        |
| 4.2.2.4    | Simulation Results .....                            | 40        |
| <b>4.3</b> | <b>Closed Loop Control.....</b>                     | <b>44</b> |
| 4.3.1      | Path Following.....                                 | 47        |
| 4.3.1.1    | Path Coordinate Model .....                         | 47        |
| 4.3.1.2    | Chain Form .....                                    | 49        |
| 4.3.1.3    | Input Scaling Controller.....                       | 50        |
| 4.3.1.4    | Simulation.....                                     | 53        |
| 4.3.2      | Trajectory Tracking .....                           | 57        |
| 4.3.2.1    | Chain Form .....                                    | 57        |
| 4.3.2.2    | Reference trajectory generation .....               | 58        |
| 4.3.2.3    | Control via Approximate Linearization.....          | 59        |
| <b>5</b>   | <b>Hardware Implementation.....</b>                 | <b>64</b> |
| <b>5.1</b> | <b>Overall System Structure.....</b>                | <b>64</b> |
| <b>5.2</b> | <b>Microprocessors .....</b>                        | <b>66</b> |
| <b>5.3</b> | <b>Sensor and Actuator Subsystems .....</b>         | <b>67</b> |
| 5.3.1      | Localization.....                                   | 67        |
| 5.3.1.1    | Position .....                                      | 68        |
| 5.3.1.2    | Orientation .....                                   | 71        |
| 5.3.2      | Boundary and Obstacle Detection .....               | 72        |
| 5.3.3      | Motor Control .....                                 | 73        |
| 5.3.3.1    | PID Control.....                                    | 74        |
| 5.3.3.2    | Wheel Encoders .....                                | 74        |
| 5.3.3.3    | Pulse Width Modulation (PWM) and the H bridge ..... | 75        |
| <b>5.4</b> | <b>Power.....</b>                                   | <b>76</b> |
| <b>6</b>   | <b>Conclusions.....</b>                             | <b>78</b> |
| <b>6.1</b> | <b>Concluding Remarks .....</b>                     | <b>78</b> |
| <b>6.2</b> | <b>Future Work.....</b>                             | <b>79</b> |
|            | <b>Bibliography.....</b>                            | <b>81</b> |

# List of Figures

|   |    |
|---|----|
| <b>Figure 3.1:</b> Filling an elemental region.....   | 12 |
| <b>Figure 3.2:</b> Graphical Representation of Closed_Curve Data Structure.....                             | 14 |
| <b>Figure 3.3:</b> OPTIMAL_PATH Step 1.....   | 15 |
| <b>Figure 3.4:</b> OPTIMAL_PATH Step 2.....   | 16 |
| <b>Figure 3.5:</b> OPTIMAL_PATH Step 3.....   | 16 |
| <b>Figure 3.6:</b> OPTIMAL_PATH Step 4.....   | 17 |
| <b>Figure 3.7:</b> Flow chart for Autonomous Lawn Mowing.....   | 18 |
| <b>Figure 3.8:</b> The ALM exploring its environment and searching for a boundary.....                      | 20 |
| <b>Figure 3.9:</b> The ALM tracing an unknown internal border.....  | 20 |
| <b>Figure 3.10:</b> The ALM traversing its calculated optimal path.....                                     | 21 |
| <b>Figure 3.11:</b> The ALM completing its traversal of the lawn.....                                       | 21 |
| <b>Figure 3.12:</b> Inefficient Yard Geometry.....  | 22 |
| <b>Figure 3.13:</b> Efficient Yard Geometry.....  | 23 |
| <b>Figure 3.14:</b> Yard Geometry in which map rotation does not minimize path overlap ...                  | 24 |
| <b>Figure 3.15:</b> Dividing a yard into subregions.....  | 24 |
| <b>Figure 3.16:</b> Boustrophedon decomposition.....  | 25 |
| <b>Figure 3.17:</b> A graph composed of nodes and edges.....  | 26 |
| <b>Figure 4.1:</b> Nonholonomic constraint of a two-wheeled differential drive robot.....                   | 29 |
| <b>Figure 4.2:</b> Kinematic Global Position Model of a two-wheeled differential drive robot<br>.....       | 30 |
| <b>Figure 4.3:</b> Open Loop Control Paradigm.....  | 32 |
| <b>Figure 4.4:</b> Inverse Kinematics of a two-wheel differential drive robot.....                          | 33 |
| <b>Figure 4.5:</b> Flow Chart of Steepest Descent Algorithm.....  | 39 |
| <b>Figure 4.6:</b> Movement of Robot (blue) vs Desired Path (green).....                                    | 40 |
| <b>Figure 4.7:</b> Transition of States, X, Y, and Theta and Input Wheel Velocities.....                    | 41 |
| <b>Figure 4.8:</b> Transition of Cost Function.....   | 41 |
| <b>Figure 4.9:</b> Movement of Robot (blue) vs Desired Path (green).....                                    | 42 |
| <b>Figure 4.10:</b> Transition of States, X (blue), Y (green), and Theta and Input Wheel<br>Velocities..... | 43 |
| <b>Figure 4.11:</b> Transition of Cost Function.....  | 43 |
| <b>Figure 4.12:</b> Closed Loop Control Paradigm.....   | 45 |

|  |    |
|--|----|
| <b>Figure 4.13:</b> Robotic Motion Tasks .....   | 45 |
| <b>Figure 4.14:</b> The ALM utilizing a path following controller.....                                 | 46 |
| <b>Figure 4.15:</b> The ALM utilizing a trajectory tracking controller.....                            | 46 |
| <b>Figure 4.16:</b> Path Coordinate Model.....   | 47 |
| <b>Figure 4.17:</b> Initial Configuration of Path Following Simulation .....                           | 53 |
| <b>Figure 4.18:</b> Path Displacement, $d$ (m), and Heading Error, $\theta_e$ (Degrees) .....          | 54 |
| <b>Figure 4.19:</b> Control Inputs, $u_1$ (m/s) and $u_2$ (rad/s).....                                 | 54 |
| <b>Figure 4.20:</b> Linear velocity, $v$ (m/s), and Rotational velocity, $\omega$ (rad/s).....         | 55 |
| <b>Figure 4.21:</b> Wheel Velocities, $\omega_L$ (rad/s) and $\omega_R$ (rad/s).....                   | 56 |
| <b>Figure 4.22:</b> Initial Configuration of Trajectory Tracking Simulation .....                      | 61 |
| <b>Figure 4.23:</b> Actual Trajectory (blue) versus Desired Trajectory (red) of Robot.....             | 62 |
| <b>Figure 4.24:</b> States $X$ (m), $Y$ (m) and $\theta$ (Degrees).....                                | 62 |
| <b>Figure 4.25:</b> Inputs, Linear velocity, $v$ (m/s), and Rotational velocity, $\omega$ (rad/s)..... | 63 |
| <b>Figure 4.26:</b> Wheel Velocities, $\omega_L$ (rad/s) and $\omega_R$ (rad/s).....                   | 63 |
| <b>Figure 5.1:</b> Functional Block Diagram of ALM.....  | 65 |
| <b>Figure 5.2:</b> Microprocessor requirements of the ALM subsystems.....                              | 66 |
| <b>Figure 5.3:</b> Physical representation of Trilateration .....                                      | 68 |
| <b>Figure 5.4:</b> Mathematical representation of Trilateration.....                                   | 69 |
| <b>Figure 5.5:</b> Effects of TOA error on Trilateration .....   | 69 |
| <b>Figure 5.6:</b> Diagram of Outside and Internal borders.....  | 72 |
| <b>Figure 5.7:</b> Block diagram of the motor control system.....                                      | 73 |
| <b>Figure 5.8:</b> Diagram of a Wheel Encoder .....  | 75 |
| <b>Figure 5.9:</b> PWM Signal with various Duty Cycles .....   | 75 |
| <b>Figure 5.10:</b> H-bridge Circuit .....   | 76 |

# Chapter 1

## 1 Introduction

### 1.1 Terminology

The definition of an autonomous vehicle is rather vague and open to much debate. It is hard to define exactly what an autonomous vehicle is because the terms used to describe it are also open-ended. This thesis will define an autonomous vehicle as a mobile robot that can intelligently navigate itself within an environment without human interaction. Unfortunately, terms such as robot, intelligence, and environment have many different meanings. Therefore, in order to understand the autonomous vehicle studied in this thesis those terms must also be defined.

When one hears the word robot many different images are conjured up in their head. Hollywood has made robots mainstream and presented its idea of a robot throughout countless movies. *Star Wars*, *A.I.*, and *I, Robot* are only a few examples of movies where robots play an important role. These robots all tend to be highly sophisticated and have many human-like qualities. However, these fictional examples are a poor representation of the actual robots that currently exist and operate today.

For the context of this thesis the term robot will describe a machine that has a perceived level of intelligence. Therefore, a robot can interpret inputs and respond to them in a useful way. For this work, a robot will collect data from its sensors, process it, and then respond to it by controlling its actuators. To an outside observer the robot appears to be making intelligent decisions based on situations it encounters in its environment.



The term environment must also be defined. With the world so complex, assumptions must be made about the environment in order for a robot to interact with it. Not every event or situation can be planned for. However, by creating a simple model of the world around the robot, hopefully, the most important situations can be planned for. Robots have been designed for numerous environments including land, air, sea, and outer space. This thesis assumes that the robot will operate on flat ground and thus will have its movement restricted to two dimensions, using an x and y coordinate system. It is also assumed that the robot will have to react to obstacles its sensors can detect.

## 1.2 Motivation

The study of autonomous vehicles is a fairly new area of research. It can be considered a specialized branch of robotics and has only been made possible due to the most recent technological advancements. The study of robotics and autonomous vehicles emerged from humans' interest in controlling the world around them. Humans have always sought new inventions that make their lives simpler. They have strived to explore and go where they have not gone before. From these desires the study of robotics and autonomous vehicles was born.

The birth of the microprocessor in the seventies created a technological explosion, opening numerous areas of research. One such field was sensor technology. Sensors are devices that change a physical quantity into an electrical signal, thus allowing that quantity to be measured. Scientists have created sensors that detect anything from temperature to velocity. Unfortunately, it has only been in the past few decades that the cost of the microprocessor and sensors have been affordable to anyone besides the military and government.

With the invention and advancement of these devices as well as their decreasing costs, the study of robotics is now open to anyone. New ideas and research are constantly emerging. Autonomous vehicles have the potential to make our lives simpler and in some cases protect our livelihood. Autonomous vehicles could mow our lawns, drive us around, or fight for our country. It is from these benefits that research and funding in this area will continue indefinitely for years to come.

According to a report issued by the U.N. Economic Commission for Europe and the International Federation of Robotics, in 2003, 607,000 domestic robots were in use worldwide. Based on statistical trends, the report estimates that by 2007, 4.1 million domestic robots will be used worldwide for various domestic chores. The numbers for industrial and military robots are also expected to skyrocket in the coming decade [10].

Unfortunately, there are a number of issues that must be addressed before autonomous vehicles become commonplace in our daily lives. One of the most restrictive problems is an autonomous vehicle's navigation and control system. These systems give an

autonomous vehicle the intelligence needed to perform a task or move in their environment. These systems have a huge impact on an autonomous vehicle's performance, robustness, and utility. Currently, autonomous vehicles can only perform the most basic of operations. It is from these problems and the potential benefits of these vehicles that this work is motivated.

## **1.3 Existing Autonomous Vehicles**

### **1.3.1 Military Applications**

In the past decade, due to constant world conflict and the rapid advancement of technology, there has been a great demand for autonomous vehicle research within the U.S. military. Within the next twenty years the U.S. military hopes to have a significant percentage of its fighting force composed of autonomous vehicles.

Autonomous vehicles are the preferred method of fighting in the future due to their efficiency, data collection abilities, and protection of human life. Autonomous vehicles will be smaller, lighter, more fuel efficient, and cheaper than their currently manned counterparts. Furthermore, since it is a machine it will not get bored of mundane tasks assigned to it. Most importantly, they can enter hostile environments and safely go where humans cannot go [9].

Autonomous vehicles are already making their presence felt on the battlefield today. Several unmanned surveillance aircraft proved their worthiness on the battlefields of Afghanistan and Iraq. The two most prominent Unmanned Aerial Vehicles (UAVs) are the US Air Force's Predator and Global Hawk.

The Predator is designed to provide constant intelligence, surveillance, and reconnaissance to US forces. The Predator is controlled from a ground control station (GCS), through a satellite link, where it is given commands and mission objects. In October 2001, attacks in Afghanistan demonstrated the Predator's ability for search and destroy missions. While the Predator has the capabilities for combat missions, a human is always kept in the loop to make the final decision to fire on a target. The Predator's ability to attack while removing troops from risk has proved its value to the US military [23].

Another example, the Global Hawk, is a strictly surveillance and reconnaissance UAV. It was built for High-Altitude, Long-Endurance missions. Equipped with sophisticated radar and imagery devices it can "supply responsive and sustained data from anywhere within enemy territory, day or night, regardless of weather." [24]

Autonomous vehicles are also being used in the sea by the Navy and on the ground by the Army. In Iraq, land mines near roads or inside buildings have killed and injured

hundreds of troops. In response, the Pentagon has deployed unmanned mine seeking vehicles. These vehicles have robotic arms that can remotely disable improvised explosive devices. Several drones were damaged while disabling bombs. However, the loss of a robot is well worth the price of saving human lives [5].

Due to the success and benefits of autonomous vehicles, the Pentagon is investing billions into their research and development. The Pentagon proposes to spend \$12 billion on UAVs between 2004 and 2009. By 2009, the armed forces plan to spend \$3.5 billion a year on UAVs. The Army and Navy are also funding numerous autonomous vehicle projects [5].

While many of these vehicles may seem like something out of a science fiction novel, there are still many hurdles that must be overcome. First, the amount of intelligence these vehicles have is still very primitive. Vehicles such as the Predator still require much of their control to be done by humans. To some degree, they can be considered advanced remote control vehicles and not autonomous vehicles. Having unmanned vehicles accomplish more complex goals will require better navigation and control schemes.

Furthermore, different environments provide different challenges. Autonomous air vehicles are by far the most sophisticated autonomous vehicles today. However, they do not face the terrain challenges of a ground vehicle or communication and location problems of an underwater vehicle.

Making a ground vehicle adapt to its environment and avoid obstacles is a daunting task. In the recent DARPA challenge, a contest where autonomous ground vehicles attempted to navigate across an approximately 175-mile stretch of the Mojave Desert, not one of the vehicles competing completed the challenge. The best performer went seven miles.

Also, underwater vehicles are not able to receive GPS signals once submerged and their methods of communication are limited. This serves as yet another navigational challenge that must be overcome. Obviously, there is much more work and research to be done to make these vehicles useful in a military setting.

### **1.3.2 Consumer Applications**

As technology advances and becomes cheaper, more applications of autonomous vehicles will become apparent. As in most cases with technology, what was once a high end, top-secret military application eventually ends up in the consumer market with a completely different function. As an example, the Global Positioning System (GPS) was developed in the 1970s and had a strictly military application. However, today it is used for many other purposes, from cars' navigation systems to mail and package tracking. Similar things will happen with autonomous vehicles and applications are already beginning to emerge today.

Roomba is an autonomous vacuum cleaner invented and manufactured by iRobot. For around \$250, iRobot claims Roomba can keep a house vacuumed with very little human assistance. Roomba hit shelves in the summer of 2002 and has currently sold more than one million units [15].

Another example of a consumer autonomous vehicle is Friendly Robotics' Robomower. Friendly Robotics claims a consumer can get their lawn cut by simply pressing a button. Not only can one schedule when Robomower mows the lawn, but Robomower will also monitor its battery power and find its way back to its charging station without any assistance [11].

Both of these products appear to be great solutions to mundane tasks. However, there are problems with both of them. Unfortunately, in a practical setting these products underperform. They are truly not intelligent devices and make a lot of assumptions in order to simplify the world around them. For instance, both devices navigate through their environment in a random fashion. They move forward until they encounter a wall or obstacle, then they make a turn randomly and begin moving forward again until another wall or obstacle is hit. In a simple environment, such as a rectangular room or circular lawn, with very few obstacles this method may be acceptable. However, in the complex environment of the real world, this method proves to be inefficient and ineffective. Reviews of these products claim that when these products were used in a real world setting they ended up not vacuuming the whole room or leaving strips of grass unmowed.

Obviously, a better system of navigation must be devised for these systems. With all the technology available today, there must be a better, more efficient way to autonomously navigate through a house or around a lawn.

## **1.4 Contributions of this Thesis**

The following contributions were made during this thesis work:

- A graphical simulation environment was created using 3DState.
- A high level navigation and path planning algorithm was implemented and demonstrated using the graphical simulation environment and C++ programming.
- Different vehicle control techniques (Optimal Control, Path Following, Trajectory Tracking) were studied and simulated in MATLAB based on a two-wheeled differential drive robot.
- The physical implementation of an autonomous lawn mower (ALM) including processors, sensors and actuators were researched, discussed and presented.
- This thesis provides full documentation of the project's software and simulations as they exist at the time of this writing.

# 1.5 Organization of this Thesis

This thesis is organized as follows:

- Chapter 2 describes the autonomous vehicle research currently occurring at Virginia Tech. It also outlines the Autonomous Lawn Mower Competition which served as an inspiration for this thesis work
- Chapter 3 investigates autonomous vehicle navigation, specifically for an Autonomous Lawn Mower (ALM). It discusses path planning and obstacle avoidance and proposes a method for covering a bounded region while avoiding obstacles. It then demonstrates this method through a 3D simulation.
- Chapter 4 derives the mathematical model for a two-wheeled differential drive vehicle. It then investigates various control techniques to control the robot's motors and traverse a desired trajectory or path. It examines optimal control theory as well as path following and trajectory tracking techniques.
- Chapter 5 outlines all the hardware that would be used to build the ALM including processors, sensors, and actuators.
- Chapter 6 gives conclusions and possibilities for future work

# Chapter 2

## 2 Autonomous Vehicle Research at Virginia Tech

### 2.1 The Complexity of Autonomous Vehicle Design

An autonomous vehicle is a complex mechanical and electrical system. Designing an autonomous vehicle requires knowledge from many disciplines of engineering, including mechanical, aerospace, electrical, and computer engineering. Usually, mechanical and aerospace engineers study the dynamics of the system. They create mathematical models to analyze the physical characteristics of the system. Modeling a system is a difficult task and can be infinitely complex. Being able to model a system in a simple yet concise way is highly dependent on the system being modeled. System modeling is still a large area of research today.

Electrical engineering is also important when designing and studying autonomous vehicles. Electrical engineers interface the electronics onto the system and design ways to control the system. Once a mathematical model of the system is generated, control techniques can be employed to get the system to act as desired. This again is difficult because many constraints come into play. Not only must a system abide by the laws of physics but it must also meet performance and financial constraints. An autonomous vehicle is laden with expensive sensors and controllers. Getting these devices to work

with one another is also complicated. A thorough knowledge of electronics, sensors, and microcontrollers is definitely required.

Finally, autonomous vehicles require knowledge from computer engineering and computer science. Giving autonomous vehicles intelligence requires complex software. A computer engineer must understand the system they are working with and then translate it into code. Acquiring sensor data, processing that data, and controlling actuators are all done via software. Making this software robust, reliable, and user friendly is a huge design task.

## **2.2 Research and Accomplishments at Virginia Tech**

Since autonomous vehicles require an abundance of knowledge and a variety of engineering backgrounds, Virginia Tech has formed the Unmanned Systems Group. The Unmanned Systems Group blends capabilities from a broad range of disciplines. The research group includes more than 40 faculty members from 12 academic departments, and represents four colleges at Virginia Tech. The group provides expertise in the following

- Design, analysis, and testing of unmanned systems for air, space, water, and land
- Design and implementation of sensors, human-computer interfaces, wireless communication systems and other support technologies
- Scientific, commercial, and military applications of unmanned systems [28]

According to Dr. James Thorp of Virginia Tech, there is currently \$4.2 million in funding for autonomous vehicle research at Virginia Tech and it is projected to grow to \$15.2 million by 2007 [27].

Within the Unmanned Systems Group many interesting applications of autonomous vehicles are being studied. In the Autonomous Underwater Vehicle (AUV) Lab, Dr. Daniel Stilwell is experimenting with the uses of AUVs to better understand the ecological effects of man-made nutrients on Virginia's coastal waters. He is also helping develop search, survey, and tracking methods for AUVs for the U.S. Navy. Dr. Stilwell's goals are to develop control laws so that autonomous vehicles can solve problems cooperatively.

Further autonomous vehicle research is being conducted by Dr. Charlie Reinholtz of the Mechanical Engineering Department. Dr. Reinholtz is developing adaptive navigation strategies for autonomous vehicles. He is also leading research in calibration and test methodologies for unmanned systems and sensors. Dr. Reinholtz sponsors students in

two different autonomous vehicle competitions: the DARPA Grand Challenge and the Intelligent Ground Vehicle Competition.

The DARPA Grand Challenge is a prestigious autonomous ground vehicle competition that was held for the first time in June 2004. The goal of the project is to have a vehicle autonomously navigate itself across an approximately 175-mile stretch of the Mojave Desert. While the basic path of the course is known, the terrain the vehicles encounter along the way serves as a major obstacle to overcome. Large rocks, ditches, bushes, and cliffs all must be avoided autonomously while the vehicles traverse the course [29].

In 2004, the competition offered a \$1 million prize and was open to universities as well as private industry. Virginia Tech was one out of only twenty-five applicants allowed to compete in the competition, out of a total of 106 initial applicants. Also, Tech was one out of only seven contestants to complete the 1.36-mile trial course. Unfortunately, Tech's vehicle malfunctioned shortly into the actual competition. Virginia Tech plans to enter two vehicles in the 2005 competition.

## **2.3 ION Navigation Competition**

The Institute of Navigation (ION) is a non-profit professional society dedicated to the advancement of the art and science of navigation. In 2004, ION held its first ever Autonomous Lawn Mower Competition.

The purpose of the competition was to inspire engineering students at universities to design and operate an autonomous lawn mower using the art and science of navigation to rapidly and accurately mow a field of grass. The idea was simple, and the lawn mower that completed the mowing in the shortest time won the competition [14].

In 2004, three schools from the Midwest competed in the competition and surprisingly none completed the objective. However, due to the competition's success, it is being held again this year. The basic premise of the competition remains the same, however, this year they have added obstacles to the course.

Hopefully, this thesis will serve as a starting point for students who wish to participate in this competition in the future. In writing this thesis and disclosing the ALM's design this work could be leveraged for the competition. This competition would be a great way for Virginia Tech to demonstrate the quality of its engineering program and of its students. Given that there is a significant amount of funding available at Virginia Tech for autonomous vehicle research, hopefully, funding for this competition could be obtained.



# Chapter 3

## 3 Navigation

### 3.1 Overview

Navigation is a huge challenge researchers and engineers must overcome when designing an autonomous vehicle. Navigation is the key to an autonomous vehicle's success and utility. Navigation describes how an autonomous vehicle intelligently moves and interacts with its environment. Navigation can be perceived as the highest level of abstraction of motion control. There are two main focuses when designing a navigation system for an autonomous vehicle: path planning and obstacle avoidance.

Path planning, as the name implies, attempts to generate an intelligent movement scheme for a robot. Path planning is very specific to the application of the robot. The objective of the robot discussed in this thesis is to autonomously mow a lawn. Therefore, its path planning will be optimized to accomplish this goal.

Obstacle avoidance attempts to account for objects the robot may encounter while traversing its path. Having a systematic way for a robot to deal with obstacles creates a more robust design. This makes it more capable of handling real world situations and environments where there are a number of unknowns. When mowing a lawn, a robot not only has to avoid known obstacles such as pools, flower beds, and patios, but it also has to avoid unknown obstacles such as trees, rocks, and debris it encounters. Detecting these obstacles can be handled in a number of different ways by a number of different sensors. This discussion is left to Chapter 5 on hardware implementation. Regardless of

how obstacle detection is handled, having a robot that can accurately avoid obstacles is critical to its success.

## 3.2 Autonomous Lawn Mower Navigation Problem

The purpose of this thesis is to propose a design for an autonomous lawn mower (ALM). Therefore, the ALM's navigation system will attempt to capture the basic behaviors of a regular manned lawn mower. By examining and evaluating the basic behaviors of a regular manned lawn mower an automated version of these behaviors can then be designed.

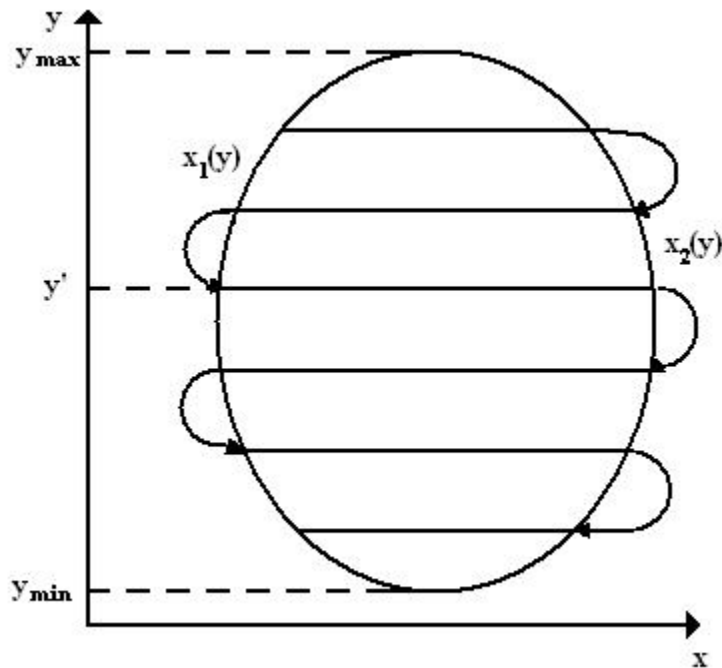
When humans mow a lawn they are actually attempting to completely cover a bounded region. They push a lawn mower across their lawn until they have traversed its entire area. Normally, humans attempt to do this in the most efficient way in order to save time and the energy of both the gas required to run the lawn mower and the physical labor they exude to push the mower. When a lawn mower travels across a stretch of grass it has already mowed it is wasting time and energy. Obviously, a preferred path is the one that minimizes the amount of overlap in the path the lawn mower takes. Therefore, minimizing the ALM's path overlap is a key issue when designing its navigation system.

This type of problem has been extensively studied in mathematics and navigation. It is known as a region filling problem and has many applications. Applications of this problem can be seen in search and rescue operations, milling, spray painting, and optical inspection. Unfortunately, the constraints of this type of problem are highly dependent on the intended application and therefore no generic solution exists.

The best region filling solution that is applicable to the lawn mowing problem was proposed by Cao [6]. In this implementation the authors attempt to move a robot across the area of a bounded region. The method they suggest is referred to as "strip filling". In this procedure "an elemental region is bounded by an arbitrary closed convex curve within which there are no obstacles."

Geometrically it can be stated that a connected region  $R$  has the form suggested by Figure 3.1. Suppose that for  $y_{\min} \leq Y \leq y_{\max}$  the line  $Y=y'$  intersects  $R$  in an interval or coincides with the boundary. Call  $R$  an elemental region and indicate the  $x$  direction for the elemental region filling. The process of strip filling may be indicated by an iterated integral [6]

$$\int_{y_{\min}}^{y_{\max}} dx \int_{x_1(y)}^{x_2(y)} dy = \iint_R dx dy = S$$



**Figure 3.1:** Filling an elemental region

The value of the integral is then equal to the area of the region. Therefore, the strip filling operation can continuously and entirely fill up a region along the elemental filling direction. Conveniently, this operation involves a series of straight-line strips, which are a finite width apart, and a sequence of turns. This paper was a good starting location for the lawn mower navigation problem. However, it still had some practical limitations that needed to be expanded upon [6].

Unfortunately, Cao only considers closed regions formed by convex curves. For the lawn-mowing problem this is a severe practical constraint because it is unlikely that a normal yard can be represented by just convex curves. A navigation technique had to be devised that would allow the ALM to cover “any” bounded region while avoiding “any” bounded obstacles.

In summary the requirements of the ALM’s navigation system is to,

1. Discover and map the lawn it is placed in
2. Move through an entire area covering the whole bounded region
3. Discover, map and avoid all obstacles it encounters online
4. Move in a continuous and sequential way
5. Attempt to use simple motion trajectories such as straight lines
6. Attempt to minimize overlap

## 3.3 Region Filling with Random Obstacle Avoidance

Before examining the implementation of the ALM's navigation system, a few assumptions must be stated and some terminology must be defined to clarify the subsequent work. The following implementation made the assumption that the ALM was operating on flat ground. Therefore, the ALM can represent and map its environment with a 2D coordinate system.

Furthermore, it is assumed that the lawn and any obstacles the ALM encounters can be represented by closed curves. Therefore, any boundary or obstacle the ALM detects can be traced in a circular fashion. By starting at any point on the boundary the ALM can move forward and eventually make its way back to its initial starting position.

Finally, it is assumed that the ALM interacts with only two types of boundaries. The outside border, as it is referenced in this thesis, refers to the outside perimeter of the lawn. It is assumed that there is only one unique outside border and at all times the ALM will stay within this boundary.

Also, there are references to internal borders. These are obstacles or boundaries placed inside the lawn that the ALM is required to stay out of. For the purpose of this thesis there was no distinction between whether these internal borders represented known obstacles such as a flowerbed or if they represented unknown obstacles such as rocks and trees. In actuality either of these cases, from a navigation stand point, can be handled the same way. The issue of determining whether it is a known or unknown obstacle is dependent on the hardware implementation discussed in Chapter 5.

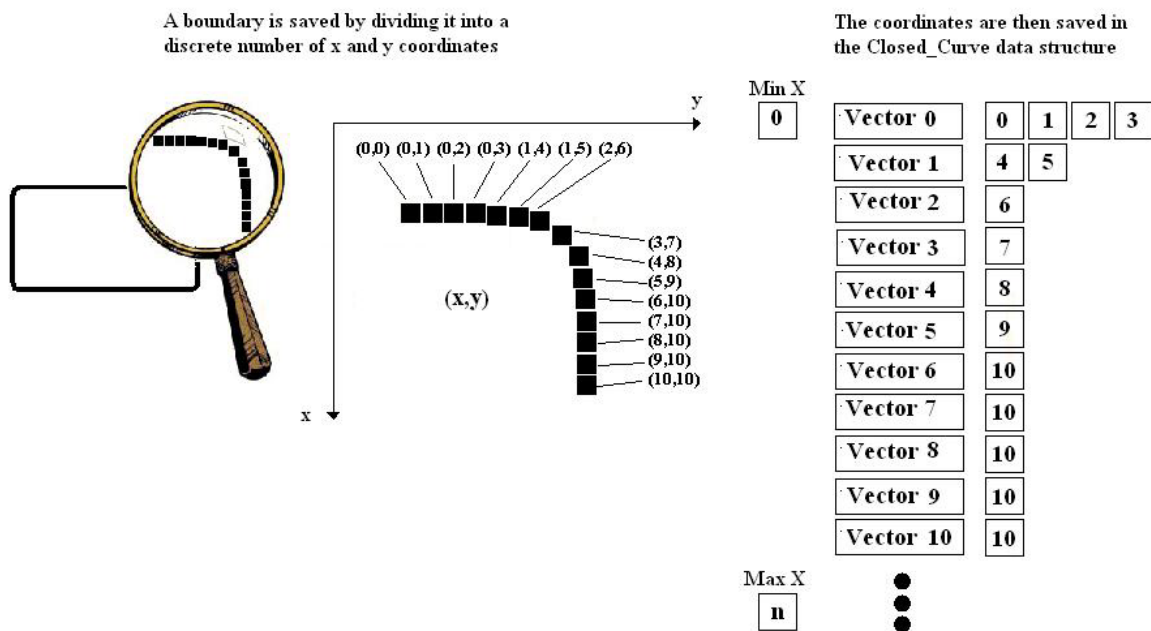
### 3.3.1 Behaviors and Data Structures

Various behaviors had to be defined in order to meet the desired goals of the ALM's navigation system. The ALM's navigation system required three behaviors: EXPLORE, TRACE\_BORDER, and CALC\_PATH. Also, since the ALM mapped its environment it also needed memory to store mapping information. A convenient data structure had to be defined to store and retrieve this data. Since memory is considered a limited resource, mapping had to be done in an efficient and conservative manner.

EXPLORE, as the name implies, is a behavior the ALM uses to discover the environment it is placed in. EXPLORE simply moves the ALM forward in a straight line until it encounters a border. EXPLORE is the simplest command the ALM uses. It requires no knowledge of the ALM's environment and requires no memory usage. EXPLORE's only function is to get the ALM to a border so it can start mapping its environment.

TRACE\_BORDER is basically a path following function. Once the ALM discovers a border, and since it is assumed that all borders are closed curves, the ALM moves around the border until it again reaches the place where it started. The ALM records its x and y coordinates as it traverses along the border. By storing only the border points the ALM can recreate a map of the environment it is in. However, storing mapping information creates the need for the data structure, Closed\_Curve.

Closed\_Curve is a data structure that is used to represent the borders of the environment. As can be seen in Figure 3.2, it is an array of vectors that holds the y coordinates of the border points. It also keeps track of the maximum and minimum x coordinate. By only saving the maximum and minimum x coordinate and every y coordinate there is enough information to uniquely characterize a border.



**Figure 3.2:** Graphical Representation of Closed\_Curve Data Structure

Consider the example where the ALM would map an acre plot of land with centimeter resolution. Each point, an x and y coordinate, could be represented by 1 byte of memory and could indicate whether the ALM could traverse that region. Given that an acre is approximately 40,468,564 square centimeters, this mapping method requires approximately

$$(40,468,564 \text{ cm}^2 * 1 \text{ byte}) / 2^{20} \approx 38.6 \text{ MB.}$$

In comparison, storing only the perimeter information, assume the above mentioned acre is square and there are no obstacles within it. Also note that a single side of a square acre is 6361 cm in length and an integer is represented by 4 bytes. Therefore, by only storing

border information using the Closed\_Curve data structure, only 99 KB of memory is required.

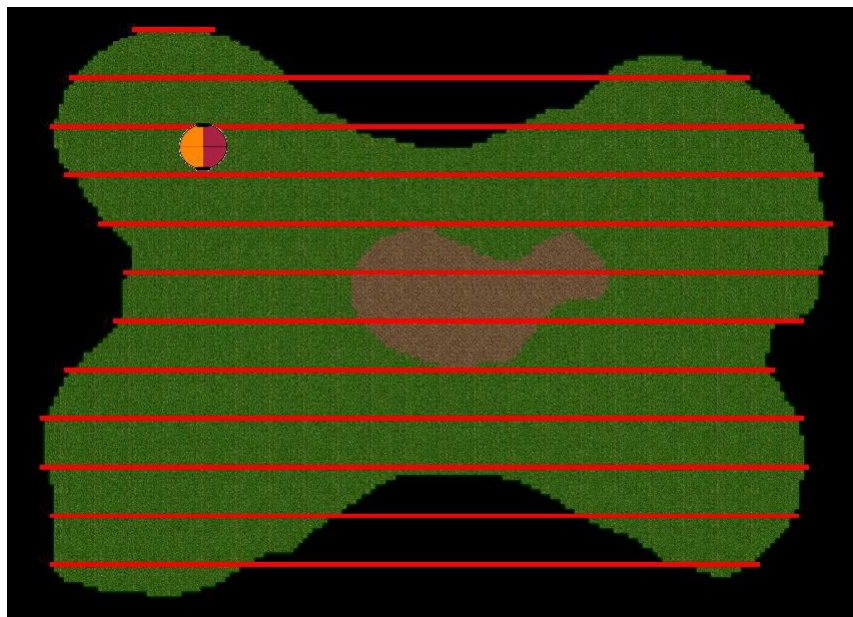
$$((6361 \text{ cm} * 4 \text{ sides} * 4 \text{ bytes}) + 8 \text{ bytes}) / 2^{10} \approx 99 \text{ KB.}$$

This method of mapping stresses memory conservation, yet still maintains adequate performance. Instead of storing every point on the map the ALM only stores the borders it encounters. Despite storing only this limited amount of information, the whole 2D map can still be recreated. This leads to significant memory savings while only requiring a small amount of extra calculation.

The final behavior, CALC\_PATH, calculates the path the ALM should take through the lawn based upon the border information the ALM currently has. CALC\_PATH requires that the ALM have knowledge of at least the outside border. If the ALM has knowledge of any existing internal borders this information is also included in the ALM's path calculation. As additional information is acquired by the ALM, CALC\_PATH can be run again to update its path plan.

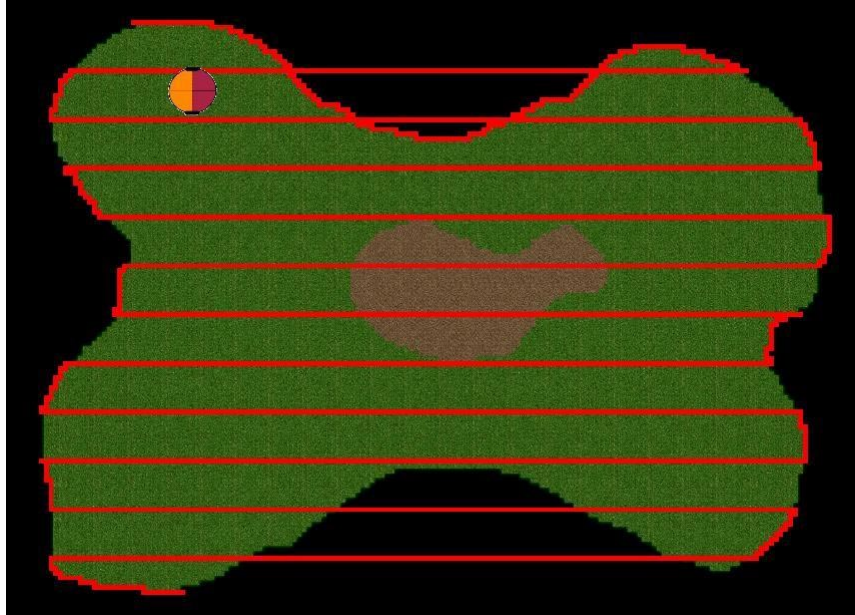
The CALC\_PATH behavior implements the following logic,

1. CALC\_PATH begins by taking the outside border and dividing it into sections based on the the ALM's mower blade diameter. This action is similar to the "strip filling" procedure discussed in the previous section. These sections are represented as a series of points that form straight lines across the outside border.



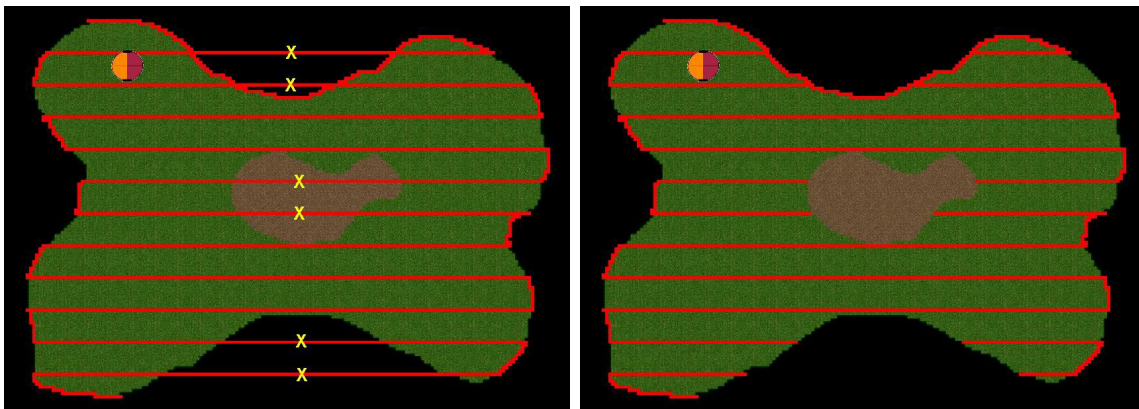
**Figure 3.3:** OPTIMAL\_PATH Step 1

2. These lines are then connected together, from top to bottom using the outside border points. The connections are made, switching between connecting the tails of consecutive lines together, to connecting the heads of consecutive lines together.



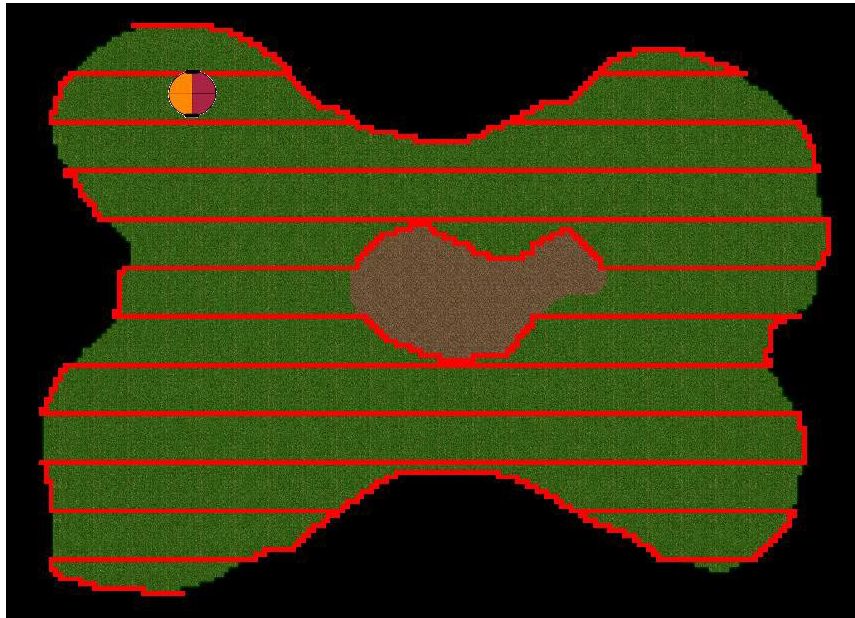
**Figure 3.4:** OPTIMAL\_PATH Step 2

3. Once the connections are made, the resulting path is scanned through and erroneous points are filtered out. Points that do not belong in the path include points that are:
  - Outside the Outside border
  - Inside a known Internal border



**Figure 3.5:** OPTIMAL\_PATH Step 3

4. The points that were removed in the previous step are then replaced by the section of border that those points intersected with.



**Figure 3.6:** OPTIMAL\_PATH Step 4

CALC\_PATH has the ALM traverse the lawn from top to bottom and generates a unique path through the bounded region. Given the same mapping information, the same path is always generated using CALC\_PATH. When the ALM discovers a new obstacle, CALC\_PATH can be called again. Once it calculates the new path, the ALM inherently knows the portion of the lawn it has already mowed, based solely on its current position. Therefore, the ALM does not need to store in memory the places on the map it has already been.



### 3.3.2 Autonomous Lawn Mowing

Autonomous lawn mowing consists of using the above-mentioned behaviors in a systematic way. To autonomously mow a lawn the following flow chart is followed

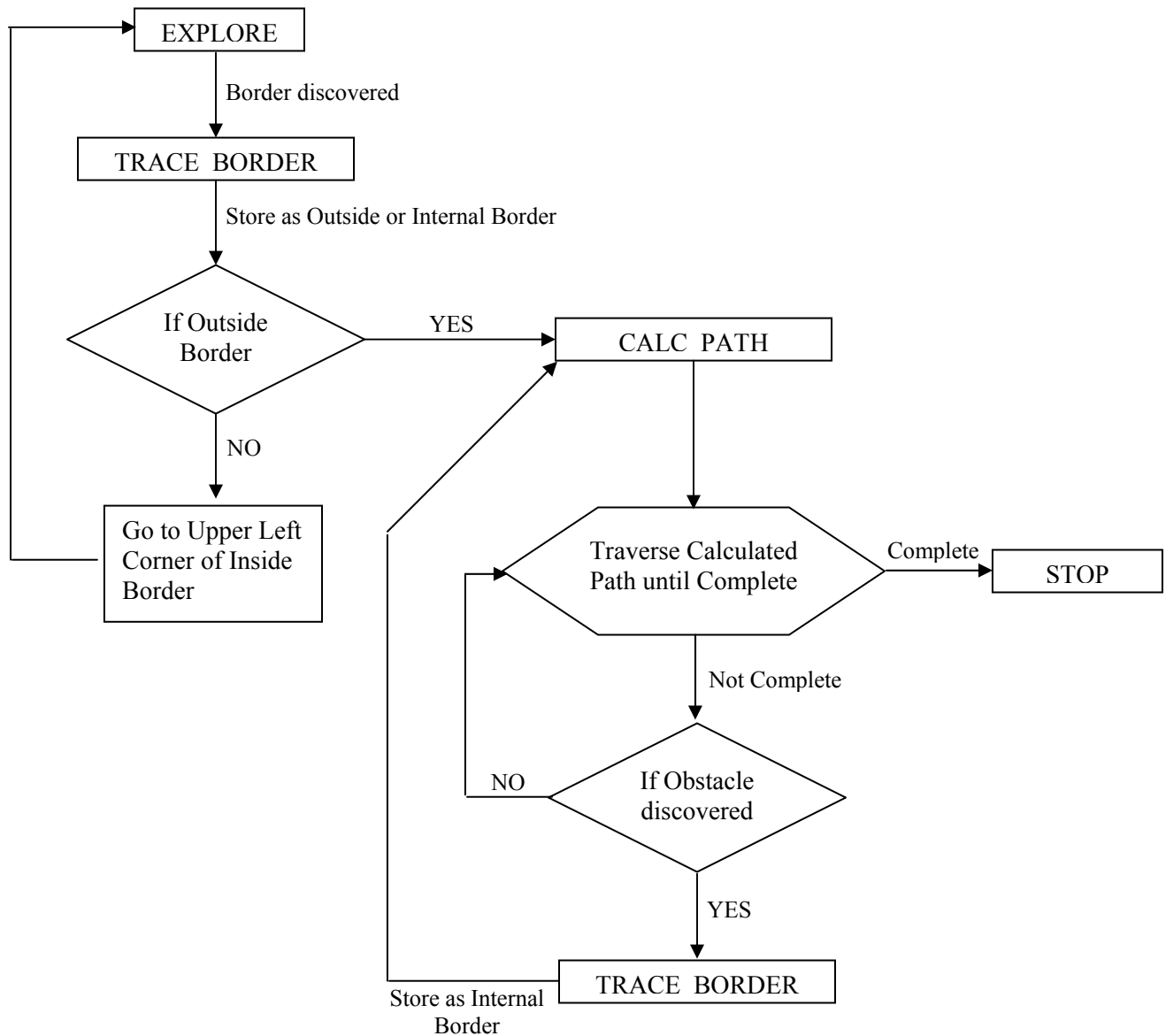


Figure 3.7: Flow chart for Autonomous Lawn Mowing

## **3.4 Simulation**

### **3.4.1 3DState's 3D Graphics Engine**

The simulation environment used to implement and test the ALM's navigational algorithm was created using 3DState's graphics engine. A graphics engine is a software library that allows a programmer to create a 3D environment where 3D objects can be displayed, moved, and rotated. Graphics engines are useful because they hide the ugly details of 3D API programming without sacrificing the performance. They create a higher abstraction and supply more robust and versatile 3D graphic functions. This in turn decreases the learning curve and development time for 3D graphics programming [2].

3DState's graphics engine was chosen due to its cost and performance. 3DState offers a free version of its 3D engine as long as it is used for noncommercial purposes. Also, 3DState's graphic engine is known for good performance and its high frames per second rate.

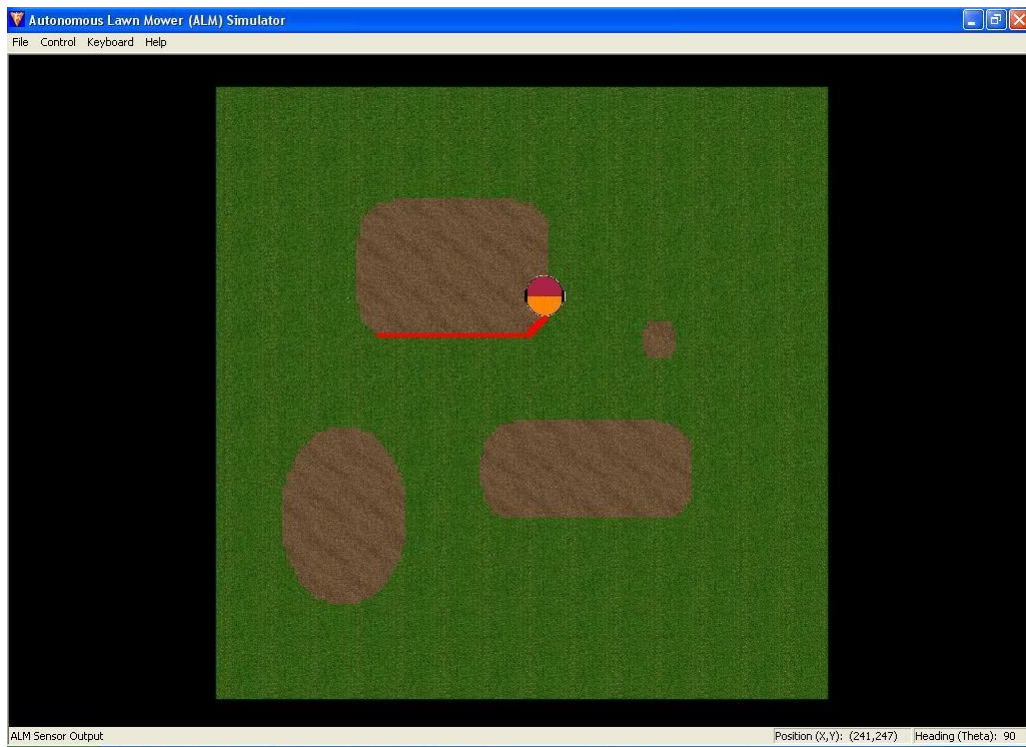
The 3DState graphics engine had all the required functionality to create the ALM simulation environment. 3DState's software development kit (SDK) for C++ greatly reduced the ALM simulation development time. Its 3D function calls were easy to use and understand. Also, the programming guide was a great reference and gave numerous examples on how to use the engine [2]. 3DState's graphic engine was very flexible and allowed the programmer to have access to all levels of abstraction. In creating the ALM simulation environment, one could work as low as the byte and bitmap level just as easily as working with polygon and object level.

### **3.4.2 Simulation Environment and Results**

The screen shots below are included as proof of the implementation of the ALM's navigational algorithm. Figures 3.8-3.11 depict the basic behaviors of the ALM.



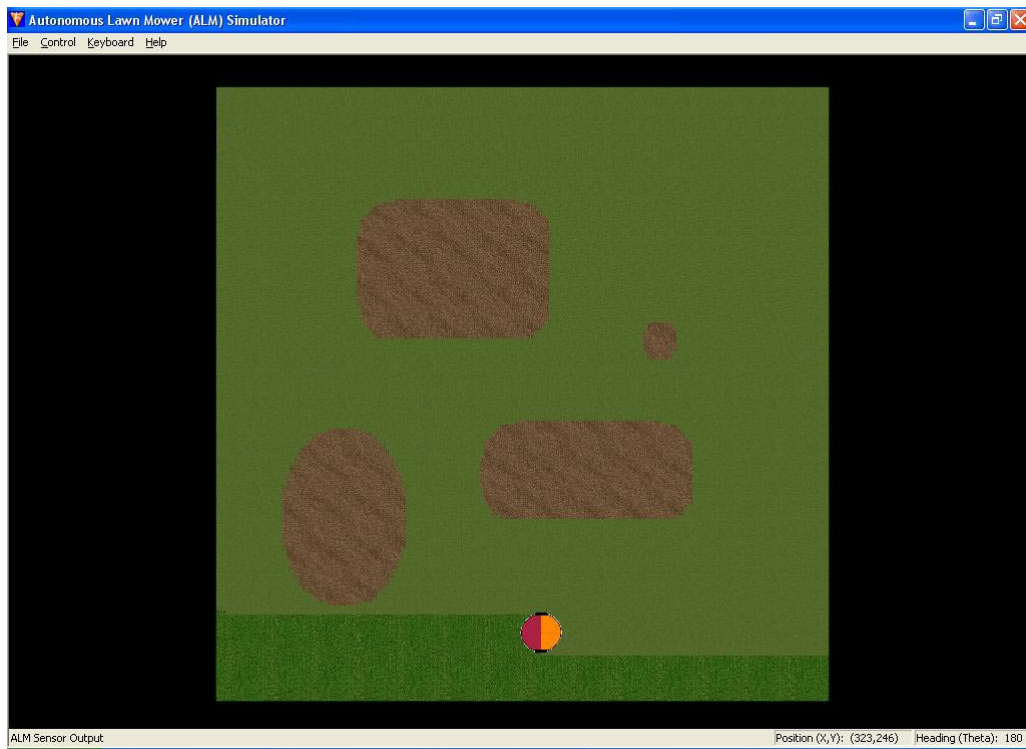
**Figure 3.8:** The ALM exploring its environment and searching for a boundary



**Figure 3.9:** The ALM tracing an unknown internal border



**Figure 3.10:** The ALM traversing its calculated optimal path



**Figure 3.11:** The ALM completing its traversal of the lawn

The ALM's navigational algorithm effectively generates a path plan that covers any bounded region with any number of bounded obstacles. Several maps were input into the simulator to test various aspects of the navigation system. To date, no problems with the functionality of the algorithm have been discovered. The ALM has been able to traverse any bounded region containing any number of bounded obstacles. However, it should be noted that the efficiency of this algorithm is highly dependent on the geometry of yard and obstacles. The following section will address this issue. Furthermore, whether the ALM is mechanically capable of traversing these paths is a control and hardware issue and left to the subsequent chapters for discussion.

### 3.5 Efficiency of the ALM's Navigation System

Efficiency is a key issue when designing a navigation system. In regards to lawn mowing, efficiency relates to the amount of overlap that occurs in the ALM's path plan. If at any time the ALM's traverses an area where it has already been, it is wasting time and energy. In order to mow a lawn efficiently this overlap should be minimized. This type of problem is known as coverage path planning [7].

After examining the implementation of the ALM's navigation system, it becomes apparent that the ALM's efficiency is dependent on the geometry of the yard and the obstacles it encounters. Consider the following yard,

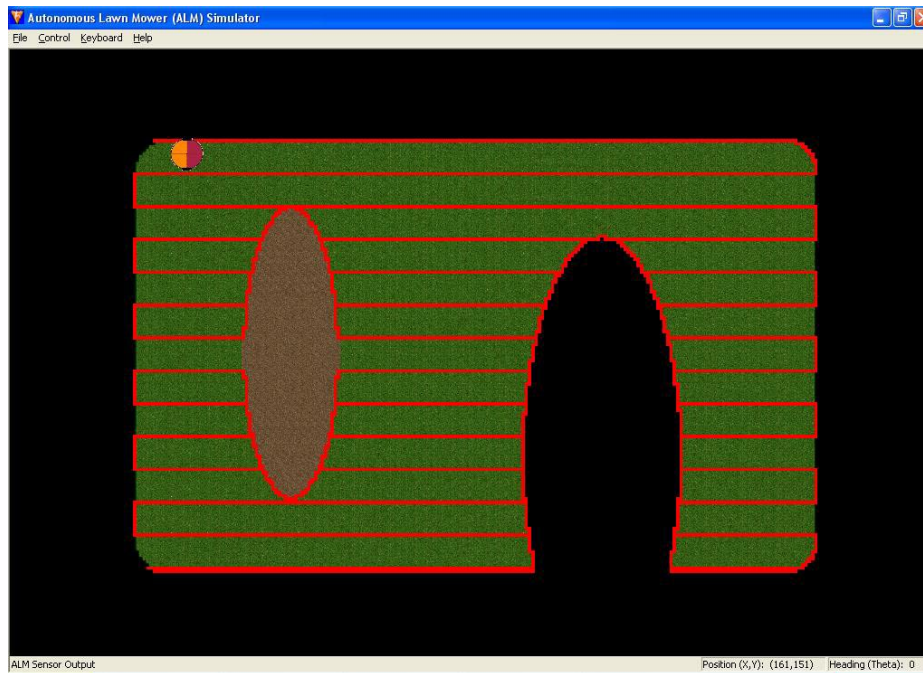
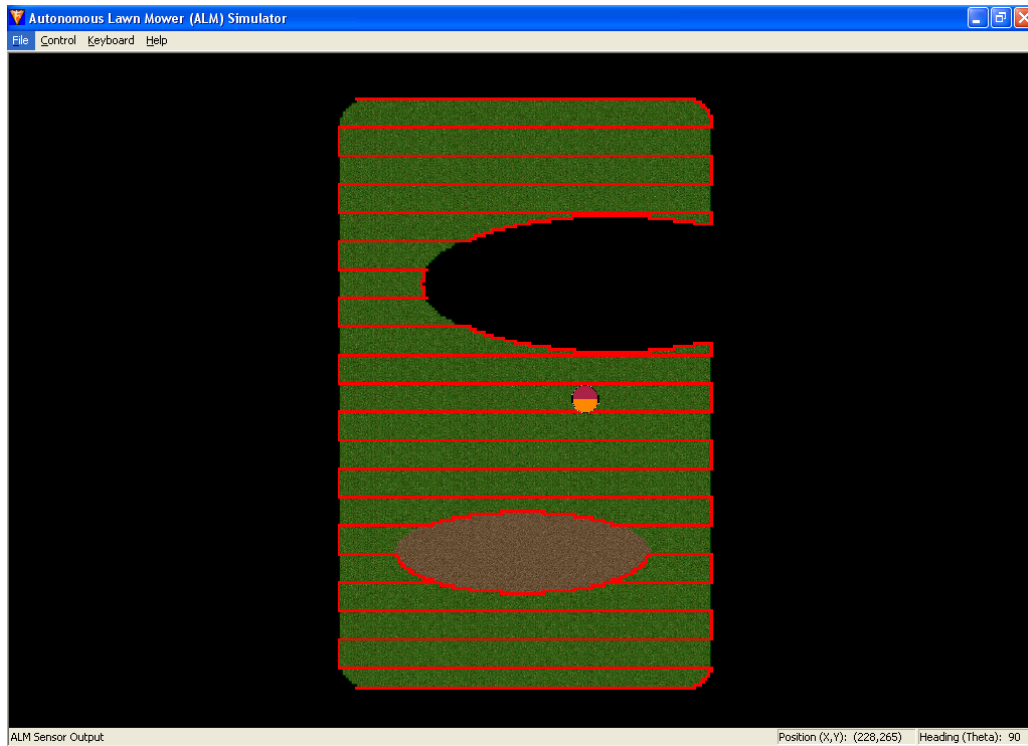


Figure 3.12: Inefficient Yard Geometry

As can be seen from Figure 3.12, this particular yard would cause a great deal of inefficiency for the ALM. Since the ALM's navigation system is limited to traveling from the top to the bottom of the yard and in the x direction, the ALM must trace the concave section of the boundary several times. As can be seen, this extreme concavity in relation to the cutting direction produces a great deal of overlap in the path plan. A similar situation occurs when obstacles stretch across the cutting direction. This overlap is related to the height of the obstacle relative to the cutting direction as can be seen in Figure 3.12.

A simple solution to this problem would be to rotate the map by  $90^\circ$ . This could be done by taking the transpose of the map's x and y coordinates. Examining Figure 3.13, the concavity of the boundary no longer exists relative to the cutting direction and therefore causes minimum overlap in the path plan. Furthermore, the path overlap due to the obstacle is minimized because its height relative to the cutting direction has been minimized.



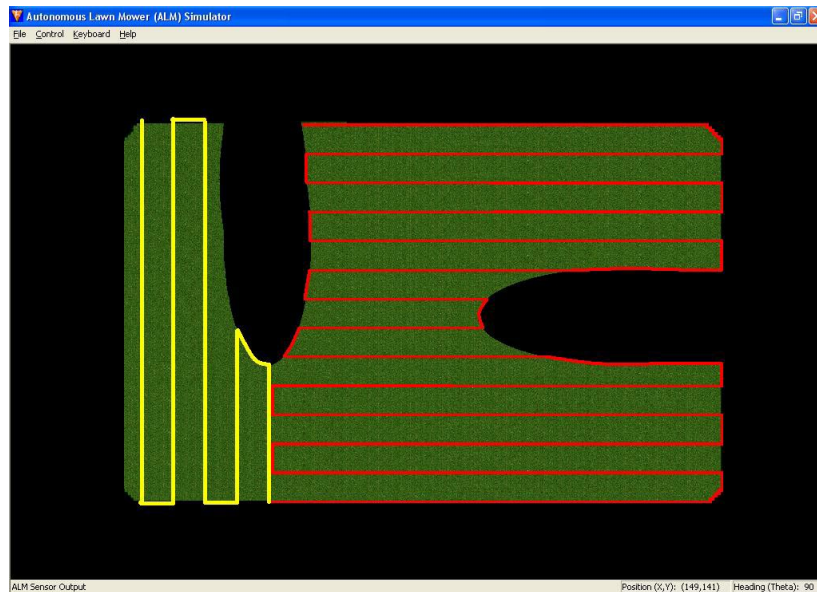
**Figure 3.13: Efficient Yard Geometry**

To ensure greater efficiency in the path plan, a more complex algorithm could be created in which the map of the yard is rotated by varying degrees through a total of  $180^\circ$ . The map rotation that produced the minimum amount of path overlap could then be chosen as the best.

Unfortunately, as Choset [7] states, any coverage algorithm “may never yield an optimal solution for all environments because one can always create an antagonistic example for any approach.” As can be seen in Figure 3.14, there is no acceptable way to rotate this map and significantly reduce path overlap. However, as can be seen in Figure 3.15, if there were a method of dividing this map into two smaller subregions the path overlap could be avoided all together.



**Figure 3.14:** Yard Geometry in which map rotation does not minimize path overlap



**Figure 3.15:** Dividing a yard into subregions

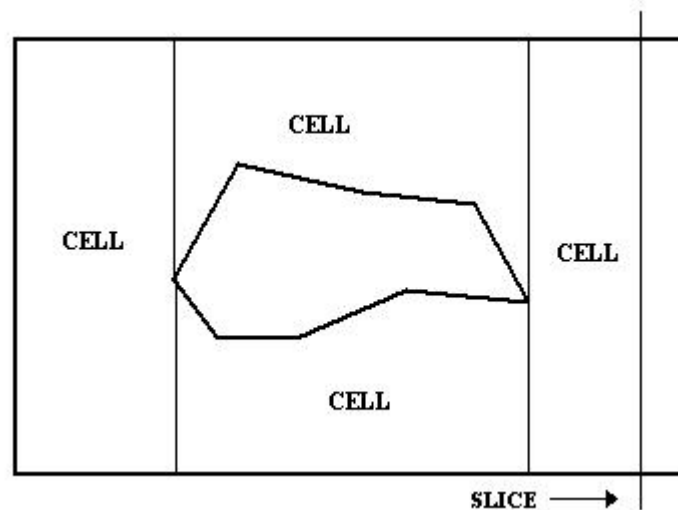
### 3.5.1 Exact Cellular decomposition

Recent research in coverage path planning has yielded coverage methods that are classified as exact cellular decompositions. An exact cellular decomposition is a set of non-intersecting regions, each referred to as a cell, whose union fills the target environment. Typically, it is assumed that the robot can cover each cell using simple back-and-forth motions, as was demonstrated by the ALM's navigation system. Therefore, using these methods, coverage path planning reduces to planning motions from one cell to another [7].

#### 3.5.1.1 The Boustrophedon Decomposition

The Boustrophedon decomposition is a method of subdividing a map into subregions, each of which are easily traversable by back-and-forth motions. The word Boustrophedon was first used by the English in 1699 and literally means the "way of the ox". The name of the method adequately describes the method this algorithm uses to cover a bounded region. The Boustrophedon decomposition was first proposed by Choset and Pignon [8].

The Boustrophedon decomposition defines a slice, as a perpendicular line segment that is swept across the map in the cutting direction. New cells are created or destroyed based upon the connectivity of the slice. Two new cells are created when the connectivity increases. Similarly, two cells are merged into one cell when the connectivity decreases [7]. A graphical interpretation of this method can be seen in Figure 3.16.



**Figure 3.16:** Boustrophedon decomposition

Once a region has been subdivided into cells, it is then possible to form an adjacency graph for the cells. By then using a graph search algorithm each cell can be visited and the whole region can eventually be covered [7]. A well known computational geometric



method for optimally visiting each cell in the bounded region is known as the Traveling Salesman Problem (TSP) and is discussed in the following section.

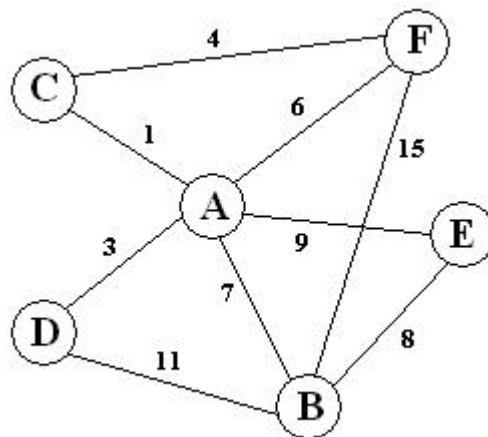
It should be noted that the Boustrophedon decomposition is an offline method and requires full knowledge of the yard and any obstacles within the yard. This violates the desire of the ALM's navigation system to plan its path online as the ALM traverses the lawn.

However, when practically evaluating a yard, it becomes apparent that a yard is a fairly static environment. The main features of the lawn, the outside boundary, and most internal borders remain constant. Few drastic changes occur to a yard over the life time of a lawn mower. Therefore, the ALM could employ the Boustrophedon method after it had initially mowed the lawn. This could drastically improve the efficiency of the ALM's navigation system for all subsequent lawn mowing sessions.

### 3.5.1.2 The Traveling salesman problem (TSP)

The Traveling salesman problem is a classic computational geometry problem in which it is desired to find the shortest path between a set of cities in which each city must be visited once. It is well known that this type of problem is NP-complete [31]. This problem can also be represented using graph theory and is shown in Figure 3.17.

In Figure 3.17, each node represents a place that must be visited, while the edges represent viable paths between the nodes. Usually, there is a weight associated with each edge that represents the cost of travel from one node to another.



**Figure 3.17:** A graph composed of nodes and edges

Given that the ALM's map can be divided into subregions that are easily traversable by back-and-forth motions using the Boustrophedon decomposition, the Traveling Salesman Problem then becomes apparent. It is desired that the ALM move from subregion to subregion in an optimal way.

To formulate this problem each subregion generated by the Boustrophedon decomposition can be represented by a node in a graph. Adjacent subregions will contain an edge connecting the two nodes. Finally, a weight is assigned to an edge that reflects the relative distance between two subregions. This representation adequately characterizes a TSP.

Extensive research has been done in order to arrive at solutions for TSP's. TSP's are highly complex problems requiring exponential time to solve in relation to the number of nodes in the problem.

Most lawns could likely be broken down into only a few subregions using the Boustrophedon decomposition. In this case a brute force method could be used to solve the inherent TSP. Using this method,  $n!$  different combinations would have to be checked to find the optimal traversal of the lawn. Therefore, this type of problem is solved in  $O(n!)$  time [31].

This method could be improved upon by using Dynamic programming. Using dynamic programming it has been proven that the problem can be solve in  $O(n^2 2^n)$  time. While this is better than  $O(n!)$  it is still an exponential solution and is only acceptable for lawns that would have a relatively small number of subregions [31].

Given a more complex lawn with several hundred subregions, other methods would have to be employed. There are numerous numerical techniques used to solve these complex cases and they usually involve heuristics or approximations of the problem.

# Chapter 4

## 4 Mathematical Modeling and Control

### 4.1 Mathematical Modeling

#### 4.1.1 Overview

Mathematical modeling is an important concept when designing any system. Mathematical models attempt to describe the physical world in a concise way. In the real world there are an infinite number of variables that affect the outcome of any situation. Models simplify the real world by abstracting and using only the most pertinent variables.

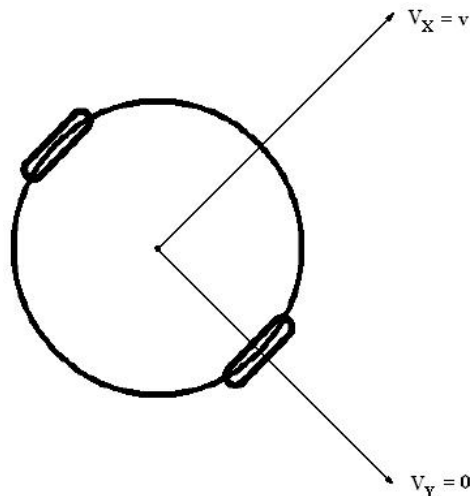
Once a model is created it can then be used to predict outcomes or behaviors of the system being modeled. How accurately a model predicts a system's physical behavior is a measure of how good the model is. Accuracy is usually directly proportional to how complex the model is. The goal when modeling a system is to use the least amount of complexity to still accurately portray the system's behavior.

Researchers are constantly looking for simple yet accurate models of systems. The complexity of the model also relates to the amount of computation required to solve the model. Mathematical models can require anything from calculators to supercomputers to solve. In some instances, the models cannot be solved at all.

In this chapter the dynamics of a two-wheeled differential drive mobile robot are derived and examined. Using this model, various control techniques can be employed to design a controller that can control the motion of the robot. Throughout this work the kinematic model is used. This type of model allows for the decoupling of vehicle dynamics from its movement. Therefore, dynamic properties such as mass, center of gravity, and moments of inertia do not influence the model. To derive this model, the nonholonomic constraints of the system are utilized.

## 4.1.2 Nonholonomic Constraints

Nonholonomic constraints occur in mechanical systems when a system has limitations on its velocity but no limitations on its position. The best example of this occurs while parallel parking a car. Pulling up to a parking spot, the vehicle is unable to slide sideways into the spot. However, by moving the car forwards, backwards, and turning the wheels, it is able to move into the spot. Simply put, the car's local movement is limited, however, its global position is unrestricted. In mathematical terms, this means that the vehicle's velocity constraints cannot be integrated to position constraints [21].

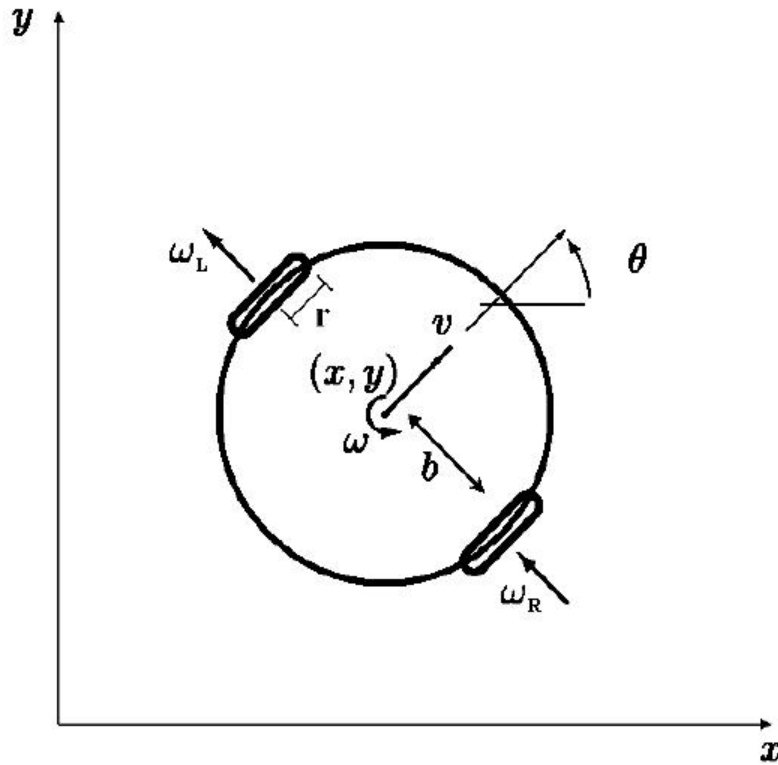


**Figure 4.1:** Nonholonomic constraint of a two-wheeled differential drive robot

Examining Figure 4.1 above, we see the nonholonomic constraint of a two-wheeled differential drive robot. Obviously, the robot's velocity is limited to the direction the wheels are facing. Assuming no slippage, the robot is not able to move sideways. However, by turning its wheels in opposite directions the robot can rotate in place, thus allowing the robot to move sideways. Therefore, the robot's global position is unrestricted.

### 4.1.3 Global Coordinate Model

The important properties of a two-wheeled differential drive mobile robot operating on a planar surface can be seen in Figure 4.2 [13].



**Figure 4.2:** Kinematic Global Position Model of a two-wheeled differential drive robot

In Figure 4.2,  $x$  and  $y$  denote the position of the center of the axle of the robot with respect to the global coordinate frame,  $\theta$  denotes the heading of the vehicle with respect to the  $x$  axis in the global coordinate frame. The variable  $r$  is the radius of the robot's wheels and  $b$  is the half distance between the two wheels. Finally,  $v$  and  $\omega$  are the linear and angular velocity of the robot, respectively.  $\omega_L$  and  $\omega_R$  are the rotational velocity of the left and right wheels.

The global coordinate model is one of the simplest and most minimalistic representations of a two-wheeled mobile robot. In this model, the robot can be represented by its three degrees of freedom, with the vector

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

From Figure 4.1, it becomes apparent that this type of robot has three positional degrees of freedom. However, it can instantaneously move in only two directions due to the nonholonomic constraint. This constraint exists due to the fact that the robot cannot have a velocity parallel to the direction of its axle. Therefore, the robot is capable of moving with a velocity  $V_X$  in its body fixed x direction and rotating with an angular velocity  $\omega$ , but it cannot have a velocity  $V_Y$  in its body fixed y direction. This no-slip nonholonomic constraint can be mathematically modeled and derived as follows [13]

$$\begin{aligned}\dot{x} &= V_X \cos \theta - V_Y \sin \theta \\ \dot{y} &= V_X \sin \theta + V_Y \cos \theta\end{aligned}$$

Noting that  $V_Y = 0$ , the above expressions for x and y can be simplified

$$\begin{aligned}\dot{x} &= V_X \cos \theta \\ \dot{y} &= V_X \sin \theta\end{aligned}$$

Therefore,

$$\begin{aligned}\frac{\dot{x}}{\cos \theta} &= \frac{\dot{y}}{\sin \theta} \\ \dot{x} \sin \theta &= \dot{y} \cos \theta\end{aligned}$$

Thus, the no-slip nonholonomic constraint is

$$\dot{x} \sin \theta - \dot{y} \cos \theta = 0 \quad (4.1)$$

Furthermore this model neglects external forces such as friction on the wheels or the possibility of wheel slippage. Due to this assumption the robot is said to satisfy the pure rolling condition

$$\dot{x} \cos \theta + \dot{y} \sin \theta = v \quad (4.2)$$

From these two constraints, equations 4.1 and 4.2, the robot's kinematic behavior can be derived as

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (4.3)$$

Unfortunately, the linear and angular velocity of the robot,  $v$  and  $\omega$ , are usually not readily available from sensor data. However, a much easier measurement to obtain is the robot's wheel velocities,  $\omega_L$  and  $\omega_R$ , through the use of wheel encoders. The relationships between  $(\omega_L, \omega_R)$  and  $(v, \omega)$  are

$$v = \frac{\omega_L + \omega_R}{2} r \quad (4.4)$$

$$\omega = \frac{\omega_R - \omega_L}{2b} r \quad (4.5)$$

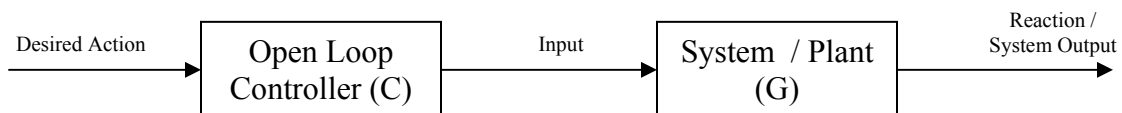
Plugging equations 4.4 and 4.5 into equation 4.3, a more useful robotic model can be derived as

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} r \cos \theta & \frac{1}{2} r \cos \theta \\ \frac{1}{2} r \sin \theta & \frac{1}{2} r \sin \theta \\ -\frac{r}{2b} & \frac{r}{2b} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \quad (4.6)$$

After obtaining the model of a two-wheel differential drive robot its position and heading are now related to its left and right wheel velocities. In order to obtain positioning information this model must be integrated and solved. To control this model, numerous mathematical techniques exist.

## 4.2 Open Loop Control

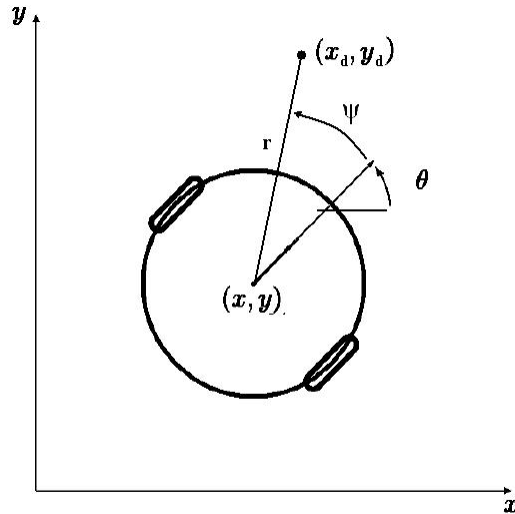
Open loop control is the most basic form of control and is usually the easiest to understand. An open loop controller supplies a series of inputs to a system and assumes the system will behave appropriately. The open loop controller does not take into account the actual consequences of its inputs or adjust its inputs due to external disturbances to the system. Open loop control can be a practical solution when the system being controlled is modeled very accurately and the system is not vulnerable to external disturbances. A block diagram of the open loop control paradigm can be seen in Figure 4.3 below.



**Figure 4.3:** Open Loop Control Paradigm

## 4.2.1 Inverse Kinematics

One of the simplest ways to control a two-wheel differential drive robot is to place further constraints on its movement. By only allowing the robot to either turn about its midpoint or travel in a straight line, inverse kinematic equations can be derived. Figure 4.4 displays the pertinent variables.



**Figure 4.4:** Inverse Kinematics of a two-wheel differential drive robot

By having the robot rotate to a desired heading and then travel in a straight line, coordinate frames can be used to describe this rotation and translation. Having the robot rotate about its midpoint is the same as a rotation about the z-axis. This rotation of  $\theta$ , can be described mathematically as

$$Rot(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Furthermore, a translation of the robot in the x or y direction can be mathematically describe as

$$Tran(x, y) = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, a robot that wishes to traverse to a new position will rotate  $\psi$  in its local frame, and then move a distance r in its local x direction. This is mathematically described as



$$Rot(\psi)Tran(r,0) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & r \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \psi & -\sin \psi & r \cos \psi \\ \sin \psi & \cos \psi & r \sin \psi \\ 0 & 0 & 1 \end{bmatrix}$$

However, since the robot has an arbitrary starting position and heading in its global coordinate frame, this initial rotation and translation must be taken into account. This motion can be described by a rotation,  $\theta$ , followed by a translation,  $x$  and  $y$ , along the global  $x$  and  $y$  axis. Mathematically it follows

$$Rot(\theta)Tran(x,y) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & x \\ \sin \theta & \cos \theta & y \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, by equating these results, the robot's desired final position and rotation is obtained. By just examining the robot's final position and ignoring the robot's final orientation, only two equations and two unknowns must be solved.

$$\begin{bmatrix} \cos \theta & -\sin \theta & x \\ \sin \theta & \cos \theta & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \psi & -\sin \psi & r \cos \psi \\ \sin \psi & \cos \psi & r \sin \psi \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} ? & ? & r \cos \psi \cos \theta - r \sin \psi \sin \theta + x \\ ? & ? & r \cos \psi \sin \theta - r \sin \psi \sin \theta + y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} ? & ? & x_d \\ ? & ? & y_d \\ 0 & 0 & 1 \end{bmatrix}$$

$$r \cos \psi \cos \theta - r \sin \psi \sin \theta + x = x_d$$

$$r \cos \psi \sin \theta - r \sin \psi \sin \theta + y = y_d$$

These equations can then be solved for  $\psi$ , the amount the robot should turn from its current heading

$$\frac{\sin(\psi + \theta)}{\cos(\psi + \theta)} = \tan(\psi + \theta) = \frac{y_d - y}{x_d - x}$$

$$\psi = \tan^{-1}\left(\frac{y_d - y}{x_d - x}\right) - \theta \quad (4.7)$$

and  $r$ , the distance the robot needs to travel

$$r \cos(\psi + \theta) = x_d - x$$

$$r \sin(\psi + \theta) = y_d - y$$

$$r = \sqrt{(x_d - x)^2 + (y_d - y)^2} \quad (4.8)$$

While deriving the inverse kinematics is fairly simplistic, it also places impractical constraints on the robot. Given this solution, the robot's movements are limited to turning in place and traveling in a straight line. This method would not allow the robot to move along a curved path. Also, if the robot veered off its desired path due to an external disturbance or an internal mechanical irregularity it would then have to stop, recalculate its desired heading, turn in place, and then continue in a straight line. This would make the robot's movements very choppy and likely place unneeded mechanical stress on the robot.

Evaluating the navigational scheme presented in Chapter 3, it becomes apparent that using the inverse kinematics is not a practical solution. To have a robot accurately navigate in an outdoor environment, namely a lawn, the robot must be able to travel in straight as well as curved trajectories.

Deriving a more practical solution for the robot's kinematic model requires the use of more complex mathematical techniques. In the next section, optimal control theory will be used on the robot's kinematic model to obtain a solution that allows the robot to traverse any specified trajectory.

## **4.2.2 Optimal Control**

### **4.2.2.1 Overview**

Classical control theory is useful when dealing with simple systems. Single Input Single Output (SISO) systems have been extensively studied. In these systems iterative techniques are used to meet desired system specifications. A SISO system's performance is evaluated based upon measures such as rise time, settling time, peak overshoot, gain and phase margin, and bandwidth. These system requirements are easily quantified and there are a number of design methodologies that can be used to design controllers to meet these requirements [17].

However, given a Multiple Input Multiple Output (MIMO) system, other methods must be used to design controllers that meet desired performance requirements. For example, classical control theory cannot be used to drive a MIMO system from one state to another in minimum time or change a MIMO system's state using the minimum amount of energy. For this type of problem, optimal control theory serves as an excellent mathematical framework.

Unfortunately, optimal control theory does have some drawbacks. Solving optimal control problems is usually very computationally complex and rarely can they be done analytically. Optimal control theory has only recently become a practical solution due to the advent of the digital computer.

The objective of optimal control theory is to determine the control signals that will cause a process to satisfy its physical constraints and at the same time minimize (or maximize)

some performance criterion. According to Kirk [17], the formulation of an optimal control problem requires:

1. A mathematical description (or model) of the process to be controlled.
2. A statement of the physical constraints.
3. Specification of a performance criterion.

Since the global position model of a two-wheeled differential drive robot was discussed in section 4.1.3, in the next section the physical constraints and performance criteria of the robot will be discussed. This will then fulfill the formulation requirements of an optimal control problem and allow optimal control theory to be applied to obtain an optimal controller for the two-wheeled differential drive robot.

#### **4.2.2.2 Physical Constraints and Performance Criteria**

As in any system, there are always physical constraints that limit a system's performance. When applying optimal control to a problem these physical constraints must be taken into account in order to obtain a realistic solution. Improperly accounting for a system's constraints can have devastating results.

Since the model being used is a kinematic model there are limitations on the robot's wheel velocity,  $\omega_{\max}$ . Considering the two-wheeled differential drive system the constraint becomes fairly apparent. Based on the robotic model derived in the previous section, this can be mathematically written

$$\text{Wheel Velocity: } |\omega| \leq \omega_{\max} \quad (4.9)$$

The maximum wheel velocity depends on the motors used to drive the robot. For generality, these quantities will remain as variables for problem formulation. Furthermore, it should be noted that since the inputs are bounded, Pontryagin's minimum principle must be checked in order to ensure an optimal control solution.

Generating useful performance criteria is usually the hardest part of formulating an optimal control problem and is very problem specific. Performance criteria are mathematically represented using a cost function,  $J$ . Optimal control theory attempts to minimize or maximize this function while taking into account the physical constraints placed on the system. The cost function greatly affects the outcome of the optimal solution.

To define a useful cost function it is helpful to articulate the performance goals of the problem. The goal of this problem is to have a robot follow a desired path,  $q_d$ , in a fixed amount of time,  $T$ .

Desired Path :  $q_d(t)$  where,

$$q_d(t_f) = q_f = \begin{bmatrix} x_f \\ y_f \\ \theta_f \end{bmatrix} \quad \text{and} \quad T = [t_0, t_f]$$

Therefore, it is important that the robot get to its final state, minimizes energy, and follow the desired path. To accomplish these goals the following cost function is devised

(4.10)

$$J(u(t)) = \frac{1}{2} [q(t_f) - q_f]^T H [q(t_f) - q_f] + \frac{1}{2} \int_{t_0}^{t_f} [q(t) - q_d(t)]^T Q [q(t) - q_d(t)] + u^T(t) R u(t) dt$$

where,

$$\begin{aligned} H &= \text{Final State Weighting Matrix} \\ Q &= \text{Desired State Weighting Matrix} \\ R &= \text{Energy Expenditure Weighting Matrix} \end{aligned}$$

The above cost function, equation 4.10, allows a great deal of flexibility and control over the behavior of the robot. Using this cost function, many properties of the robot can be controlled simultaneously. Using this cost function, a trade-off between how close the robot comes to its desired trajectory and how much energy the robot uses can be adjusted. Obviously, numerous simulations with different weighting matrices must be tried in order to obtain a desirable controller. Generating meaningful numbers for the weighting matrices is a trial and error process.

Now that the system is modeled, physical constraints are realized, and performance criteria are specified, optimal control theory can then be applied to obtain an optimal solution.

### 4.2.2.3 Obtaining a Solution – Numerical Techniques

In order to minimize this cost function given the robot's dynamic constraints the Hamiltonian is defined

$$H(q(t), u(t), p(t)) = \hat{g}(q(t), u(t), t) + p^T(t) [a(q(t), u(t), t)]$$

For this robotic system the Hamiltonian is

$$H = \frac{1}{2} [[q(t) - q_d(t)]^T Q [q(t) - q_d(t)] + u^T(t) R u(t)] + p^T(t) \begin{bmatrix} \frac{1}{2} r \cos(\theta) & \frac{1}{2} r \cos(\theta) \\ \frac{1}{2} r \sin(\theta) & \frac{1}{2} r \sin(\theta) \\ -\frac{r}{2b} & \frac{r}{2b} \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix}$$

From the Hamiltonian the costate dynamics and the input constraints that lead to the minimizing input,  $u^*(t)$  are obtained

$$\dot{p}(t) = -\frac{\partial H}{\partial q} = -\begin{bmatrix} \frac{\partial H}{\partial x} \\ \frac{\partial H}{\partial y} \\ \frac{\partial H}{\partial \theta} \end{bmatrix}$$

$$\dot{p}(t) = -\begin{bmatrix} [x(t) - x_d(t)]Q_{11} \\ [y(t) - y_d(t)]Q_{22} \\ [\theta(t) - \theta_d(t)]Q_{33} + p_1(t)\left[-\frac{1}{2}r\sin(\theta)\omega_L(t) - \frac{1}{2}r\sin(\theta)\omega_R(t)\right] + p_2(t)\left[\frac{1}{2}r\cos(\theta)\omega_L(t) + \frac{1}{2}r\cos(\theta)\omega_R(t)\right] \end{bmatrix}$$

where,

$$p(t_f) = \frac{\partial h}{\partial q}(x(t_f)) = H \begin{bmatrix} x(t_f) - x_f \\ y(t_f) - y_f \\ \theta(t_f) - \theta_f \end{bmatrix}$$

and

$$0 = \frac{\partial H}{\partial u} = \begin{bmatrix} \frac{\partial H}{\partial \omega_L} \\ \frac{\partial H}{\partial \omega_R} \end{bmatrix}$$

$$= \begin{bmatrix} R_{1,1}\omega_L(t) + p_1(t)\left[\frac{1}{2}r\cos(\theta)\right] + p_2(t)\left[\frac{1}{2}r\sin(\theta)\right] + p_3(t)\left[-\frac{r}{2b}\right] \\ R_{2,2}\omega_R(t) + p_1(t)\left[\frac{1}{2}r\cos(\theta)\right] + p_2(t)\left[\frac{1}{2}r\sin(\theta)\right] + p_3(t)\left[\frac{r}{2b}\right] \end{bmatrix}$$

In order to obtain a solution to the problem, two sets of differential equations whose conditions are given at different times have to be solved simultaneously. This is known as a two-point boundary value problem. In most cases numerical techniques must be employed to solve these types of problems. In order to solve this problem a numerical technique known as the method of steepest descent was chosen.

The general idea behind this method is to guess a  $u(t)$ , derive the solutions to the state and costate equations from this guess, and see if  $dH/du$  is approximately 0. If  $dH/du$  is close to 0 then the  $u(t)$  chosen is close to optimal. If  $dH/du$  is not close to 0, adjust the guess of  $u(t)$  using the gradient of  $dH/du$ . By adjusting the guess by the gradient the optimal solution is approached in the steepest direction, hence the name the method of steepest descent. An algorithm for this method is displayed in the flow chart below.

Select a discrete approximation to the nominal control history  $u^{(0)}(t)$ ,  $t \in [t_0, t_f]$ , and store this in the memory of the digital computer.

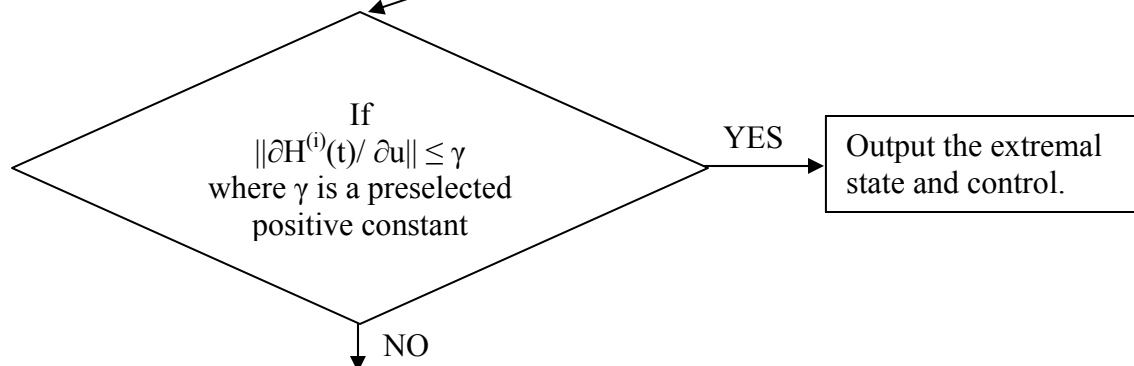
For example, subdivide the interval  $[t_0, t_f]$  into  $N$  subintervals (generally of equal duration) and consider the control  $u^{(0)}$  as being piecewise-constant during each of these subintervals; that is,

$$u^{(0)}(t) = u^{(0)}(t_k), \quad t \in [t_k, t_{k+1}], \quad k = 0, 1, \dots, N-1$$

Let the iteration index  $i$  be zero.

Using the control history  $u^{(i)}$ , integrate the state equations from  $t_0$  to  $t_f$  with initial conditions  $x(t_0) = x_0$  and store the resulting state trajectory  $x^{(i)}$  as a piecewise-constant vector function.

Calculate  $p^{(i)}(t_f)$  by substituting  $x^{(i)}(t_f)$  from step 2. Using this value of  $p^{(i)}(t_f)$  as the “initial condition” and the piecewise-constant values of  $x^{(i)}$  stored in step 2, integrate the costate equations from  $t_f$  to  $t_0$ , evaluate  $\partial H^{(i)}(t) / \partial u$ ,  $t \in [t_0, t_f]$ , and store this function in piecewise-constant fashion. The costate trajectory does not need to be stored.



Generate a new piecewise-constant control function given by,

$$u^{(i+1)}(t_k) = u^{(i)}(t_k) - \tau \partial H^{(i)}(t_k) / \partial u, \quad k = 0, \dots, N-1$$

where

$$u^{(i)}(t) = u^{(i)}(t_k) \quad \text{for } t \in [t_k, t_{k+1}), \quad k = 0, \dots, N-1.$$

Replace  $u^{(i)}(t_k)$  by  $u^{(i+1)}(t_k)$ ,  $k = 0, \dots, N-1$

**Figure 4.5:** Flow Chart of Steepest Descent Algorithm

### 4.2.2.4 Simulation Results

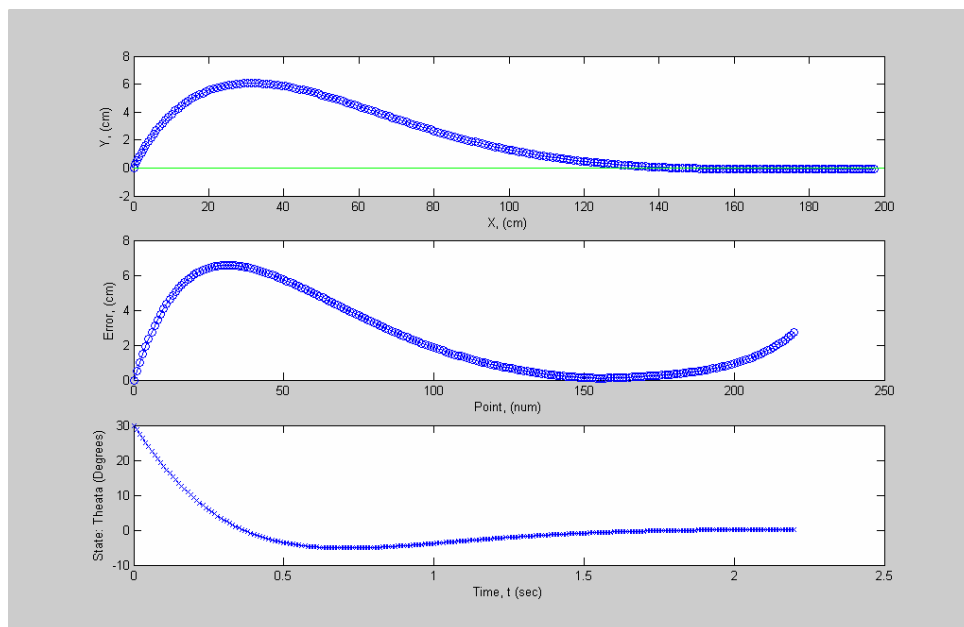
Using this method, the following results were obtained. This simulation was run for both a straight and curved trajectory. The robot had a diameter of 1 meter,  $b = 50$  cm, a wheel radius,  $r = 7.5$  cm, a maximum wheel speed of,  $\omega_{\max} = 26.6$  rad/s, and the following weighing matrices were used

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

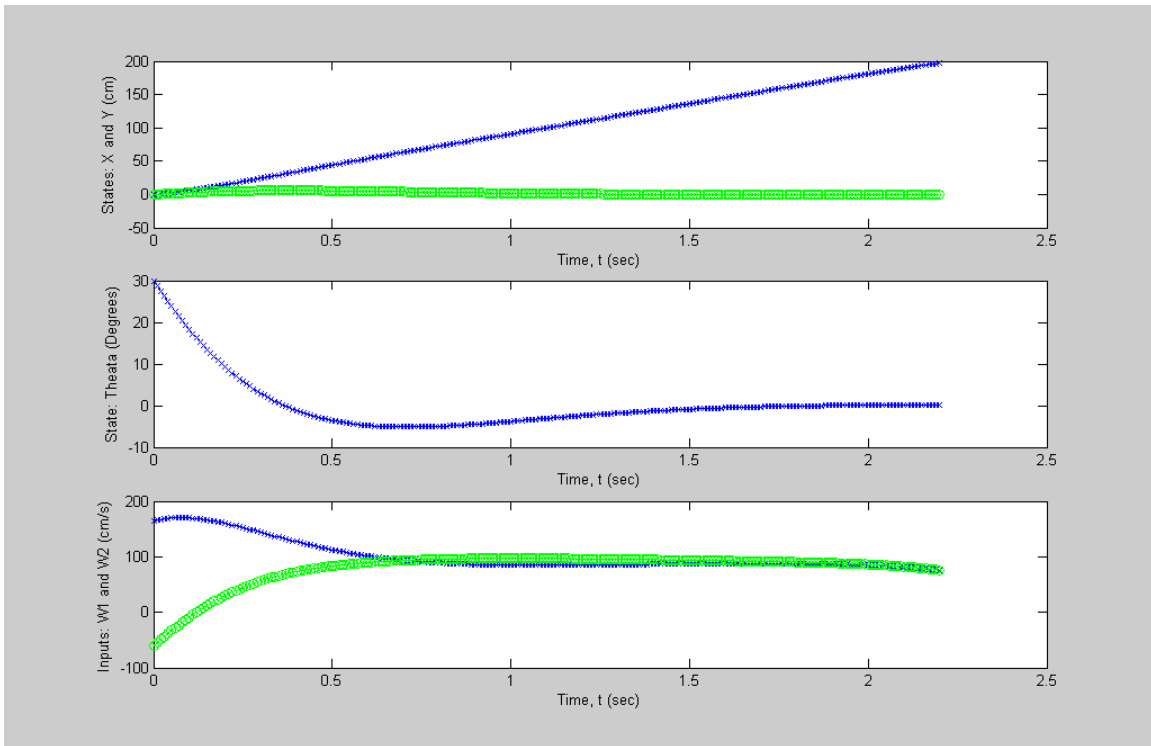
Also, it was required that the norm of  $dH/du$  be less than 0.01 before exiting. The results are displayed in Figures 4.6 – 4.11 below,

#### Straight Path:

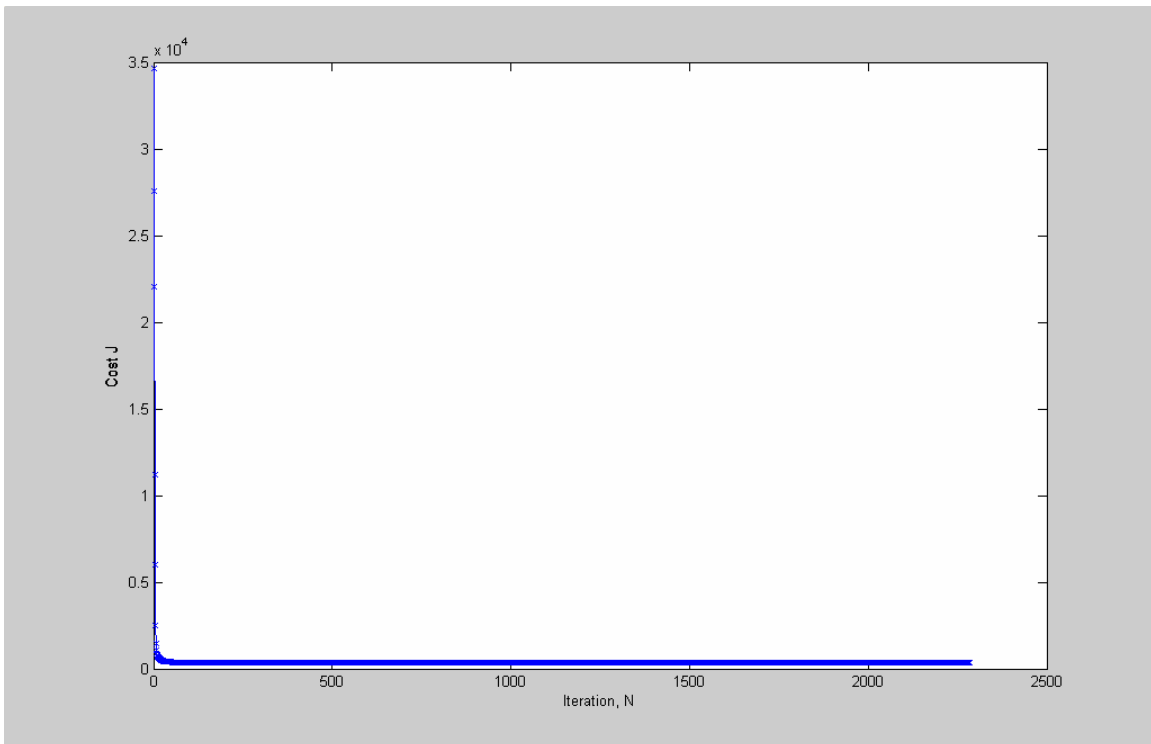
In the first simulation the robot's desired trajectory was a straight line from the state  $[x, y, \theta] = [0 \text{ cm}, 0 \text{ cm}, 0^\circ]$  to  $[x, y, \theta] = [200 \text{ cm}, 0 \text{ cm}, 0^\circ]$  in 2.2 seconds. Common sense would imply that to make the robot go straight, equal velocities should be applied to each wheel. As was expected, running the simulation for this case did in fact converge to this solution. However, in order to make the simulation more interesting the robot was initially given a heading error of  $30^\circ$ . This error can be seen in Figure 4.6. As was expected, the controller compensates for this heading error and brings the robot back to the desired trajectory in about 1.5 seconds.



**Figure 4.6:** Movement of Robot (blue) vs Desired Path (green)



**Figure 4.7:** Transition of States, X, Y, and Theta and Input Wheel Velocities

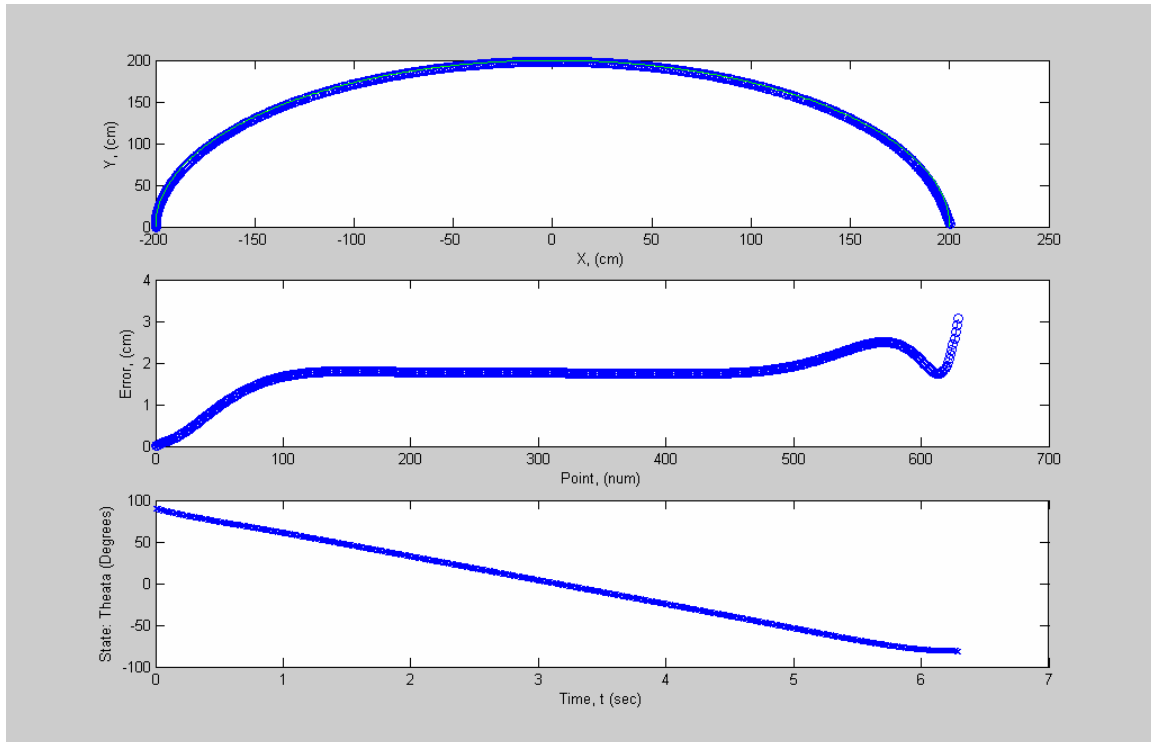


**Figure 4.8:** Transition of Cost Function

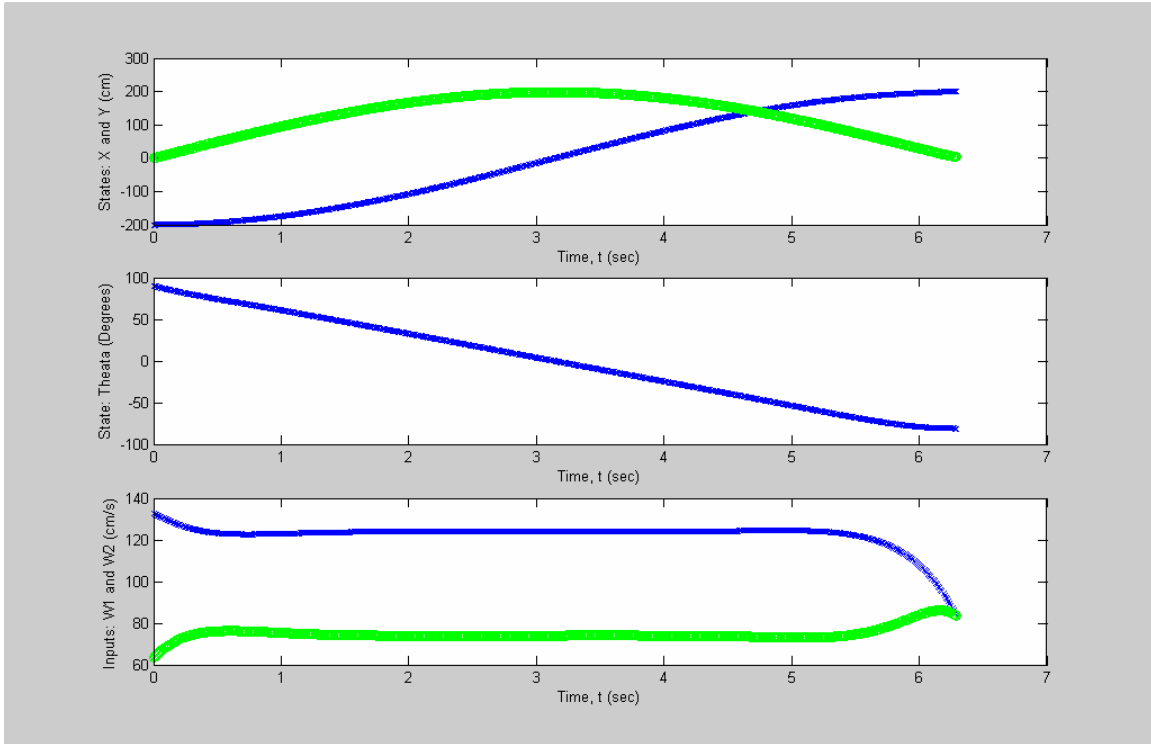


### Curved Path:

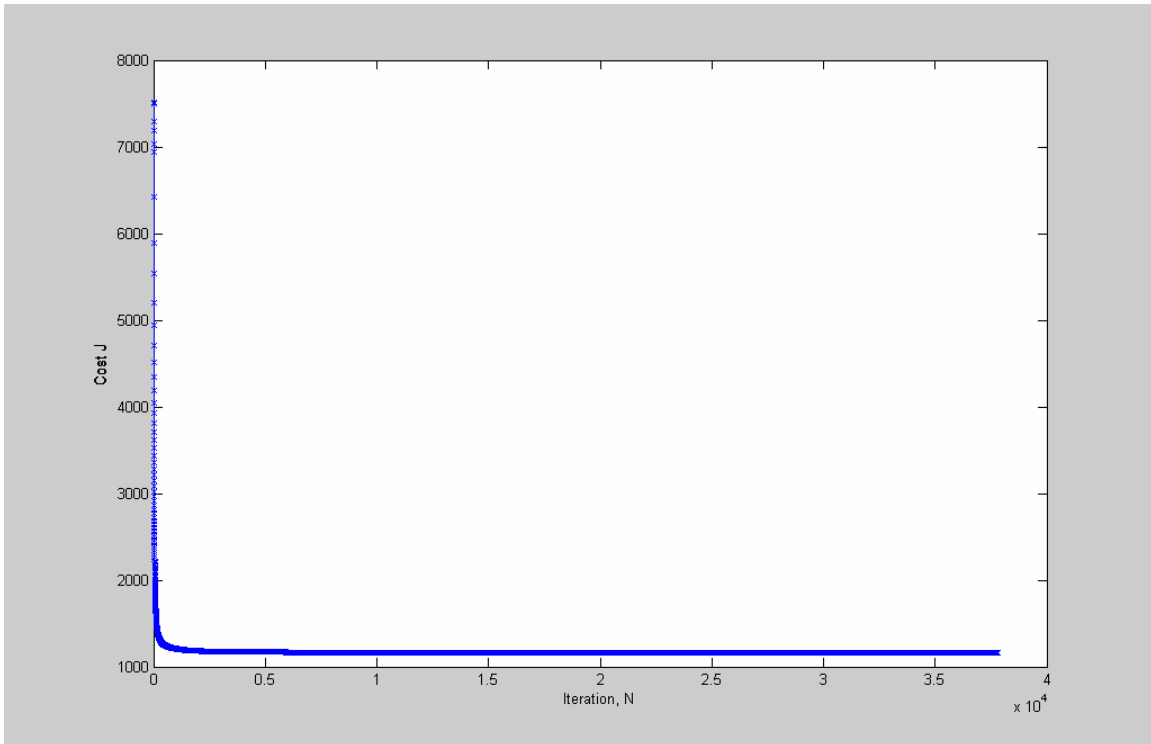
In this simulation the desired path of the robot was a half circle with a radius of 2 meters. The robot started in the state  $[x, y, \theta] = [-200 \text{ cm}, 0 \text{ cm}, 90^\circ]$  and its final desired state was  $[x, y, \theta] = [200 \text{ cm}, 0 \text{ cm}, -90^\circ]$ . It was desired that the robot complete this path in 6.28 seconds. As can be seen in Figure 4.9, the controller converged to a solution that kept the robot a distance no farther than 3 cm from the desired path. This can be considered an acceptable solution given the intended lawn mowing application.



**Figure 4.9:** Movement of Robot (blue) vs Desired Path (green)



**Figure 4.10:** Transition of States, X (blue), Y (green), and Theta and Input Wheel Velocities



**Figure 4.11:** Transition of Cost Function

Unfortunately, after running the simulation there were some obvious problems with optimal control. The first problem encountered was the computational complexity of these problems. Despite using a Pentium III 500 Mhz computer, deriving a solution to even the simplest case required several hours.

Due to this complexity, the problems had to be solved offline. This was a major limitation given the desired lawn mowing application. In order to use optimal control the ALM would have to have a predetermined route through the lawn and all the obstacles would have to be known in advance. As was mentioned in Chapter 3, it was desired that the ALM map and traverse the lawn as well as avoid obstacles in real time. Therefore, this served as a major practical limitation.

Furthermore, determining the appropriate weights for the weighting matrices was a trial and error process. Unfortunately, since the computational time required to solve these problems was so long, determining appropriate weights proved to be an excruciating and exhaustive process. Each time new weighting matrices were tried, the simulation would have to be run again, in turn requiring several hours for results.

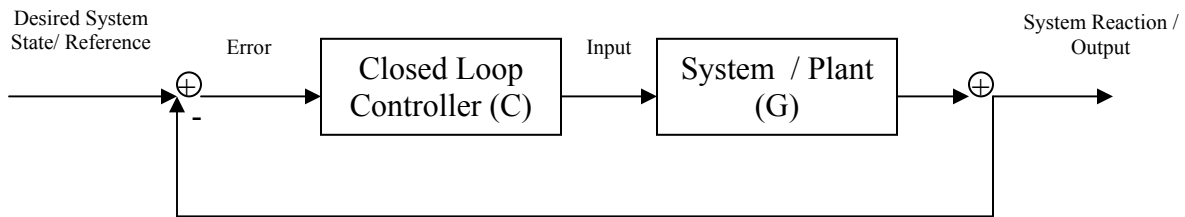
Finally, it should be noted that the solutions obtained by this simulation were open loop. Therefore, each solution would be for transitioning the robot from some initial state to some final state, over a specified trajectory in a certain amount of time. This made the robot very vulnerable to any disturbances it encountered. If at any point during the robot's traversal it got off its desired path, all subsequent movements would be invalid and possibly move the robot further off its desired path. Examining a lawn, there are a number of ways the ALM could get thrown off course, including rocks, ruts, roots, holes or elevation changes. This again was a severe practical limitation.

In order to overcome these critical problems, additional control methods were examined. In the next section, closed loop control methods are considered. These methods do not have the limitations discussed above and proved to be the best method of motion control for the ALM.

## **4.3 Closed Loop Control**

Closed loop control differs from open loop control in the fact that it requires feedback of the system state. In a closed loop control system the output of the system is constantly monitored and fed back to the controller. Therefore, the controller makes decisions not only on the reference input but also on the current state of the system. Therefore, the controller actually utilizes the error between the reference input and the actual state of the system to make its control decisions. A closed loop controller is constantly trying to bring the error between the reference input and the system state to zero. Due to this

methodology, closed loop control is more robust to disturbances than open loop control. A block diagram of the closed loop control paradigm can be seen in Figure 4.12 below.

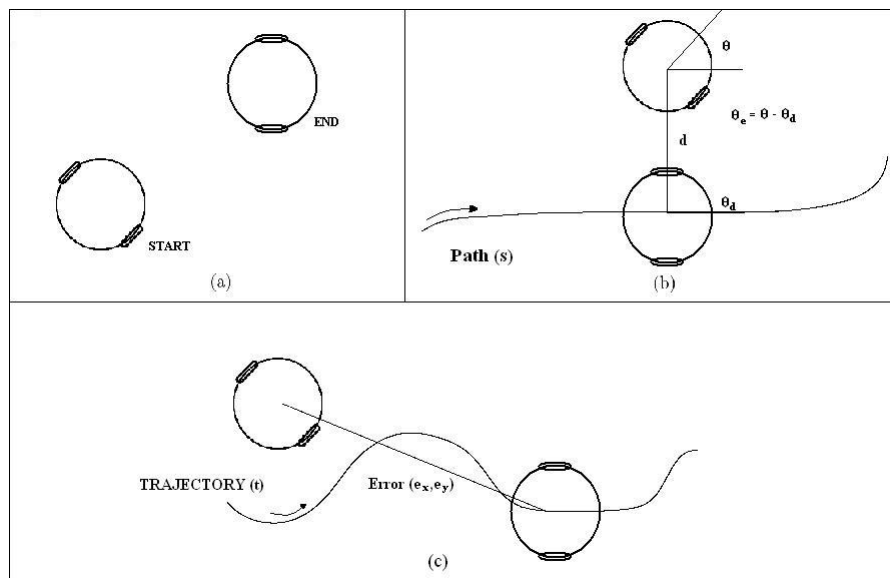


**Figure 4.12:** Closed Loop Control Paradigm

For nonholonomically constrained robots three different motion tasks have been previously studied. They are point-to-point stabilization, path following, and trajectory tracking. Based on previous research [19], closed loop controllers have been designed for each type of problem. Each task can be classified as follows:

1. Point-to-point stabilization: The robot must reach a desired goal configuration starting from a given initial configuration.
2. Path following: The robot must reach and follow a geometric path in Cartesian space starting from a given initial configuration (on or off the path).
3. Trajectory tracking: The robot must reach and follow a trajectory in Cartesian space (i.e., a geometric path with an associated timing law) starting from a given initial configuration (on or off the trajectory).

These motion tasks are further exemplified in Figure 4.13 below.



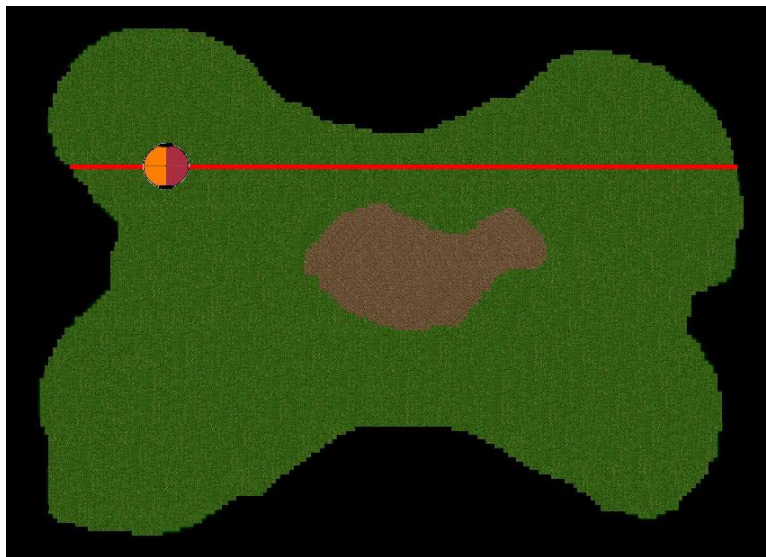
**Figure 4.13:** Robotic Motion Tasks

(a) Point-to-point stabilization, (b) Path following, (c) Trajectory tracking

When examining these different motion tasks and comparing them to the desired motion tasks of the ALM, it is apparent that the ALM would require both path following and trajectory tracking controllers. The ALM would use path following when it is tracing the outside border of the yard or tracing the perimeter of a known obstacle within the yard. Additionally, the ALM would use a trajectory tracking scheme when it is moving in straight lines across the yard. By equipping the ALM with controllers for these two types of movement schemes, the ALM would then have all the functionality needed to implement the navigational algorithm discussed in Chapter 3.



**Figure 4.14:** The ALM utilizing a path following controller

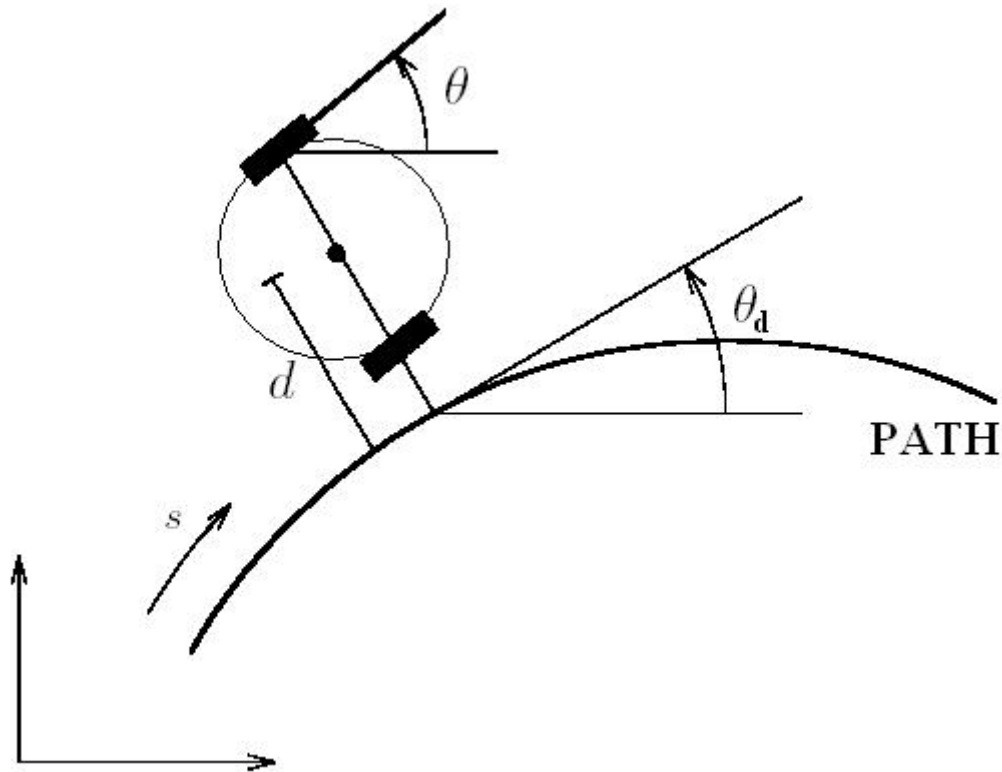


**Figure 4.15:** The ALM utilizing a trajectory tracking controller

## 4.3.1 Path Following

### 4.3.1.1 Path Coordinate Model

In order to design a closed loop controller for path following, a different mathematic model must be formulated. This model must include path parameters and error parameters that relate the desired path coordinates and path orientation with the robot's current location and orientation. Figure 4.16 displays the relevant parameters in order to derive the path coordinate model [32].



**Figure 4.16:** Path Coordinate Model

Assume that  $\rho_d(x_d(t), y_d(t))$  is the desired path the robot wishes to traverse, and it is parameterized with  $s(t)$ , the distance along the path. Therefore,

$$s(t) = \int_0^t V_d(\tau) d\tau + s(0)$$

where  $s(0)$  is the initial distance and

$$V_d(t) = \sqrt{\dot{x}_d^2(t) + \dot{y}_d^2(t)}$$

is the linear velocity of the projection P of (x,y) onto  $\rho_d$ . Let d denote the lateral error where a positive magnitude represents a situation where the path is to the right and negative magnitude represents a situation where the path is to the left. Also, let

$$\theta_e = \theta - \theta_d$$

denote the orientation error.  $\theta_d$  is the desired orientation of the robot and is tangent to the desired path,  $\rho_d$ . It should be noted that the desired curvature at any given point on the path is

$$c(s) = \frac{d\theta_d}{ds} = \frac{\dot{\theta}_d(t)}{\dot{s}(t)}$$

and the linear and angular velocity of the robot are denoted  $V(t)$  and  $\omega(t)$  respectively. Therefore,

$$\dot{d}(t) = V(t) \sin \theta_e(t)$$

$$\begin{aligned} \dot{s}(t) &= V(t) \cos \theta_e(t) + d(t) \cdot \dot{\theta}_d(t) \\ &= V(t) \cos \theta_e(t) + d(t) \cdot c(s) \cdot \dot{s}(t) \\ &= \frac{V(t) \cos \theta_e(t)}{1 - d(t) \cdot c(s)}, \text{ where } d(t) \cdot c(s) \neq 1 \end{aligned}$$

$$\begin{aligned} \dot{\theta}_e(t) &= \dot{\theta}(t) - \dot{\theta}_d(t) \\ &= \omega(t) - c(s) \cdot \dot{s}(t) \\ &= \omega(t) - c(s) \cdot \frac{\cos \theta_e(t)}{1 - d(t) \cdot c(s)} V(t) \end{aligned}$$

Writing these equations in matrix form, the path coordinate model is then

$$\begin{bmatrix} \dot{s}(t) \\ \dot{d}(t) \\ \dot{\theta}_e(t) \end{bmatrix} = \begin{bmatrix} \frac{\cos \theta_e(t)}{1 - d(t) \cdot c(s)} \\ \sin \theta_e(t) \\ -\frac{c(s) \cos \theta_e(t)}{1 - d(t) \cdot c(s)} \end{bmatrix} V(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega(t) \quad (4.11)$$

This model derives a relationship between the robot's position and orientation in relation to its desired path. In the following section, this model will be used to design a controller

that will drive its states,  $d$  and  $\theta_e$  to zero, thus, forcing the robot to follow the desired path.

### 4.3.1.2 Chain Form

The development of a controller is much easier if the system is put into chain form. Chain form is useful because it gives the system a very controllable structure. The  $(2,n)$  single-chain form has the structure

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= x_2 u_1 \\ &\vdots \\ \dot{x}_n &= x_{n-1} u_1 \end{aligned}$$

The chained form system, although nonlinear, has a strong underlying linear structure. This becomes apparent when  $u_1$  is assigned as a function of time, and is no longer regarded as a control variable. In this case, it becomes a single-input, time-varying linear system. Based on the path coordinate model, the robot studied in this section has three states. Therefore, it has the  $(2,3)$  chain form

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= x_2 u_1 \end{aligned}$$

In order to put the model in chain form the following variable transformation must be performed

$$\begin{aligned} x_1 &= s \\ x_2 &= [1 - dc(s)] \tan \theta_e \\ x_3 &= d \end{aligned}$$

where  $s$  is the path length,  $d$  is the lateral error from the path,  $\theta_e$  is the orientation error of the robot from the path and  $c(s)$ , is the curvature of the path. The inputs then become

$$\begin{aligned} V &= \frac{1 - dc(s)}{\cos \theta_e} u_1 \\ \omega &= \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 \end{aligned}$$



where,  $V$  is the linear velocity of the robot and  $\omega$  is the angular velocity of the robot, and

$$\begin{aligned}\alpha_1 &= [\sin^2(\theta_e) + 1]c(s) \\ \alpha_2 &= \frac{\cos^2(\theta_e)}{1 - dc(s)} \\ \alpha_3 &= \frac{\cos\theta_e \sin\theta_e \cdot d}{1 - dc(s)} c'(s)\end{aligned}$$

### 4.3.1.3 Input Scaling Controller

The design of an input-scaling controller for a “car-like” robot was presented in [19]. The following work is a modification of this work using the path coordinate model for a two-wheel differential drive robot. Proof that the input scaling controller drives a chain form system’s error to zero can be seen below [25].

Given any system in chain form, its variables can be rearranged as

$$\chi = (x_1, x_2, \dots, x_{n-1}, x_n) = (x_1, x_n, \dots, x_3, x_2)$$

Therefore, the chained form system becomes

$$\begin{aligned}\dot{x}_1 &= u_1 \\ \dot{x}_2 &= x_3 u_1 \\ \dot{x}_3 &= x_4 u_1 \\ &\vdots \\ \dot{x}_{n-1} &= x_n u_1 \\ \dot{x}_n &= u_2\end{aligned}$$

or

$$\dot{\chi} = h_1(\chi)u_1 + h_2u_2, \quad h_1(\chi) = \begin{bmatrix} 1 \\ \chi_3 \\ \chi_4 \\ \vdots \\ \chi_n \\ 0 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

Let  $\chi = h_1(x_1, \chi_2)$ , with  $\chi_2 = (x_2, x_3, \dots, x_n)$ . Writing the problem in this notion, the objective of the controller is to stabilize  $\chi_2$  to zero. Examining the states of this transformed system for a two wheel differential drive robot,  $x_2$ , is the lateral error of the robot from the desired path, and  $x_3$  relates to the orientation error of the robot from the desired path. Therefore, the controller will attempt to drive these two variables to zero, therefore forcing the robot to follow the desired path.

To begin, the new chain form system can be written as

$$\begin{aligned} \dot{\tilde{x}}_1 &= 0 \\ \dot{\chi}_2 &= \begin{bmatrix} 0 & u_1(t) & 0 & \dots & \dots & 0 \\ 0 & 0 & u_1(t) & 0 & \dots & 0 \\ \vdots & & & & & \vdots \\ 0 & \dots & \dots & 0 & u_1(t) & 0 \\ 0 & \dots & \dots & \dots & 0 & u_1(t) \\ 0 & 0 & \dots & \dots & \dots & 0 \end{bmatrix} \chi_2 + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2 \end{aligned}$$

with,

$$\tilde{x}_1 = x_1 - \int_0^t u_1(\tau) d\tau$$

When  $u_1$  is constant and nonzero, the system becomes time-invariant and the second part is clearly controllable. Furthermore, it can be proved that controllability will hold whenever  $u_1(t)$  is piecewise-continuous, bounded, and a strictly positive (or negative) function. Under these assumptions,  $x_1$  varies monotonically with time, and differentiation with respect to time can be replaced by differentiation with respect to  $\chi_1$ , being

$$\frac{d}{dt} = \frac{d}{dx_1} \dot{x}_1 = \frac{d}{dx_1} u_1$$

Therefore,

$$\text{sign}(u_1) \frac{d}{dx_1} = \text{sign}(u_1) \frac{1}{u_1} \frac{d}{dt} = \frac{1}{|u_1|} \cdot \frac{d}{dt}$$

This change of variables is equivalent to an input scaling procedure, hence the name of the controller that is derived. The second part of the system may be rewritten

$$\begin{aligned}
x_2^{[1]} &= \text{sign}(u_1)x_3 \\
x_3^{[1]} &= \text{sign}(u_1)x_4 \\
&\vdots \\
x_{n-1}^{[1]} &= \text{sign}(u_1)x_n \\
x_n^{[1]} &= \text{sign}(u_1)u_2'
\end{aligned}$$

where,

$$x_i^j = \text{sign}(u_1) \frac{d^j x_i}{dx_1^j} \text{ and } u_2' = \frac{u_2}{u_1}$$

Having transformed the system, it is now a linear time-invariant system and the equivalent input-output representation is

$$x_2^{[n-1]} = \text{sign}(u_1)^{n-1} u_2'$$

It can be proved that this system is controllable and admits an exponentially stable linear feedback in the form

$$u_2'(\chi_2) = -\text{sign}(u_1)^{n-1} \sum_{i=1}^{n-1} k_i x_2^{[i-1]}$$

where the gains  $k_i > 0$  are chosen so as to satisfy the Hurwitz stability criterion. Therefore, the time-varying control is

$$u_2(\chi_2, t) = u_1(t)u_2'(\chi_2)$$

Deriving the input scaling control law for the two-wheel differential drive robot is then

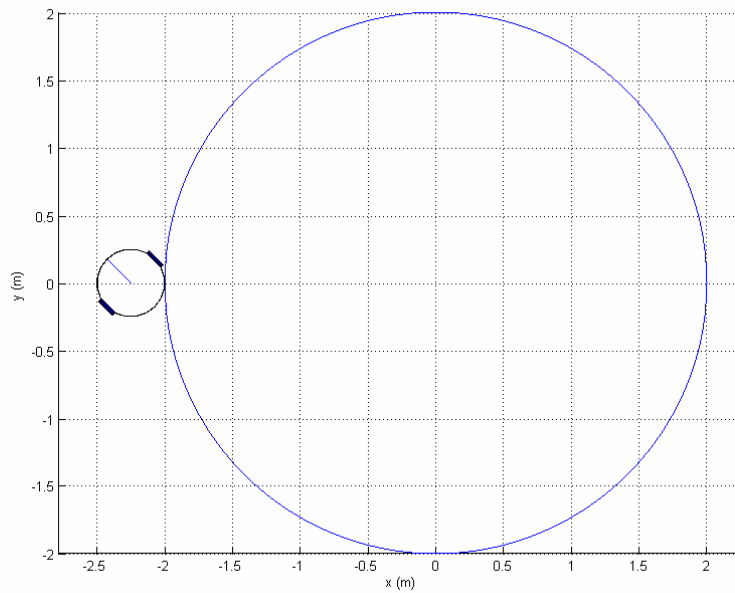
$$\begin{aligned}
u_2'(x_2, x_3) &= -1 \cdot [k_1 x_2^0 + k_2 x_2^1] \\
&= -k_1 x_2 - k_2 \text{sign}(u_1) x_3
\end{aligned}$$

$$\begin{aligned}
u_2(x_2, x_3, t) &= u_1(t)u_2'(x_2, x_3) \\
&= -k_1 u_1(t) x_2 - k_2 |u_1(t)| x_3
\end{aligned}$$

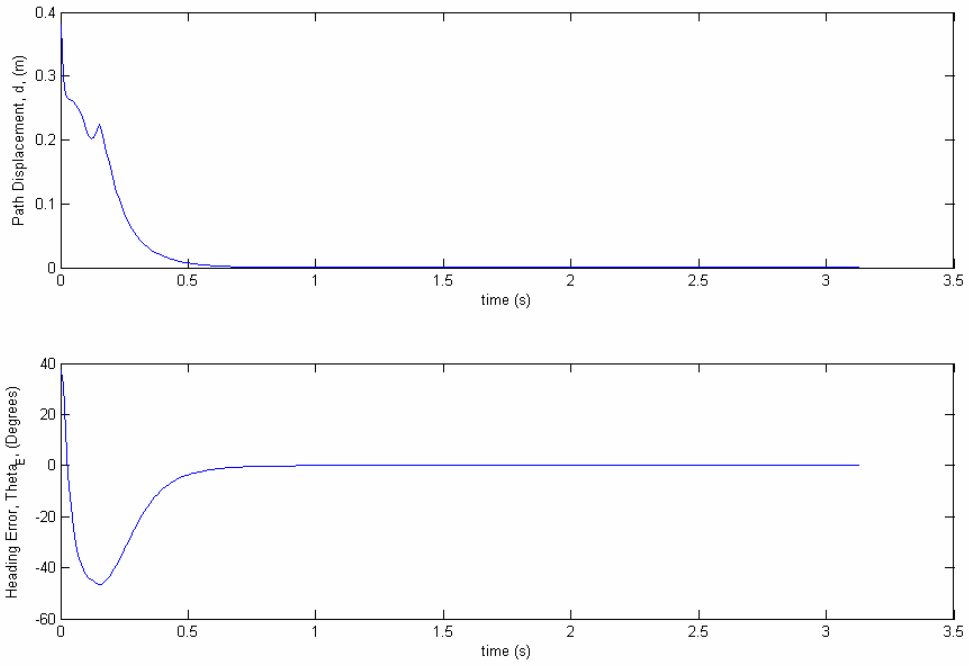
Therefore, it has been proven that the input scaling controller will drive the system exponentially to zero, in turn, forcing the robot to follow the desired path. This solves the robotic path following problem. This is demonstrated in the simulation in the following section.

### 4.3.1.4 Simulation

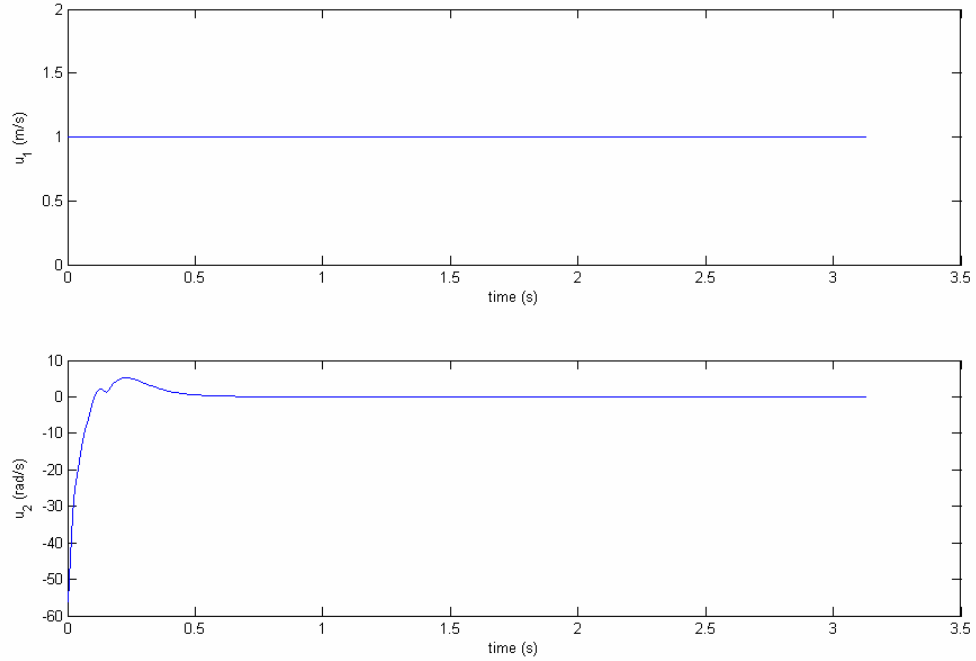
Here are the simulation results of a two-wheel differential drive robot using an input scaling controller for path following. The robot has a wheel radius,  $r$  set to 7.5 cm and a half wheelbase,  $b$  equal to 25 cm. Gain constants of the input scaling controller were  $\alpha = 10$  and  $k_1 = 2\alpha$  and  $k_2 = \alpha^2$ . Furthermore, the desired path was a circle of radius  $R$ , set to 2 meters. Note that initially, the state of the robot is set to  $[x, y, \theta] = [-2.25 \text{ m}, 0 \text{ m}, 135^\circ]$ . Therefore, the robot is not directly on top of the circle nor is it tangent to the desired path. This gives it an initial path displacement,  $d$  of 0.25 m and an initial heading error,  $\theta_e$  of  $45^\circ$ .



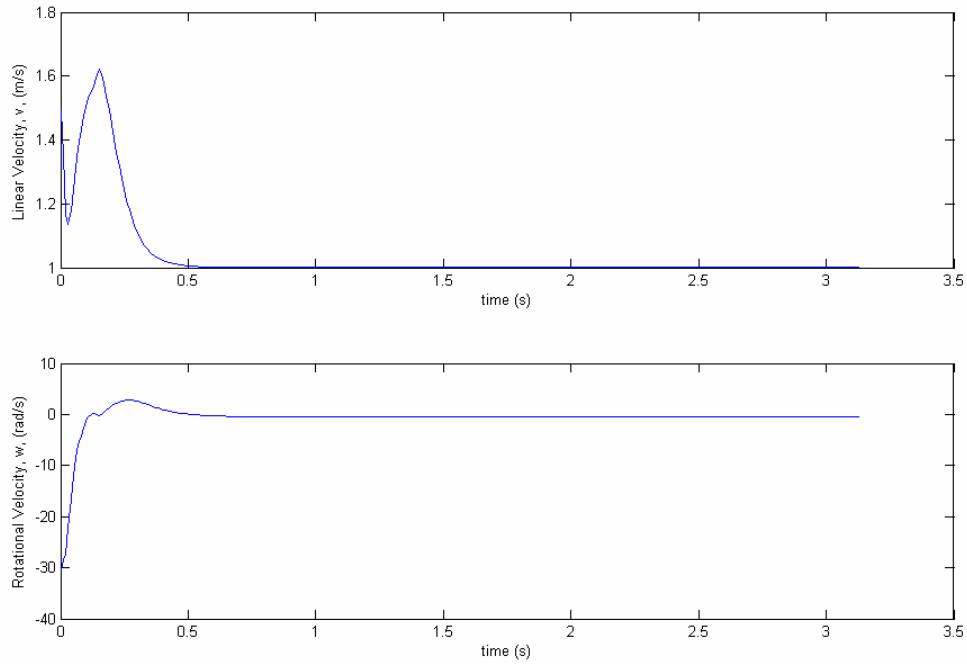
**Figure 4.17:** Initial Configuration of Path Following Simulation



**Figure 4.18:** Path Displacement,  $d$  (m), and Heading Error,  $\theta_e$  (Degrees)



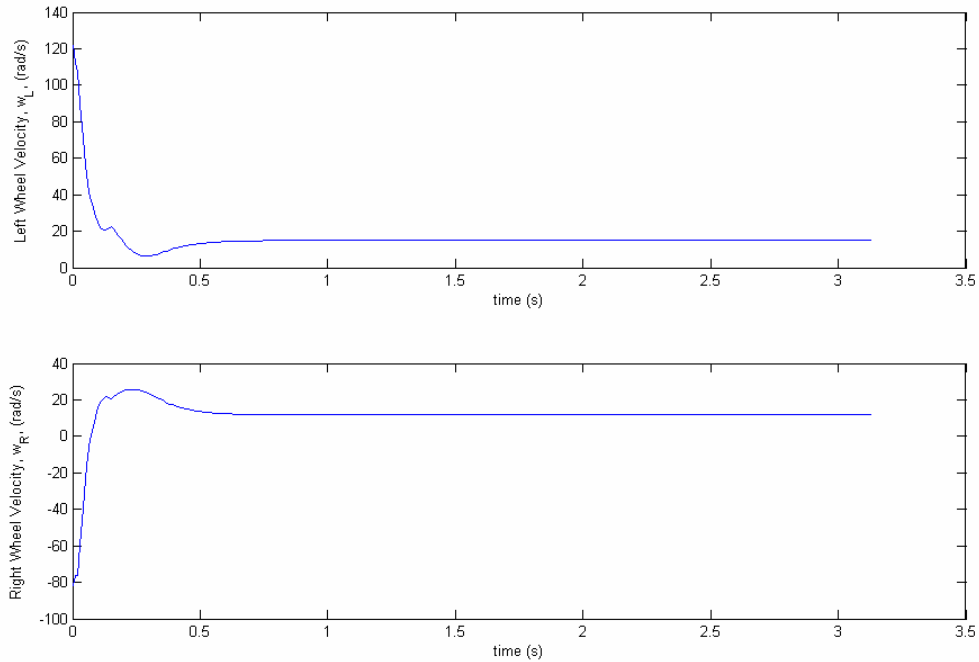
**Figure 4.19:** Control Inputs,  $u_1$  (m/s) and  $u_2$  (rad/s)



**Figure 4.20:** Linear velocity,  $v$  (m/s), and Rotational velocity,  $\omega$  (rad/s)

It should be noted that from the velocity control inputs,  $v$  and  $\omega$ , and from the robotic parameters,  $r$  and  $b$ , the required wheel velocities,  $\omega_L$  and  $\omega_R$  can be calculated

$$\begin{aligned}\omega_L &= \frac{v - b\omega}{r} \\ \omega_R &= \frac{v + b\omega}{r}\end{aligned}\tag{4.12}$$



**Figure 4.21:** Wheel Velocities,  $\omega_L$  (rad/s) and  $\omega_R$  (rad/s)

The simulation results shown above demonstrate the input scaling controller's ability to regulate a two-wheel differential drive robot's global position and force it to follow a desired path.

The simulation consisted of a circular path of radius  $R$ , set to 2 meters. This simplified the simulation because the curvature of the path is known a priori. The magnitude of the curvature of a circle is

$$c(s) = \frac{1}{R}$$

Furthermore, the sign of the curvature is positive if the path is turning left and negative if the path is turning right. Knowing where the robot started and the direction it should follow the path allowed the sign of the curvature to be calculated.

Choosing a circle as the desired path also made the curvature constant. Therefore, the derivative of the curvature was assumed to equal 0.

$$c'(s) = 0$$

By knowing the curvature a priori and having it be constant may seem like an unrealistic simulation. However, the purpose of this simulation was to demonstrate how closed loop control could be utilized to force a two-wheel differential drive robot to follow a desired path. Estimating the curvature of a path in real time is definitely a problem that must be considered if this controller were to have practical implications on the ALM.

Finally, it should be noted that for the simulation the linear velocity along the path,  $u_1$ , was considered a constant and set at 1 m/s. This is a realistic assumption because in practice it would be preferred that the ALM mow the lawn at a constant speed. Based on the derivation of the input scaling controller in the previous section,  $u_1$  must be a piecewise continuous, bounded, and strictly positive or negative function. Keeping  $u_1$  at a constant 1 m/s satisfied this constraint.

## 4.3.2 Trajectory Tracking

### 4.3.2.1 Chain Form

As was the case in the path following problem, derivation of a closed loop controller for trajectory tracking requires that the system be put into chain form. For trajectory tracking the global coordinate model derived in section 4.1.3 is used.

To put the system in (2,3) chain form

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= x_2 u_1 \end{aligned} \tag{4.13}$$

the following variable transformation is used

$$\begin{aligned} x_1 &= x \\ x_2 &= \tan \theta \\ x_3 &= y \end{aligned} \tag{4.14}$$

The inputs then become

$$\begin{aligned} V &= \frac{u_1}{\cos \theta} \\ \omega &= \frac{u_2}{\sec^2 \theta} \end{aligned} \tag{4.15}$$

where  $V$  is the linear velocity of the robot and  $\omega$  is the angular velocity of the robot.



### 4.3.2.2 Reference trajectory generation

For the trajectory tracking problem it is required that the robot follow a desired path  $(x,y)$  that is parameterized by time  $(t)$ . Therefore, it is assumed that a feasible, smooth output trajectory can be represented as the Cartesian position of the midpoint of the axial of a two-wheel differential drive robot. This output trajectory can be expressed

$$x_d = x_d(t) \quad y_d = y_d(t), \quad t \geq t_0 \quad (4.16)$$

From equation 4.16, the time evolution of the remaining states (state trajectory), as well the time evolution of the associated input commands (input trajectory) can be derived [19, 30].

Given the two-wheel differential drive system in chained form, equation (4.13), and the change of coordinates, equations (4.14)

$$\begin{aligned} \dot{x}_{d1} &= u_{d1} & x_{d1} &= x_d \\ \dot{x}_{d2} &= u_{d2} & x_{d2} &= \tan \theta_d \\ \dot{x}_{d3} &= x_{d2} u_{d1} & x_{d3} &= y_d \end{aligned}$$

the state trajectory and input trajectory in terms of the desired output trajectory can be obtained as

$$\begin{aligned} x_{d1} &= x_d \\ x_{d2} &= \frac{\dot{x}_{d3}}{u_{d1}} = \frac{\dot{y}_d}{\dot{x}_d} \\ x_{d3} &= y_d \end{aligned} \quad (4.17)$$

$$\begin{aligned} u_{d1} &= \dot{x}_d \\ u_{d2} = \dot{x}_{d2} &= \frac{d}{dt} \begin{bmatrix} \dot{x}_{d3} \\ \dot{x}_{d2} \end{bmatrix} = \frac{d}{dt} \begin{bmatrix} \dot{y}_d \\ \dot{x}_d \end{bmatrix} = \frac{\ddot{y}_d \dot{x}_d - \dot{y}_d \ddot{x}_d}{\dot{x}_d^2} \end{aligned} \quad (4.18)$$

It should be noted that there is a singularity in both the state and input trajectory, equations 4.17 and 4.18, when  $\dot{x}_d = 0$ . This would need to be taken into account in the actual implementation of the ALM. However, since the ALM's navigation system only uses trajectory tracking to travel in straight lines, it would need to be ensured that the desired trajectory of the ALM was never purely in the y direction. As was seen in the

ALM's navigational algorithm in Chapter 3, the ALM only travels in straight lines in the x direction; therefore, this would never be an issue.

### 4.3.2.3 Control via Approximate Linearization

Having the system in chain form allows a number of closed loop control techniques to be applied. One such technique, approximate linearization, uses standard linear control theory. This technique uses the approximate linearization of the system equations about the desired trajectory, which in turn leads to a time-varying system [19, 30].

For the chained form system, with the desired state and input trajectory, equations (4.17) and (4.18), the state and input errors are defined as

$$x_e = x_{di} - x_i, i = 1, \dots, 3 \quad u_e = u_{dj} - u_j, j = 1, 2$$

The error equations are

$$\begin{aligned} \dot{x}_{e1} &= u_{e1} \\ \dot{x}_{e2} &= u_{e2} \\ \dot{x}_{e3} &= x_{d2}u_{d1} - x_2u_1 \end{aligned}$$

Linearizing about the desired trajectory yields the following linear time-varying system

$$\dot{x}_e = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & u_{d1}(t) & 0 \end{bmatrix} x_e + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ x_{d2}(t) & 0 \end{bmatrix} u_e = A(t)x_e + B(t)u_e$$

Define the feedback term  $u_e$  as the following linear time-varying law

$$\begin{aligned} u_{e1} &= -k_1 x_{e1} \\ u_{e2} &= -k_2 x_{e2} - \frac{k_3}{u_{d1}} x_{e3} \end{aligned}$$

with  $k_1$  positive, and  $k_2$ , and  $k_3$  such that

$$\lambda^2 + k_2 \lambda + k_3$$

is a Hurwitz polynomial. With this choice, the closed loop system matrix

$$A_{cl}(t) = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & k_3 / u_{d1} \end{bmatrix}$$

has constant eigenvalues with negative real parts.

The ability to place the closed loop eigenvalues in the open left half-plane allows for control over the speed at which the tracking error converges to zero, as well as the amount of control effort that is used. This is an inverse relationship, therefore trade offs between convergence and control effort must be determined.

It should be noted that the second control input requires  $u_{d1} \neq 0$ . If the eigenvalues are assigned at constant locations, larger gains are required as the variable  $x_{d1}$  comes to a stop. One way to overcome this limitation is to assign the eigenvalues as functions of the input  $u_{d1}$ . For example imposing (beside the eigenvalue at  $-\lambda_1$ ) two coincident real eigenvalues at  $-\alpha|u_{d1}|$ , with  $\alpha > 0$ ,

$$u_{e2} = -2\alpha|u_{d1}|x_{e2} - \alpha^2|u_{d1}|x_{e3}$$

With this input scaling procedure, the second control input simply goes to zero when the desired trajectory  $x_{d1}$  comes to a stop.

Furthermore it should be noted, that the overall control input to the chained form representation is

$$u = u_d - u_e$$

The transformations, equation 4.15, should be used to determine the actual linear and rotational velocity inputs to the system, or in the case of the ALM, equation 4.12, for the right and left wheel velocities.

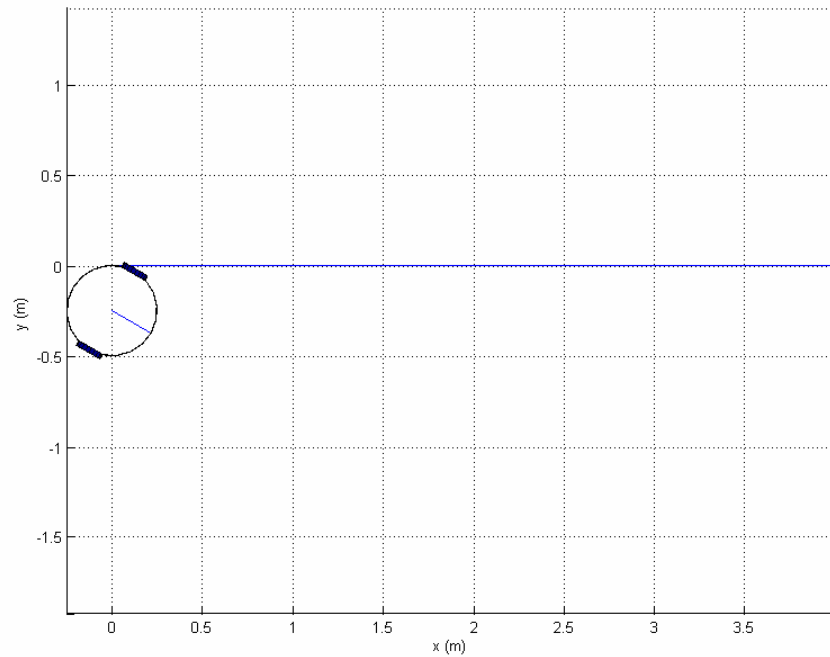
#### 4.3.2.4 Simulation

This simulation demonstrates the ability to design a closed loop controller for trajectory tracking. In this simulation, the two-wheel differential drive robot has a wheel radius,  $r$  set to 7.5 cm and a half wheel base,  $b$  equal to 25 cm. Gain constants of  $k_1 = \alpha = 5$  were used.

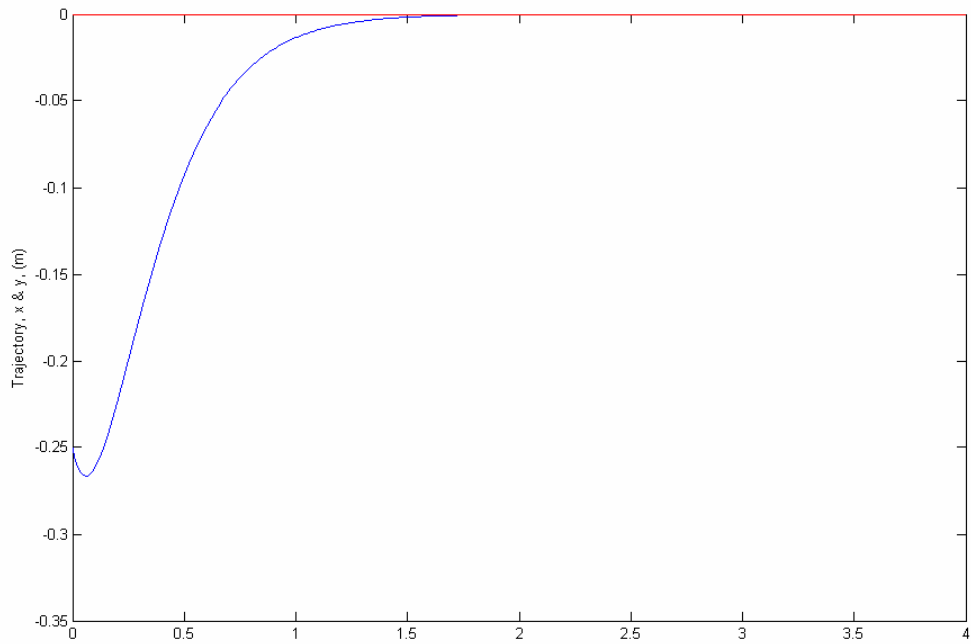
In the simulation a linear trajectory was chosen and the robot was initially set 0.25 meters off the desired trajectory with an initial heading error of 30 degrees. As can be seen from the figures below, the robot converges to the desired trajectory in approximately 1.5 seconds. By adjusting the gain constants  $k_1$  and  $\alpha$ , the time to convergence of the trajectory can be adjusted. However, a speedier convergence requires greater control effort. This increase in control effort would have to be practical, given the constraints of the actual hardware used on the ALM. Also, increasing the gains leads to increased

overshoot of the trajectory and eventually instability. Given actual hardware, various gain constants would have to be tried to meet practical performance criteria.

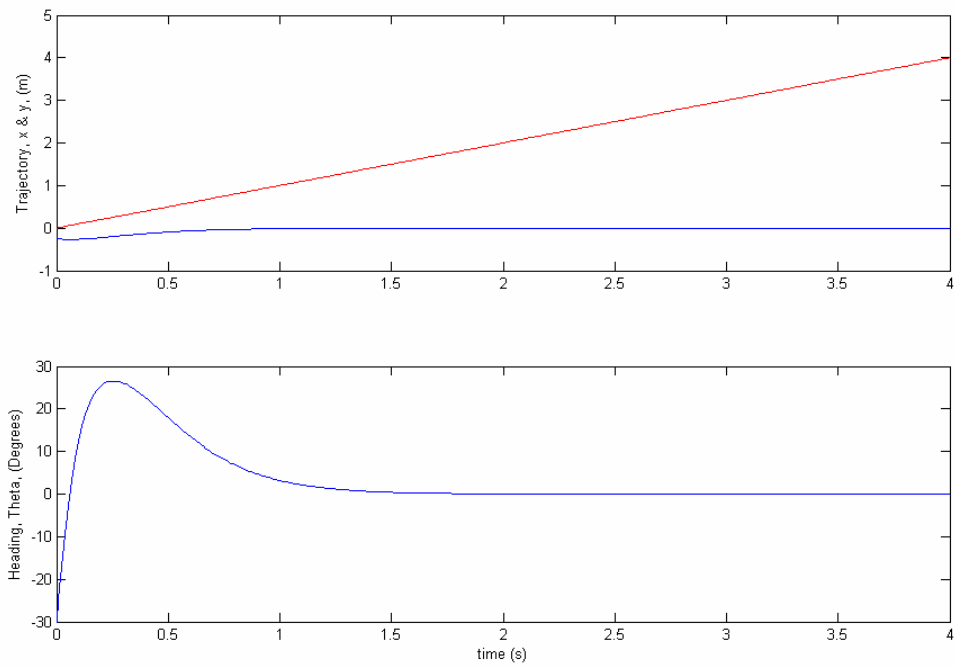
It should be noted that the ALM's navigational algorithm discussed in Chapter 3 would use trajectory tracking only when it needed to traverse straight sections of the lawn. Therefore, a linear trajectory is the only trajectory considered since it is the most applicable trajectory to the ALM.



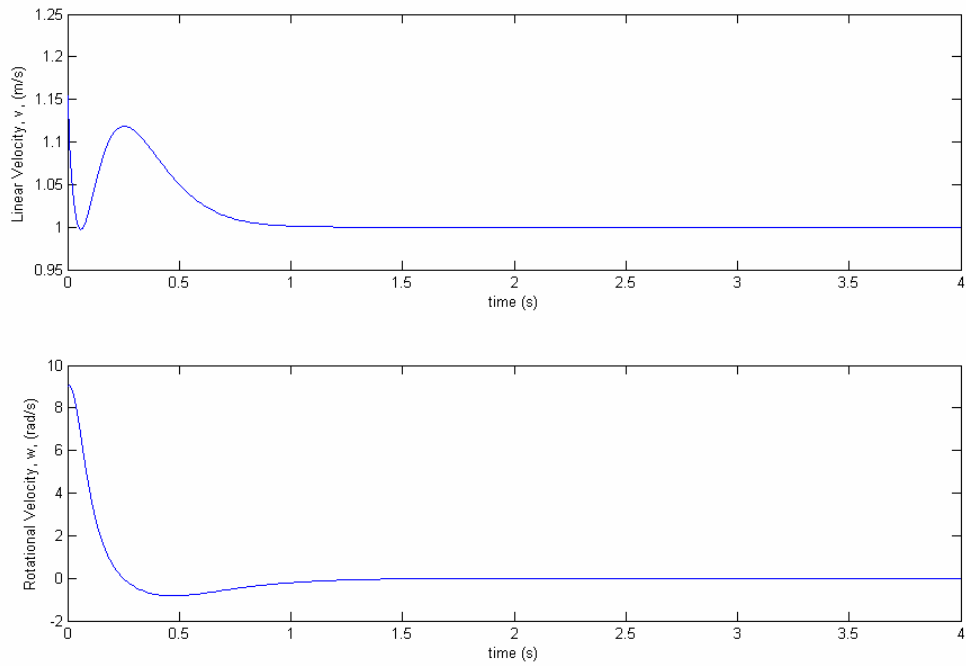
**Figure 4.22:** Initial Configuration of Trajectory Tracking Simulation



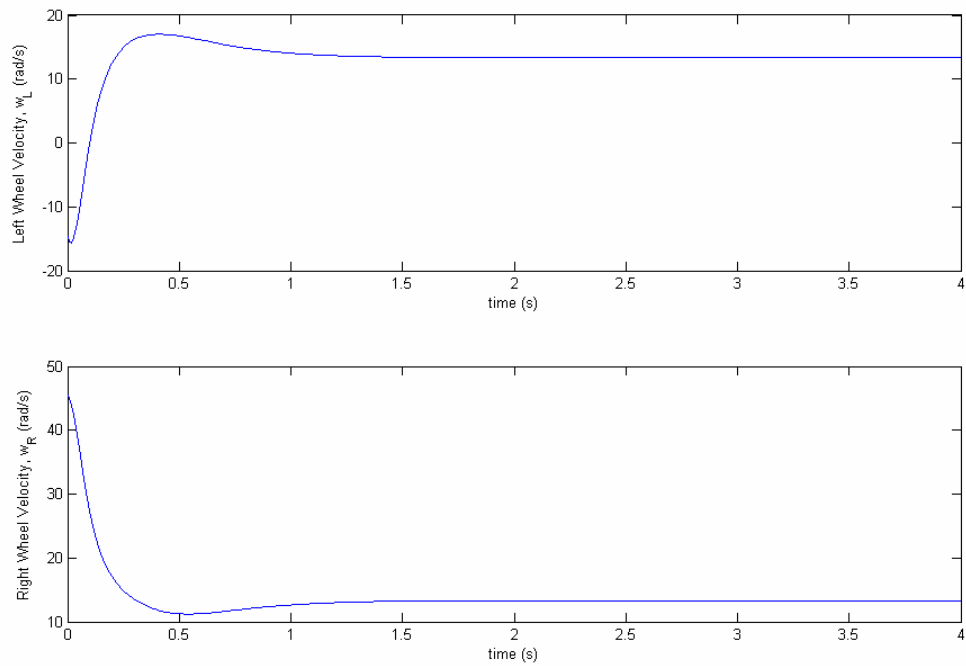
**Figure 4.23:** Actual Trajectory (blue) versus Desired Trajectory (red) of Robot



**Figure 4.24:** States X (m), Y (m) and  $\theta$  (Degrees)



**Figure 4.25:** Inputs, Linear velocity,  $v$  (m/s), and Rotational velocity,  $\omega$  (rad/s)



**Figure 4.26:** Wheel Velocities,  $\omega_L$  (rad/s) and  $\omega_R$  (rad/s)

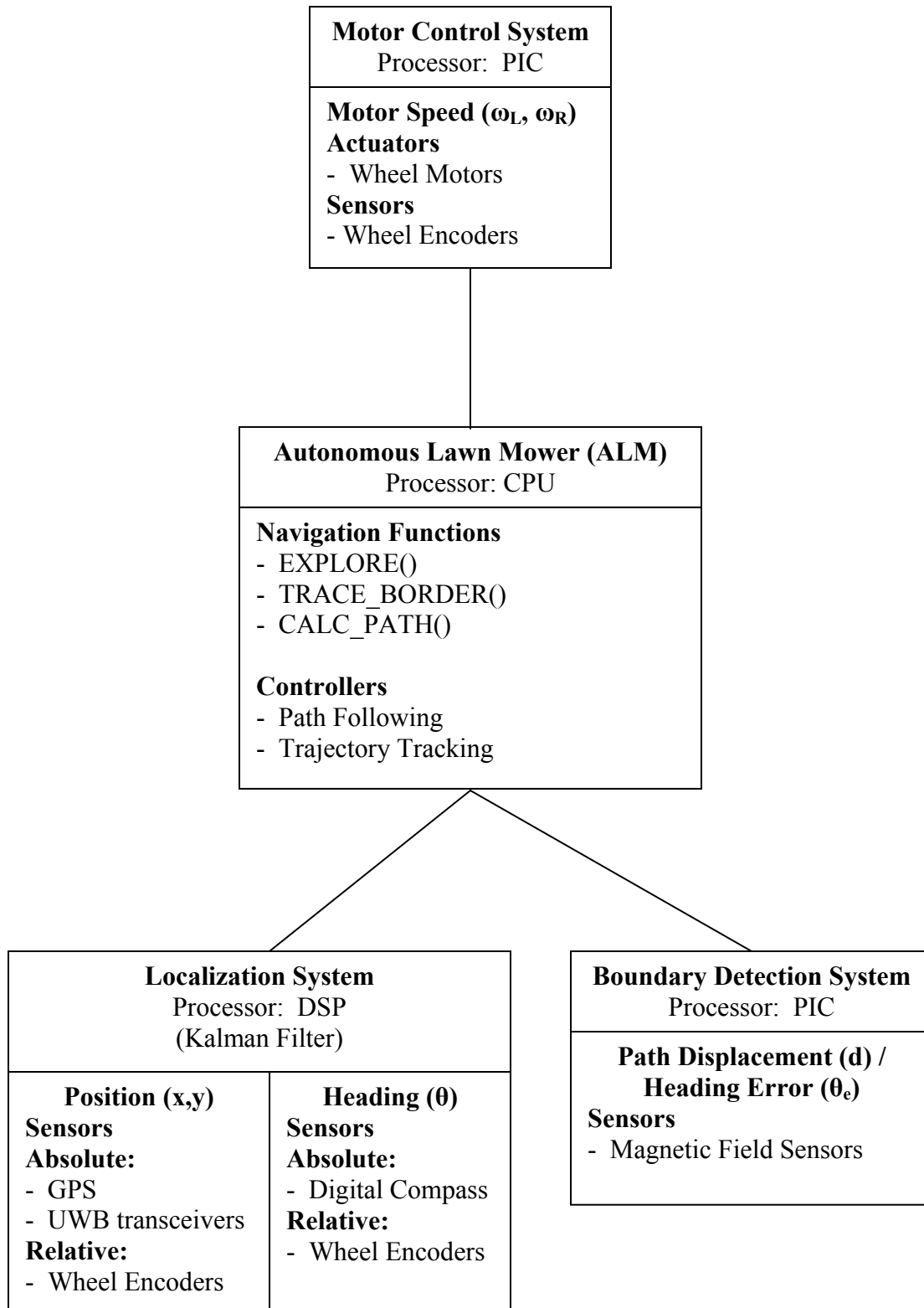
# Chapter 5

## 5 Hardware Implementation

### 5.1 Overall System Structure

While the act of mowing a lawn may seem simple, this thesis has demonstrated that the theory behind automating this process is quite complex. Creating an ALM would require the combination of numerous actuators, sensors, and microprocessors, as well as an extensive amount of software. This chapter will attempt to address some of the hardware issues as well as discuss some of the problems that arise when physically implementing such a system.

Before discussing the underlying components of the system, a broad architectural understanding of the ALM must be obtained. The ALM would require a number of systems acting in unison to achieve the desired functionality. Figure 5.1 is a functional block diagram of the ALM and displays the various subsystems the ALM would require. Figure 5.1 gives a broad architectural overview of the overall system.



**Figure 5.1:** Functional Block Diagram of ALM



As can be seen from the diagram above, the ALM requires a number of subsystems. The following sections discuss these various subsystems and their actual hardware implementation.

## 5.2 Microprocessors

Microprocessors are a key component of the overall design of the ALM, and serve as the brains behind all of the control. They are responsible for interpreting sensor data, making decisions based on that data, and generating actuator commands. Microprocessors constantly monitor the system's state and make decisions and perform actions based on that state. When programmed right, microprocessors are extremely fast and efficient at making calculated decisions. A microprocessor processes data at rates as high as a billion times a second. It is for these reasons that microprocessors are so widely used in so many different applications.

Every subsystem of the ALM would require its own microprocessor. Since cost would be an issue different types of processors would be used for the different systems within the ALM. A table of the systems, as well as what type of microprocessors they might use is displayed in Figure 5.2 below.

| System / Subsystem                | Processor Type        | Reason  |
|-----------------------------------|-----------------------|---|
| ALM Navigation and Control System | Microprocessor or DSP | This system needs to handle all the high level logic and control of the system. To implement the Path Following and Trajectory Tracking Controller some complex mathematical computations are required. For convenience it would need to be written in a high level language such as C. |
| Motor Control System              | PIC                   | This subsystem is very basic and only needs to implement the functionality of a PID controller. This subsystem's functionality would likely be written in assembly code.  |
| Boundary Detection System         | PIC                   | This subsystem serves as a basic interface between the magnetic sensors and the main system. Minimum computation is required and the code could be written in assembly code. If enough input pins exist on the main processor, a separate processor might not be needed.                |
| Localization System               | DSP                   | The most complicated subsystem with several sensor inputs. An extensive amount of complex mathematical computation is required. Implementation would be easiest with a DSP written in C.  |

**Figure 5.2:** Microprocessor requirements of the ALM subsystems

From the table above it is apparent that numerous microprocessors would be needed for the various subsystems of the ALM. However, these subsystems are not independent of each other and there would have to be a way for each subsystem to communicate and relay data to one another. One of the easiest ways of doing this is by using a shared bus architecture. There are a number of different protocols that use a shared bus for communication, but one of the most frequently used is the I<sup>2</sup>C protocol.

I<sup>2</sup>C, the Inter-IC bus, is a control bus developed by Philips in the 1980s. I<sup>2</sup>C is a simple two-wire bus with a software-defined protocol that provides various data transfer rates, up to 3.4 Mbps. I<sup>2</sup>C allows several devices to be connected on the same bus. I<sup>2</sup>C has software-controlled collision detection and arbitration to prevent data corruption and ensure reliable performance. I<sup>2</sup>C is one of the easiest and most practical bus architectures. Due to these benefits, I<sup>2</sup>C would likely be used for subsystem communication within the ALM [22].

## 5.3 Sensor and Actuator Subsystems

Sensors and actuators play a key role in the detection and reaction capabilities of the ALM. Sensors detect the state of a system, while actuators serve as a means to update the state. Sensors and actuators greatly affect the performance of the system. Practical, cost effective, and accurate sensors are vital to the success of the overall system. Sensors and actuators make up the three main subsystems of the ALM: the localization subsystem, the boundary and obstacle detection subsystem, and the motor control subsystem.

### 5.3.1 Localization

Localization describes a robot's ability to estimate its position and heading based on its sensor measurements. Given the application of the ALM, it is desired that the ALM be able to estimate its position and heading to within a few centimeters and degrees. The accuracy of a robot's localization system is dependent on the sensors it uses, its system dynamics, and the localization algorithm it implements. Localization schemes can generally be classified as relative or absolute depending on the type of sensors used [16].

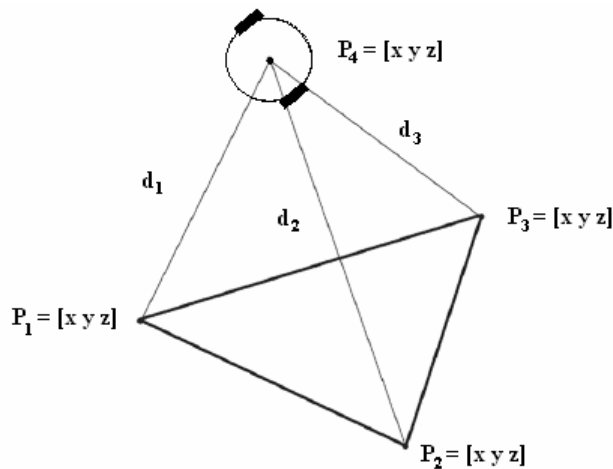
Relative localization, often referred to as dead reckoning, provides localization by integrating internal system sensor measurements. Wheel encoders, gyroscopes, and accelerometers are examples of relative sensing devices. Relative sensing allows for high accuracy and high bandwidth over short durations. Unfortunately, error from unmodeled systems dynamics grows quickly and unbounded over time. Therefore, relative localization cannot be utilized as a robot's sole localization scheme.

Absolute localization provides a way for a robot to obtain positioning information external of its system model. Absolution localization comes from “exteroceptive” sensing based on the environment around the robot. A robot utilizes known landmarks to determine its position in its global coordinate frame. GPS, vision, and transceiver systems are all examples of absolute localization. A great advantage of absolute localization schemes is that their measurements are obtained without using any previous sensor information and therefore their error is bounded. Unfortunately, these systems are much lower bandwidth than relative localization schemes. Also, these systems are greatly dependent on the geometrical arrangement of the landmarks relative to the robot. If the placement of landmarks is not optimal, the accuracy of these type of systems quickly degrade or fail all together.

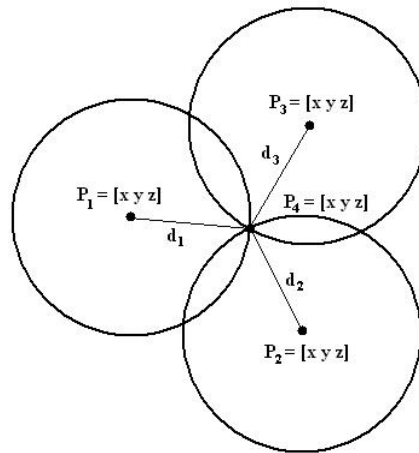
When examining the benefits and disadvantages of these two types of systems, it becomes obvious that combining these sensing techniques would be ideal. In the short term, relative localization could provide accurate localization. While less frequent updates using absolute localization would provide a necessary bound to the relative localization error. Combining these two localization schemes is referred to as sensor fusion. Sensor fusion, takes all the available sensor data, relative or absolute, and combines it into a probabilistic estimate of the robot’s position and orientation. One of the most widely used techniques for sensor fusion is through the use of a Kalman filter.

### 5.3.1.1 Position

In theory, absolute positioning can be achieved by a method known as Trilateration. Trilateration uses simultaneous range measurements from three known points in space to an unknown point in space. As can be seen in Figure 5.3, a robot at unknown location ( $P_4$ ) can determine its location if it knows how far away it is ( $d_1, d_2, d_3$ ) from three known locations ( $P_1, P_2,$  and  $P_3$ ). Mathematically, given this information, this problem simplifies to determining the point of intersection of three spheres, shown in Figure 5.4 [26].



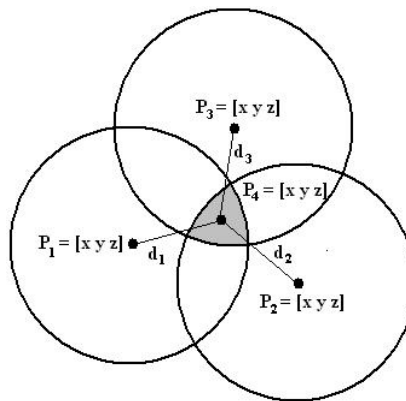
**Figure 5.3:** Physical representation of Trilateration



**Figure 5.4:** Mathematical representation of Trilateration

Trilateration is commonly implemented using a range estimation method known as Time-of-Arrival (TOA). This method uses radio transmitters and receivers called transceivers. TOA utilizes the fact that a signal from one transceiver to another will propagate at the speed of light. Therefore, knowing the speed of light and the time it took for the signal to propagate between transceivers allows the distance between the transceivers to be calculated.

Unfortunately, there are several practical considerations when using TOA. The accuracy of TOA is highly dependent on clock synchronization and the time resolution of the transceivers. Furthermore, sources of error such as multipath propagation, non-line-of-sight (NLOS), and multiple access interference all affect the range estimates. Given these inaccuracies, the calculated spheres of trilateration do not converge to a point but to an area, as can be seen in Figure 5.5. The size of this area reflects the relative accuracy of the positioning system. There are currently two systems based on trilateration that have the desired accuracy required for the ALM.



**Figure 5.5:** Effects of TOA error on Trilateration

The Global Positioning System, GPS, was originally created by the US military in the 1970s. GPS consists of a constellation of 24 satellites that are constantly revolving around earth and broadcasting positioning information. GPS receivers are commonplace today and are used in both military and commercial applications. Given the signals a GPS receiver obtains from the orbiting satellites, it can then calculate its position on earth. A GPS receiver needs information from four satellites in order to resolve its 3D position in space and time [20].

Due to errors such as Ionospheric and Atmospheric delays, satellite and receiver clock errors, and multipath issues, inexpensive commercial GPS receivers usually have an accuracy of around 30 to 50 meters. However, additional hardware and infrastructure can be added to reduce these errors and obtain much higher accuracy GPS systems.

The only currently existing implementation of GPS that can obtain centimeter level accuracy is known as Real-Time Kinetic (RTK) GPS. RTK GPS requires the installation of a reference station, which remains at a fixed and known location close to the mobile GPS receiver. Through a radio link the reference station transmits data to the mobile GPS receiver. The mobile GPS receiver receives this data as well as data directly for its own GPS unit and processes it. Since the two GPS units are being operated in very close proximity of each other and the receiver at the reference station remains at a fixed location, errors and positioning ambiguities can be significantly reduced [20].

RTK GPS has an accuracy of 1 to 5 cm, making it very appealing for use on the ALM. Unfortunately, due to the additional hardware needed, the complexity of the data processing, and the high cost (upwards of \$10,000), this is not a viable commercial solution for the ALM.

Another technology that has potential for high accuracy localization is Ultra Wide Band (UWB). A UWB signal approximates the characteristics of an impulse and is tightly regulated in the time domain. Due to this fact, UWB signals have a very large bandwidth. A key limitation of current sine wave positioning systems is their time resolution. According to AEther Wire, “with conventional sine wave technology, the bandwidth of the signal relative to its carrier frequency is very small, at most a few percent. On the other hand, UWB has a relative bandwidth approaching 100%.” If utilized, UWB has the potential to create highly accurate positioning system [1].

Additional benefits of UWB include,

- UWB can be totally integrated in CMOS circuits. Therefore, transceivers can be made small (coin sized), low power, low weight, and low cost (a few dollars).
- UWB antennas can be made equally small and driven by CMOS.
- UWB requires minimal transmit power and is spread across a large frequency range. Therefore, it can coexist and not interfere with current sine wave technology.

- As with any spread spectrum technology it has features such as multipath immunity, tolerance of interference from other radio sources, and low probability of intercept.
- UWB signals have very good penetrating capabilities and can operate within buildings, urban areas, and forests.

This leads to a very accurate, practical, and cost effective solution to localization. Unfortunately, UWB has yet to be utilized commercially mostly due to politics. It was only recently, in 2002, that the FCC licensed space for UWB. However, there are strict regulations on the power levels UWB can be transmitted at. Current regulations limit UWB's range to approximately 10 m. Hopefully through further research and understanding, these restrictions will be loosened and the full potential of UWB can be realized. UWB is an ideal solution for mobile robot localization and for use with the ALM. Unfortunately, due to regulation and lack of commercial development, only time will tell if it will be an available option for localization.

### **5.3.1.2 Orientation**

The easiest and most practical way to obtain absolute heading information would be through the use of a digital magnetic compass. Digital magnetic compasses are made by a variety of vendors and offer a variety of resolutions and features. A digital magnet compass works by detecting the horizontal component of the earth's magnetic field. Various sensor technologies exist for detecting this field. The best type of magnetic compass for outdoor robotic applications is the Fluxgate compass. The Fluxgate compass is known for low power consumption, no moving parts, its intolerance to shock and vibration, rapid start up, and relatively low cost [3].

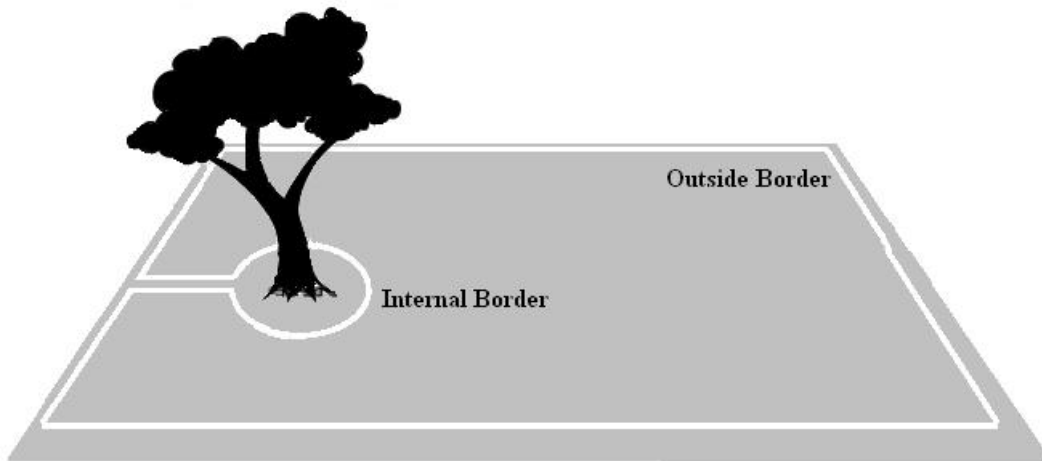
Unfortunately, the performance of these devices is highly dependent on the slope of the terrain. The accuracy of these devices can be increased if they are gimbal-mounted and mechanically dampened. An alternative to mechanically leveling the system is to have the sensing device float in an inert liquid, thus keeping the device level for moderate inclines.

Current commercial digital compasses can achieve heading measurements with accuracies of  $\pm 0.5$  degrees, update rates up to 10 Hz, and can handle inclines up to  $\pm 16$  degrees [18].

This would likely be a viable solution for obtaining absolute heading measurements for the ALM. However, further experimentation would have to be done to make sure that this sensor would not be affected by magnetic fields created by any of the electronics, sensors or motors onboard the system.

### 5.3.2 Boundary and Obstacle Detection

Boundary and obstacle detection could be handled in one of two ways. Both of these systems rely on the same basic premise; however, their detection mechanism differs. A boundary would be created by burying a wire around the perimeter of the yard as well as around places the mower was not intended to go. These include places such as flower beds, gardens, patios, and pools. As was discussed in Chapter 3, the perimeter of the yard corresponds to the “outside border” and the flower beds, gardens, and pools correspond to “internal borders”.



**Figure 5.6:** Diagram of Outside and Internal borders

These buried wires would then be interfaced with electronics that would allow them to either emit radio waves or a magnetic field. Sensors on the ALM would then detect these waves and know when the ALM had encountered a boundary. Numerous types of these systems have already been developed and commercially sold as “Invisible Dog Fences”.

Further tests would have to be done with these technologies to ensure that the needed accuracy could be acquired. This detection system would be used by the path following controller and the Trace\_Border behavior. A centimeter level accuracy would be needed in order to achieve desired path following performance.

Finally, if a magnetic field detection system was chosen, it would need to be verified that the magnetic field created by the buried wire did not interfere with the digital compass measurements. This could severely degrade the performance of the localization system and impact the overall operation of the ALM.

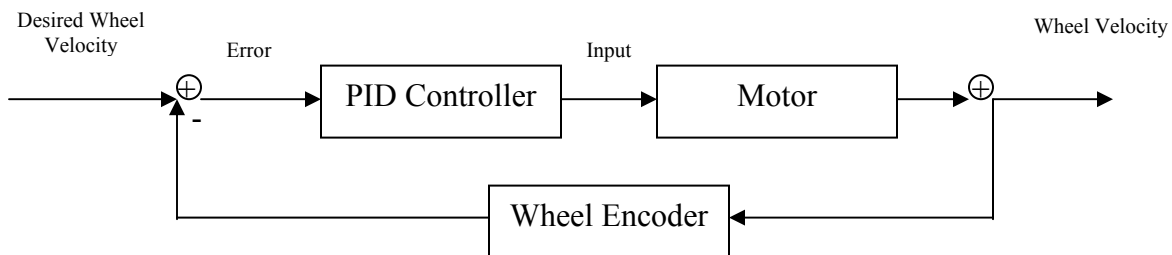
### 5.3.3 Motor Control

The final ALM subsystem controls the motors onboard the ALM. As was seen in previous chapter, the inputs of the path following and global coordinate model were the vehicle's linear and rotational velocities. Based on the vehicle's wheel radius and half wheel base,  $r$  and  $b$ , there is a one to one mapping between the vehicle's linear and rotational velocity,  $v$  and  $\omega$ , and the vehicle's wheel velocities,  $\omega_L$  and  $\omega_R$ .

The kinematic models and controllers used to derived the path following and trajectory tracking controllers assumed that the motors of the actual system were ideal. Based on this assumption, it is implied that the motors have an infinite resolution and can instantaneously change to any desired speed. However in actuality, these assumptions are not true. If a controller is implemented digitally, a motor only has a discrete number of speeds it can be set to. Furthermore, there is always a delay in the amount of time it takes for a motor to achieve a certain speed. Not taking these practical constraints into account in the actual implementation of the ALM could lead to undesired and unpredictable results.

Luckily, motor control has been extensively studied. Motors are simple electro-mechanical devices and several simple models exist. Furthermore, motors can be considered SISO systems. This simplifies controller design immensely, and there are a number of known techniques to meet motor performance criteria such as rise time, settling time, peak overshoot and steady state error.

One of the most frequently used controllers for motor control is the PID controller. This closed loop controller provides robustness and noise rejection while still allowing design flexibility. Wheel encoders would be used to sense the wheel speeds. A block diagram of the motor control system is shown in Figure 5.7.



**Figure 5.7:** Block diagram of the motor control system

The following subsections discuss the PID controller and the hardware used for the motor control system.



### 5.3.3.1 PID Control

PID controllers are one of the most frequently used controllers. PID control is an acronym for proportional, integral, derivative control. A PID controller is a closed loop controller that regulates a system based on its error signal. The error signal can be more specifically defined as the difference between the desired value of the signal and the current value of the signal. A PID controller has the following form,

$$u = k_p e + k_i \int e + k_d \dot{e}$$

where  $e$  is the error signal, and  $k_p$ ,  $k_i$ , and  $k_d$  are the gain constants of the relative error signal. It should be noted that if the controller is implemented discretely the controller can further be simplified by these approximations

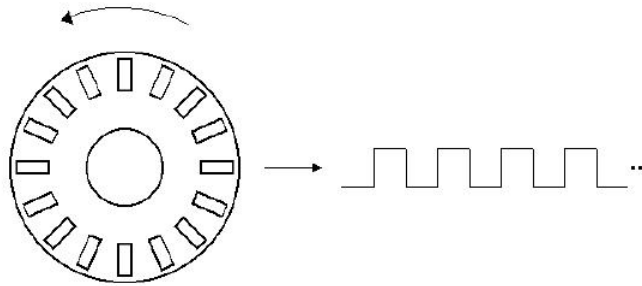
$$\int e \approx \sum e$$
$$\dot{e} \approx e_k - e_{k-1}$$

Using a PID controller allows a great deal of flexibility when achieving system performance and stability. The proportional constant,  $k_p$  reduces the rise time as well as reduces the steady state error. The integral constant,  $k_i$  eliminates the steady-state error but has a negative effect on the transient response. The derivative constant,  $k_d$  increases stability and reduces the overshoot of the transient response.

By fine tuning these constants the performance of the system can easily be adjusted. However, it should be noted that these constants are correlated and not completely independent. Attempting to affect a single performance criterion by adjusting one constant could negatively impact another performance criterion. Fine tuning a PID controller is a trial and error process.

### 5.3.3.2 Wheel Encoders

Wheel Encoders are the most practical, inexpensive, and frequently used sensors when measuring wheel speed. A wheel encoder is also sometimes referred to as an optical encoder. A wheel encoder consists of an optical disk with alternating transparent and opaque divisions. An example of this can be seen in Figure 5.9. The optical disk is connected to the shaft of the motor so it spins at the same angular velocity as the motor. An optical emitter and detector are placed on either side of the disk. The transparent parts of the disk allow light from the emitter to reach the detector thus causing the detector to turn on and off as the disk rotates. Therefore, the output of the encoder is a square wave whose frequency is proportional to the angular velocity of the wheel, as can be seen in Figure 5.8.

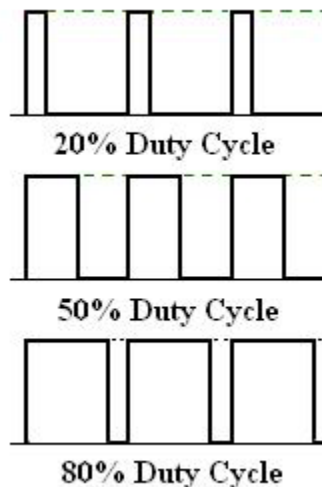


**Figure 5.8:** Diagram of a Wheel Encoder

Using a wheel encoder, the measurements can then be fed back to the PID controller to regulate the speed of the motor.

### 5.3.3.3 Pulse Width Modulation (PWM) and the H bridge

Most motors operate using a pulse width modulated (PWM) signal. A PWM signal has a fixed frequency and variable duty cycle. Therefore, in a given cycle the signal is “ON” for a certain period of time, and then “OFF” for the remaining time. The percent of the time the signal is “ON” is known as the duty cycle. The longer the signal is “ON” the more average power the motor receives, thus making it turn faster. The speed of the motor is directly proportional to the duty cycle. A diagram of a PWM signal and various duty cycles are shown in Figure 5.9.

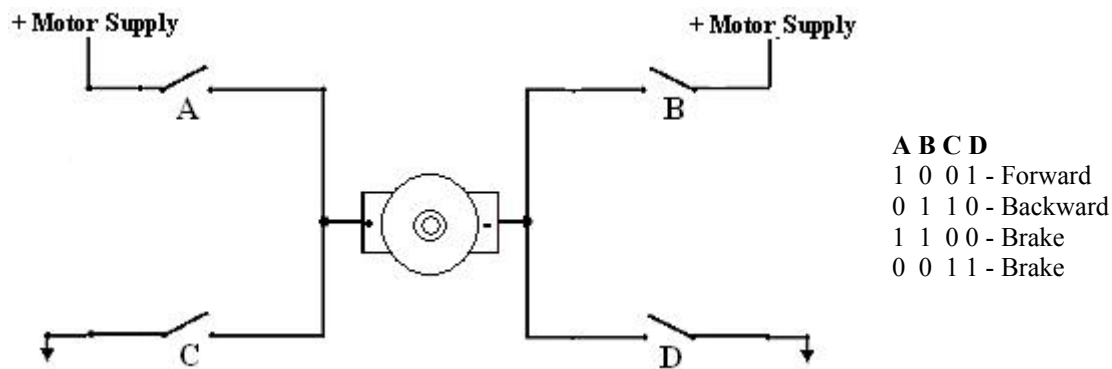


**Figure 5.9:** PWM Signal with various Duty Cycles

In order to create a PWM signal, usually a digital output of a microprocessor is used. The output pin of the microprocessor is turned “ON” for a certain amount of time and then switched “OFF” for the rest of the time in that period. Unfortunately, most

microprocessors are not able to supply the large amount of power required by most motors.

Therefore, an H-bridge is added to the circuit to alleviate this problem. An H-bridge is a component that separates a motor and its necessary power supply from the power sensitive microprocessor. An H-bridge allows the microprocessor to control the duty cycle of the power supply, while not requiring the microprocessor to actually supply the power. A typical H-bridge circuit is shown in Figure 5.10 below. As can be seen in Figure 5.10, the microprocessor controls switches (A,B,C,D) which serve as digital switches between a separate power supply and the motor [4].



**Figure 5.10:** H-bridge Circuit

The switches in the figure above can be implemented in a number of ways including relays, FETs, or Mosfets. There are also numerous inexpensive commercial H-bridge chips available today. As can be seen in the diagram, the direction of the motor can also be controlled based on which switches are activated. Therefore, an H-bridge is a cost effective way of controlling and supplying power to a motor.

## 5.4 Power

Power is also a critical part of the ALM design. There are numerous devices on the ALM that need various forms of power. While these problems are beyond the scope of this thesis, a few power issues and considerations will be mentioned.

One of the first things to consider when designing the power system for the ALM is what type of energy sources it will use. The bottom line when designing any product is, what is the most practical, cost effective, and efficient solution? Should the ALM use only electrical energy, through the use of a battery? Or perhaps it should use a combination of battery and gas power in much the same way a car does? Maybe it should run strictly on

solar power? These would all be questions that would have to be evaluated and answered before actual implementation of the ALM.

Choosing the energy sources of the ALM would also have a great impact on its recharging system. If the ALM used a gas engine to turn the mower blades, it would have to be refueled periodically with human intervention. However, by using a gas engine an alternator could be included in the system that could supply electric power to all the electronics and wheel motors.

In contrast, using a strictly electrical system would require the ALM to only have one central battery. The ALM could monitor the battery and autonomously find its way back to a recharging station. The ALM could then return to operation once its battery had been replenished.

Finally, solar power could be considered as a power source. Likely due to current technological limitations, solar power would not be the sole energy source for the ALM. However, a solar power system could be leveraged with one of the previously mentioned systems as a means for additional power or serve as a recharging system. Perhaps solar power could run the electronics on the system, or be used to recharge the battery power? Despite its limitations, solar power is an attractive solution when power conservation is a major issue.

Another major consideration in the actual implementation of the ALM is the ranges of power that are need. The electronics, microprocessors, and sensors all require relatively small power sources to operate. Usually these types of devices run on +5 and +12 volt powers supplies and use very little current. This is in stark contrast to the power hungry motors of the ALM. Motors usually required a great deal of power to run, usually +12 or +24 volts and several amps. Motors would be utilized to turn the wheels of the mower and possibly spin the mower blades. The distribution of power throughout the systems would have to be considered. Obviously, there are numerous power issues that have to be addressed before the ALM could be physically implemented.

# Chapter 6

## 6 Conclusions

### 6.1 Concluding Remarks

This thesis has effectively addressed the issues involved with navigation and control of an autonomous vehicle, specifically that of an autonomous lawn mower (ALM). Since lawn mowing was studied, a navigational scheme was created that optimized the lawn mowing behavior. A robust navigation algorithm was created for the ALM that covered a bounded region, while avoiding obstacles.

Translating this navigational scheme into motor commands required the use of complex control theory. Optimal control, path following, and trajectory tracking were all considered as possibilities and simulated in MATLAB. Based on the results, it was determined that a combination of both path following and trajectory tracking would be the most efficient control scheme for implementing the ALM's navigation algorithm.

Finally, the hardware and sensors required to implement the path following and trajectory tracking control schemes were discussed. Further software, sensor fusion, and hardware integration would need to be done in order to physically implement the ALM. However, the overall theory of using an autonomous vehicle to mow a lawn has been successfully completed in this work.

## 6.2 Future Work

Obviously, there is an abundance of work still to be done in order to physically implement the ALM. It is well known that there is fine line between theory and implementation of a system. Whole new problems arise and much more testing and debugging would be required to actually build the ALM. The following list is an attempt to point out some of the issues not discussed in this thesis

### **Theoretical Issues:**

- Further research would have to be done regarding the Travel Salesman Problem. An efficient method for solving these problems would need to be obtained in order to optimize the ALM's navigation system.
- A convenient and practical way to estimate the curvature in the path coordinate model must be determined.
- A robust way of dealing with unknown obstacles (both static and dynamic) must be research and studied. This would allow the ALM to avoid large rocks and trees without placing a border around them. Furthermore, it would provide a safety mechanism so the ALM would avoid people and pets in the yard. This research area is referred to as collision detection and obstacle avoidance.
- Further research in both nonlinear systems and hybrid systems would possibly provide additional control methodologies for the ALM. Further work should be done to evaluate the robustness of the system.
- Localization is very important in mobile robotics. Further simulation and experimentation would have to be done with Kalman filtering, in order to obtain an accurate position and heading estimate.

### **Practical Implementation Issue:**

- Hardware integration is a major implementation issue. Evaluating actual hardware and getting to work together would be a major challenge. Also, timing issues would have to be considered since data would be received at different rates. Further knowledge of Real Time Operating Systems (RTOS) would be necessary.
- Cost versus performance, as in any engineering design, would have significant impact on an actual commercial implementation of an ALM.
- Various testing and debugging schemes would have to be considered when initially implementing the system. A valid and practical method of measuring the performance of the system would have to be devised.

These are only a few of the issues and problems that would need to be addressed. Hopefully, through further work and technological progress, the ability to accurately and autonomously mow a lawn is not far from reality.

# Bibliography

- [1] AEther Wire & Location, Inc., “Ultra-Wideband Localizers,” <http://www.aetherwire.com/> (Nicasio, CA: AEther Wire & Location Inc, 1999).
- [2] Berger, Ari, *3D Programming for Windows. The 3DState Graphics Engine User Manual*, <http://www.3dstate.com>.
- [3] Borenstein, J., “Mobile Robot Positioning: Sensors and Techniques,” *Journal of Robotic Systems*, Vol. 14, Issue 4 (1997), pp. 231-249.
- [4] Brown, Jim, “Brief H-bridge Theory of Operation,” <http://www.dprg.org/tutorials/1998-04a/>, (Dallas, TX: Dallas Personal Robotics Group. April, 1998).
- [5] Cahlink, George, “War of Machines,” *Government Executive*, (15 July, 2004), <http://www.govexec.com/features/0704-15/0704-15s5.htm>.
- [6] Cao, Zuo Liang, Yuyu Huang, and Ernest L. Hall, “Region Filling Operations with Random Obstacle Avoidance for Mobile Robots,” *Journal of Robotic Systems*, Vol. 5 Issue 2 (1988), pp. 87-102.
- [7] Choset, Howie, “Coverage for robotics – A survey of recent results,” *Annals of Mathematics and Artificial Intelligence*, Vol 31 (2001), pp. 113-126.
- [8] Choset, Howie and Philippe Pignon, “Coverage path planning: The boustrophedon decomposition,” *Proceedings of the International Conference on Field and Service Robotics*, Canberra, Australia (December 1997).
- [9] Dao, James, Andrew C. Revkin, “Machines are Filling in for Troops,” *New York Times*, (16 April 2002), <http://www.cybracero.com/articles/machinetroops1.html>.



- [10] Fowler, Jonathan, "U.N. Predicts Boom in Robot Labor," *CBS News.com*, (20 October 2004),  
<http://www.cbsnews.com/stories/2004/10/20/tech/main650274.shtml>.
- [11] Friendly Robotics, Inc., "RoboMower," <http://www.robomowerusa.com/>.
- [12] Friendly Robotics, Inc., "RoboMow User's Manual,"  
<http://www.friendlyrobotics.com/NewSite/manual.htm>.
- [13] Hao, Yongxing, "A Practical Framework for Formation Planning and Control of Multiple Unmanned Ground Vehicles," University of Delaware, (May 2004).
- [14] ION Satellite Division, "ION Autonomous Lawn Mower Competition,"  
<http://www.ion.org/satdiv/alc/index.cfm>, (Dayton, OH: ION, 2004).
- [15] iRobot, "Roomba Robotic Floor Vac," <http://www.irobot.com/consumer/>.
- [16] Kiriya, Evgeni, "A Localization System for Autonomous Golf Course Mowers," McGill University, Montreal, (November 2002).
- [17] Kirk, Donald E., *Optimal Control Theory: An Introduction*, (Mineola, New York: Dover Publications, Inc., 1998).
- [18] KVH Industries, Inc., "KVH C100 Compass Engine Technical Specifications," (Middletown, RI: KVH Industries, Inc., 1996).
- [19] Laumond, Jean-Paul, *Robot Motion Planning and Control*, (Springer, 1998).
- [20] Leica Geosystems AG, "Introduction to GPS (Global Positioning System)," (Heerbrugg, Switzerland: Leica Geosystems AG, 1999).
- [21] Mellodge, Patricia, "Feedback Control for a Path Following Robotic Car," Virginia Tech, (April 2002).
- [22] Philips Semiconductor, "Philips Semiconductor's I<sup>2</sup>C-bus Information,"  
<http://www.semiconductors.philips.com/markets/mms/protocols/i2c/>, (Philips Semiconductor, 2005).
- [23] Pike, John, "RQ-1 Predator MAE UAV,"  
<http://www.fas.org/irp/program/collect/predator.htm>, (Federation of American Scientists, 6 November 2002).
- [24] Pike, John, "RQ RQ-4A Global Hawk,"  
[http://www.fas.org/irp/program/collect/global\\_hawk.htm](http://www.fas.org/irp/program/collect/global_hawk.htm), (Federation of American Scientists, 6 November 2002).

- [25] Samson, Claude, "Control of Chained Systems Application to Path Following and Time-Varying Point-Stabilization of Mobile Robots," *IEEE Transaction on Automatic Control*, Vol. 40, No. 1 (January 1995), pp. 64-76.
- [26] Thomas, Federico and Lluís Ros, "Revisiting Trilateration for Robot Localization," *IEEE Transactions on Robotics*, Vol. 21, No. 1 (February 2005), pp. 93-101.
- [27] Thorp, James, Virginia Tech Electrical and Computer Engineering Department Head, "Virginia Tech Autonomous Vehicle Funding," (Blacksburg, VA: 29 December 2004) *email to the author*.
- [28] Virginia Tech, "Unmanned Systems Group," <http://www.unmanned.vt.edu/>, (Blacksburg, VA: Virginia Tech, 19 October 2003).
- [29] Virginia Tech, "Virginia Tech Darpa Grand Challenge 2005" <http://www.me.vt.edu/grandchallenge/>, (Blacksburg, VA: Virginia Tech, 10 February 2005).
- [30] Wadoo, Sabiha, "Feedback Control and Nonlinear Controllability of Nonholonomic Systems," Virginia Tech, (January 2003).
- [31] Wikipedia, "Traveling Salesman Problem," [http://en.wikipedia.org/wiki/Traveling\\_salesman\\_problem](http://en.wikipedia.org/wiki/Traveling_salesman_problem).
- [32] Yuan, Jing, Yalou Huang and Qiang Han, "A Strategy of Path Following Control for Wheeled Mobile Robots," Proceedings of the 5<sup>th</sup> World Congress on Intelligent Control and Automation, (15-19 June 2004), *IEEE*, Vol. 6, pp. 4991–4995.

# Vita

Ian Josef Schworer was born on January 31, 1981 in Fairfax, Virginia. He attended Herndon High School and graduated in 1999. Ian entered Virginia Polytechnic Institute and State University, in August of 1999 and graduated summa cum laude in June of 2003. During this time, Ian completed an eight month co-op and a summer internship at Intel in Hillsboro, Oregon. Upon graduation, Ian was offered a Bradley Fellowship to continue his education at Virginia Tech. From August 2003 to May of 2005, Ian completed the requirements for an M.S. in Electrical Engineering. He also participated in a summer internship at Lockheed Martin. Ian has taken a consulting position with Booz Allen Hamilton, a technology consulting firm in Tyson's Corner, Virginia.