Path Following Control System for a Small Autonomous Vehicle

by

Raj Chander Midha

B.S., Mechanical Engineering, Purdue University, May 1999

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

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2001

© 2001 Raj Midha. All rights reserved.
The author hereby grants MIT permission to reproduce and to distribute publicly paper and
electronic copies of this thesis document in whole or in part.

Author............................................................................... 

Department of Mechanical Engineering

May 11, 2001

Certified by.................................................................

Edward Lanzilotta
Senior Scientist
The Charles Stark Draper Laboratory, Inc.
Thesis Supervisor

Certified by.................................................................

David Brock
Principal Research Scientist
Department of Mechanical Engineering
Thesis Supervisor

Accepted by........................................................................

Ain A. Sonin
Chairman, Committee on Graduate Studies
Path Following Control System for a Small Autonomous Vehicle

by
Raj Chander Midha

Submitted to the Department of Mechanical Engineering
on May, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

This thesis presents a control algorithm that enables an autonomous ground vehicle to follow a specified path. The vehicle is assumed to have knowledge of both the path geometry and its location relative to the path. The vehicle also has the ability to estimate its current kinematic state and to adjust its heading and longitudinal velocity. Based on these abilities, end-to-end path traversal time is minimized while path following error is maintained within allowable bounds.

A simulation of the control system is shown which verifies the algorithm as well as aids the selection of control gains for implementation of the algorithm. Implementation of the control system on a 1/10th scale radio-controlled car is presented. A kinematic vehicle model used in the implementation is shown and a Kalman filter used to interpret sensor data is given.

Technical Supervisor: Dr. Edward J. Lanzilotta
Title: Senior Scientist, Draper Laboratory

Thesis Advisor: Dr. David L. Brock
Title: Principle Research Scientist, Department of Mechanical Engineering
Acknowledgments

This thesis was prepared at The Charles Stark Draper Laboratory, Inc. under Internal Company Sponsored Research Project 3054, DLF Support.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

Raj C. Midha

I would like to thank all of the people at Draper Lab who helped me in my efforts, especially Mark Little and Steve Friend for explaining things to me very slowly so I could understand them. And thanks to the rest of the ‘kids’ office (Kara, Dave, Kyrilian, Carissa and honorary member Gary) for keeping me from working too hard.

Special thanks to my Draper Lab advisor, Dr. Edward Lanzilotta and my thesis advisor Dr. David Brock for their help and guidance with this work.

Finally, thanks to my parents Ashok and Christine for their support and encouragement.

Thank you for your interest.
# Table of Contents

Abstract ........................................................................................................................................... 3

Chapter 1 Introduction .................................................................................................................. 11

1.1 Overview .................................................................................................................................. 11

1.2 Background .............................................................................................................................. 11

1.3 Other Related Work ................................................................................................................... 12

1.4 Thesis Scope ............................................................................................................................. 14

Chapter 2 Control Algorithm ....................................................................................................... 15

2.1 Overview .................................................................................................................................. 15

2.2 Assumptions .............................................................................................................................. 16

2.3 Speed Control .......................................................................................................................... 17

2.4 Steering Control ....................................................................................................................... 19

2.5 Algorithm Development ......................................................................................................... 22

Chapter 3 Analysis and Modeling ............................................................................................... 31

3.1 Overview .................................................................................................................................. 31

3.2 Vehicle Model .......................................................................................................................... 31

3.3 Controller Design and Tuning ................................................................................................. 34

Chapter 4 Sensors and Signal Filtering ....................................................................................... 39

4.1 Overview .................................................................................................................................. 39

4.2 Background .............................................................................................................................. 39

4.3 Signal Filtering .......................................................................................................................... 41
Chapter 5 Simulation

5.1 Overview .................................................................45
5.2 Control System Simulation .........................................45
5.3 Simulation of Kalman Filter ........................................48

Chapter 6 Hardware Implementation

6.1 Overview ..................................................................51
6.2 Sensors ....................................................................51
6.3 Vehicle ......................................................................52
6.4 Electronics ..................................................................53

Chapter 7 Results and Conclusions

7.1 Summary of Results ..................................................55
7.2 Future Work .............................................................55
7.3 Conclusions ...............................................................56
Figures and Tables

Figure 1: Example of a Situation for a Path-following Autonomous Vehicle......................... 15
Figure 2: Diagram of Path Representation............................................................................. 16
Figure 3: Velocity vs. Time Profile for Case 1 ...................................................................... 18
Figure 4: Velocity vs. Time Profile for Case 2 ...................................................................... 18
Figure 5: Steering Control System Block Diagram................................................................. 20
Figure 6: Illustration of different control options.................................................................. 21
Figure 7: Example of discretized path.................................................................................... 23
Figure 8: Velocity vs. Time Profile for Case 1. ..................................................................... 25
Figure 9: Velocity vs. Time Profile for Case 2. ..................................................................... 26
Figure 10: Determination of path steering direction............................................................... 28
Figure 11: Determination of heading steering direction. ......................................................... 29
Figure 12: Vehicle Kinematic Model Diagram....................................................................... 32
Figure 13: Vehicle Control Model Diagram. ......................................................................... 34
Figure 14: Control System Model Block Diagram. ................................................................. 35
Figure 15: First Quadrant Linearization of the Sine of an angle........................................... 36
Figure 16: Simulation Test-path Examples............................................................................. 45
Figure 17: Position of the simulated vehicle as it follows the path........................................ 47
Figure 18: Distance between vehicle and the path vs. distance traveled. ............................ 47
Figure 19: Vehicle speed vs. time. .......................................................................................... 48
Figure 20: Simulation of vehicle response with 2cm noise in encoder measurements and 1m noise in GPS measurements. ................................................................................. 49
Figure 21: Simulation of vehicle response with 20cm noise in encoder measurements and 1m noise in GPS measurements. ................................................................. 50

Figure 22: Wheel Rotation Measurement System ........................................................................ 52

Figure 23: Selected Test Vehicle - Associated RC10T3 Sport Truck ........................................ 53

Table 1: Variables used in Explanation of Control Algorithm .................................................. 22

Table 2: Variables used in Explanation of Control and Vehicle Models .................................. 34

Table 3: Inputs to Path Control System Simulation ................................................................. 46

Table 4: Vehicle Specifications .............................................................................................. 53
Chapter 1 Introduction

1.1 Overview

This thesis presents a time-optimal path following control system for a small autonomous vehicle. The algorithms developed allow for travel along curvilinear pathways and the adjustment of vehicle response based on mission objectives. A simulation of the control system was developed using a commercial modeling package. A control model was developed to aid in the selection of control parameters and a vehicle model was created to allow implementation of the algorithm on a vehicle. The system was implemented in hardware using a 1/10th scale radio-controlled car and a single board computer (SBC) which processed the sensor measurements and controlled the vehicle actuators.

1.2 Background

Draper Laboratory is currently developing a remotely operated surveillance ground vehicle (Throwbot) under the Tactical Mobile Robotics (TMR) program to the Defense Advanced Research Projects Agency (DARPA).

The vehicle is small enough to be carried by a single infantry soldier in a cargo pocket or butt pack. The soldier can then throw the vehicle through a window, where the vehicle provides remotely-operated audio/visual surveillance.

One of the required capabilities of the Throwbot vehicle is the ability to operate autonomously. Because the vehicle will be tele-operated using radio frequency (RF) communication there is a risk that it may be directed to a position at which radio contact is lost...
and RF control of the vehicle motion is no longer possible. Therefore, the vehicle must be able to recover from the loss of communication by traveling autonomously to a location at which communication can be re-established. Autonomous vehicle motion will also be required in the event that the vehicle must quickly respond to danger. The operator would need to instruct the vehicle to travel as quickly as possible to a certain safe point.

Autonomous motion of the vehicle in both of these cases requires both path generation and vehicle motion control capabilities. Given a path, the vehicle speed and direction must be controlled. Performance specifications such as deviation from the path, power consumption, and speed around the path can be optimized through the selection of the control algorithm.

1.3 Other Related Work

The first major study of unmanned ground vehicles (UGVs) was a DARPA funded project called Shakey, developed in the late 1960s at the Stanford Research Institute [4]. Shakey was equipped with a steerable TV camera, an ultrasonic range-finder, and touch sensors. It communicated remotely with a mainframe computer that commanded the vehicle. This project reemerged in the early 1980s as the DARPA Autonomous Land Vehicle (ALV) Program.

The ALV was built on an eight-wheel all-terrain vehicle capable of speeds of 72 km/h on the highway and 29 km/h on rough terrain. Sensors included a color video camera and a laser scanner for distance estimation. Video and range data were used to determine road-edge information that was used to direct the vehicle. By 1987, the vehicle was able to drive autonomously at an average speed of 14.5 km/h over a 4.5 km course through varying pavement types, road widths, and shadows, while avoiding obstacles.
Automated Guided Vehicles (AGVs) are commonly used in industrial settings to transport workpieces, and in warehouses to transport inventory. These systems generally follow a fixed guidepath embedded in the floor. Based on the distance between the guidepath and the vehicle, control signals are generated and sent to a regulating unit that uses a proportional controller to initiate a course correction [11].

The problem with this type of system is that it is not very flexible because installation of guidepaths requires cutting through the floor. A system that could follow a path to transport materials, while being flexible to path changes would be very beneficial.

The National Automated Highway System Consortium (NAHSC) has done a significant amount of work on autonomous vehicles to demonstrate the feasibility of an automated highway system, demonstrated in San Diego in 1997 [18]. The system uses one-inch diameter magnetic nails embedded into the center of automated highway lanes to provide feedback to vehicles on their lateral position and longitudinal speed.

Magnetometer sensors are mounted on the front and rear bumpers of the vehicle to determine position and speed. Radar sensors are mounted to the front bumper to determine the location of other vehicles. This information is sent to an on-board computer that issues commands to the car’s steering, braking and throttle actuators.

Vehicles are guided under lateral and longitudinal control. Lateral steering control keeps the car in the center of the lane and allows the vehicle to perform lane change maneuvers. Longitudinal control adjusts the spacing in front of and behind the vehicle as well as the vehicle speed. Hingwe and Tomizuka [7] designed a vehicle control system for lateral and yaw control using a bicycle model for path following.
The benefits of an automated highway system are many. Crashes related to driver error would be significantly reduced. The capacity of existing freeways would be increased and traffic congestion would be relieved. Fuel consumption would be reduced, and driver comfort and convenience would be increased.

Several issues surround the implementation of autonomous vehicles where an infrastructure like wire guidepaths or the automated highway is not available. The vehicle needs to both sense its surroundings and use this information to generate paths that avoid obstacles. If a detailed environment map is not available, the vehicle must construct one using sensor data. Brock et al. [2] studied and devised various obstacle avoidance and path generation algorithms.

1.4 Thesis Scope

Chapter 2 provides an overview of the vehicle control algorithm, including both longitudinal control of the vehicle to optimize speed and lateral control of the vehicle. Assumptions about the vehicle are listed, and the details of the algorithm are presented.

Chapter 3 presents the analysis and modeling of the control algorithm. A kinematic vehicle model is shown and equations for controller gain selection are introduced.

Chapter 4 considers several vehicle position sensors and the filtering of the position signals obtained from these sensors.

Chapter 5 provides results of a simulation of the proposed control system.

Chapter 6 presents hardware implementation of the control system including vehicle characteristics, sensors, and signal filtering.

Chapter 7 summarizes the results of the work done and conclusions and conclusions that can be drawn as well as opportunities for future work.
Chapter 2 Control Algorithm

2.1 Overview

The fundamental problem is to define a control algorithm that will optimize the end-to-end path speed of an autonomous path following vehicle. It is assumed that the control system will have some knowledge of the environment and the desired path within that environment, as well as some means for estimating its kinematic state relative to that environment. It is also assumed that the control system has the means to modulate the vehicle speed and heading. The control system is designed for a vehicle path trajectory that includes travel around curved paths. The vehicle uses coordinated longitudinal acceleration and deceleration as the curves are approached and departed. This offers a higher level of overall path speed performance relative to the more typical “point and shoot” vehicle control algorithms used in small robotic vehicles. Figure 1 shows an example of a path generated to avoid obstacles and reach a destination point.

Figure 1: The vehicle follows a given path to avoid obstacles and reach its destination.
2.2 Assumptions

Certain assumptions were made about the information available to the vehicle and about the capabilities of the vehicle. It was assumed that the desired path consists of a set of 'path segments' connected at the endpoints, as shown in Figure 2. These path segments exist in two-dimensional space. Each segment $i$, has a fixed start point in x-y coordinates $(x_i, y_i)$, a fixed length $l_i$, and a constant curvature in heading $\rho_i$, where curvature is defined as the inverse of curve radius [9]. The orientation of the end of each path segment is the same as the orientation of the beginning of the next segment so that the direction along the path is continuous. The vehicle has knowledge of the length, curvature, and orientation of the current path segment as well as that of the next path segment. The vehicle also has knowledge of its location and heading relative to the path.

![Figure 2: The path is made up of a set of constant curvature path segments.](image)

It is assumed that vehicle has no minimum turn radius and can follow a path of any curvature. The vehicle also has an absolute maximum speed. The vehicle is also assumed to have maximum prescribed lateral and longitudinal accelerations. The maximum lateral acceleration limits the speed of the vehicle depending on the path curvature. The maximum
longitudinal acceleration would be the same in both braking and accelerating. This assumption was made based on the friction-circle tire model, which says that the maximum force that a tire can provide is the same in all directions [6].

The following list summarizes the assumptions stated above.

1. Constant curvature path segments.
2. No discontinuities in path (tangency between path segments).
3. Vehicle has information about length, curvature, and orientation of current and next path segments.
4. No minimum turn radius for the vehicle.
5. Vehicle has maximum speed on straight segments.
6. Vehicle maximum speed on curved segments is a function of maximum lateral acceleration (which is a function of path curvature).
7. Vehicle maximum longitudinal acceleration is the same for accelerating and braking.
8. The control effort of the vehicle can be divided between two main functions: speed control and steering control.

2.3 Speed Control

Speed control attempts to maximize the speed of the vehicle around the path while making sure that the acceleration of the vehicle does not exceed pre-determined maximum values in the longitudinal and lateral directions. This control effort uses knowledge of both the current path segment and the next path segment to plan a time-optimal velocity profile for the vehicle. There are two different types of velocity profiles, shown in Figures 3 and 4 below. Case 1 occurs when the remaining length of the current segment is not sufficient to allow the
vehicle to attain its maximum speed before it needs to decelerate in preparation for the next segment. If the remaining length of the current segment is large enough, or if the difference in speeds between the maximum speed on the current segment and the speed on the next segment is small enough, Case 2 will occur. In both cases, it is assumed that the entry speed to the next segment is the maximum possible speed in that segment and that it is less than the maximum speed of the current segment.

By setting different values for $t_{0-1}$, $t_{1-2}$, and $t_{2-3}$, all possible velocity profiles for a segment can be managed by one of these two cases. To determine which case is applicable, the following steps must be taken:
1. Assume that case 2 exists.

2. Determine the distance that the vehicle must travel to decelerate from the maximum speed for the current segment to the maximum speed of segment 2.

3. Determine the distance that the vehicle must travel to accelerate from its initial speed to its maximum speed.

4. If the sum of these two distances is less than the length of the segment, case 2 exists. If the sum is greater than the length of the segment, case 1 exists.

2.4 Steering Control

Steering control can be divided into three basic categories: nominal control, path control, and heading control.

Nominal control uses information about the upcoming path curvature to predict the steering forces necessary to follow the path. The path curvature along with the vehicle model are used to estimate necessary steering forces.

Because the vehicle model is not exact and nominal control can not perfectly predict necessary steering forces, the steering must compensate for deviations from the desired path. Path control tries to minimize the distance between the vehicle and the path by multiplying the distance between the vehicle and the path (path error) by a controller gain. This value then determines the vehicle steering rate that will lead the vehicle back onto the path.

Heading control tries to minimize the difference between the heading angle of the vehicle and the path direction (heading error). The heading error is then multiplied by a controller gain. This value determines the steering rate to keep the vehicle parallel to the path. This control
effort acts as derivative control for the “path error” variable. Figure 5 shows a block diagram of the control system.

![Block Diagram of Control System](image)

**Figure 5:** Vehicle heading and position are compared to the desired path to determine steering inputs.

The path following strategy depends on the nature of the mission and the environment. Some of the mission and environment considerations include:

1. The types of obstacles the vehicle will be avoiding.
2. Knowledge of the location of these obstacles.
3. Importance of obstacle avoidance.
4. Importance of mission speed.

There are two extreme options for a vehicle that has strayed from the path as shown below. The first option is for the vehicle to turn sharply back toward the path in order to return to the path as quickly as possible. The second option is for the vehicle to continue traveling in the correct direction, but not on the path. This assumes that the vehicle’s deviation from the path is small, and large corrections are not necessary.
Figure 6: The vehicle can steer quickly to get back on the path or it can maintain proper heading.

The different mission objectives and variables dictate which option is preferred. For instance, if mission speed is the most important consideration, option 2 is used because it minimizes the time between the current location and the destination (assuming small deviation from the path and a goal that is larger than this deviation). However, if obstacle avoidance is critical, option 1 is used because it allows the vehicle to stay closer to the path.

The algorithm controls the nature of the vehicle's response to departure from the path through selection of path control and heading control gains. Path control multiplies the distance between the vehicle and the path (path error) by a controller gain to determine a vehicle steering rate that will lead the vehicle back to the path. Heading control multiplies the difference in heading angles between the vehicle and the path (heading error) by a controller gain to determine a vehicle steering rate that will keep the vehicle heading parallel to the path. This control effort acts as derivative control for the "path error" variable.
A response closer to option 1 will occur when the path control gain is very large and the heading control gain is very small. A response closer to option 2 will occur when the heading control gain is very large and the path control gain is very small.

2.5 Algorithm Development

This section explains the algorithm on a step by step basis. First interpretation of the path is outlined, followed by speed control and then steering control. To limit the computation necessary in executing the algorithm, the path segment information is converted to a series of equally spaced points on the path. The following table lists the variables used in the algorithm.

Table 1: Variables used in Explanation of Control Algorithm.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_{decel})</td>
<td>Distance vehicle would need to decelerate from the maximum speed on the current segment to the maximum speed on the next segment.</td>
</tr>
<tr>
<td>(d_{accel})</td>
<td>Distance vehicle would need to accelerate from the current speed to the maximum speed on the current segment.</td>
</tr>
<tr>
<td>(v_x)</td>
<td>Speed of the vehicle at point (x).</td>
</tr>
<tr>
<td>(t_x)</td>
<td>Time at which vehicle reaches point (x).</td>
</tr>
<tr>
<td>(d_x)</td>
<td>Distance between start of path and point (x).</td>
</tr>
<tr>
<td>(d_{segment})</td>
<td>Length of the current path segment.</td>
</tr>
<tr>
<td>(a)</td>
<td>Maximum acceleration of the vehicle, accelerating and braking.</td>
</tr>
</tbody>
</table>
2.5.1 Path Interpretation

Before the vehicle control begins, the desired path to follow must be created. The desired path can be changed continuously based on changing environmental conditions or mission objectives.

A. Create an array describing each path segment length, curvature, and orientation.

B. Convert the array into a set of discrete points on the path (as shown in Figure 6 below).

1. Decide the spacing between the points on the path.

2. At each point, determine the x and y coordinates of each path point, the curvature and heading of the path at that point, and within which segment that point resides.

![Figure 7: The path is discretized to facilitate computation.](image)

2.5.2 Speed Control

The vehicle speed is controlled to minimize the time it takes the vehicle to travel the path while not allowing the vehicle to exceed its maximum speed or acceleration in any direction.
A. Determine which of the two cases, as illustrated in Figures 3 and 4, exist when going from a path segment of larger curvature to a path segment of lesser curvature.

1. Assume that case 2 exists.

2. Determine the distance that the vehicle must travel to decelerate from the segment 1 maximum speed to the segment 2 speed,

\[ d_{\text{decel}} = \frac{v_2^2 - v_3^2}{2a} \]  

(1)

3. Determine the distance that vehicle must travel to accelerate from its initial speed to the segment 1 maximum speed,

\[ d_{\text{accel}} = \frac{v_1^2 - v_0^2}{2a} \]  

(2)

4. If the sum of these two distances is less than the length of the segment, case 2 exists. If the sum of these two lengths is greater than the length of the segment, case 1 exists.

If \( d_{\text{decel}} + d_{\text{accel}} < d_{\text{segment}} \) ... Case 2

If \( d_{\text{decel}} + d_{\text{accel}} > d_{\text{segment}} \) ... Case 1

B. Use the appropriate formulas listed below to determine the speed profile.

For case 1, given \( d_0, t_0, v_0, d_2, \) and \( v_2 \), equations (3) – (6) can be used to find \( d_1, t_1, v_1, \) and \( t_1 \).
Figure 8: The vehicle accelerates until it reaches point $d_1$ and then it decelerates in preparation for segment 2.

\[
v_1 = \sqrt{ad_2 + \frac{v_0^2}{2} + \frac{v_2^2}{2}}
\]  \hspace{1cm} (3)

\[
t_1 = \frac{(v_1 - v_0)}{a}
\]  \hspace{1cm} (4)

\[
d_1 = \frac{(v_1 + v_0)t_1}{2}
\]  \hspace{1cm} (5)

\[
t_2 = \frac{v_1 - v_2}{a} + t_1
\]  \hspace{1cm} (6)

For case 2, given $d_0$, $t_0$, $v_0$, $v_1$, $d_3$ and $v_3$, equations (7) – (11) can be used to find $t_1$, $t_2$, $t_3$, $d_1$, and $d_2$. 
Figure 9: The vehicle accelerates to $d_1$, maintains the maximum speed on segment 1 until $d_2$, and then decelerates.

\[ t_1 = \frac{(v_1 - v_0)}{a} \]  \hspace{0.5cm} (7)  \\
\[ d_1 = \frac{v_1^2 - v_0^2}{2a} \]  \hspace{0.5cm} (8)  \\
\[ d_2 = d_3 = \frac{1}{2a} \left( v_1^2 - v_3^2 \right) \]  \hspace{0.5cm} (9)  \\
\[ t_2 = \frac{d_2 - d_1}{v_1} + t_1 \]  \hspace{0.5cm} (10)  \\
\[ t_3 = t_2 + \frac{v_1 - v_3}{a} \]  \hspace{0.5cm} (11)  \\

C. For Case 1, if $t_1 > 0$, accelerates. If $t_1 < 0$, decelerate.

D. For Case 2, if $t_1 > 0$, accelerate. If $t_1 < 0$, but $t_2 > 0$, keep speed constant.

If $t_1 < 0$ and $t_2 < 0$, decelerate.
2.5.3 Steering Control

The steering of the vehicle is controlled to both maintain the correct heading and to stay close to the path.

A. Determine Nominal Steering Effort.

1. Determine in which segment the vehicle resides and the radius of curvature of that segment.

2. Apply a steering control effort proportional to path curvature.

B. Determine Path Steering Effort.

1. Determine which point on the path is closest to the vehicle. This is done using the following algorithm: Given the point on the discretized path (shown in Figure 6) that is closest to the vehicle on the last cycle, check the distance between the vehicle and the next point on the path. If this is closer, check the next point on the path. Continue this process until the next point on the path is farther away than the previous point, and use the previous point as the point on the path that is closest to the vehicle.

2. Determine the distance between the vehicle and this point.

3. Determine which direction the vehicle needs to turn in order to reach the path.

The direction of steering effort is found by taking the cross product of the vehicle heading vector, \( \mathbf{v} \), and the vector between the vehicle and the closest path point, \( \mathbf{p} \), as shown in the figure below. The vehicle heading vector is found by looking at the last two vehicle locations. If \( \mathbf{p} \times \mathbf{v} \) is positive, the vehicle is on the left hand
side of the path and needs to turn right. If $p \times v$ is negative, the vehicle is on the right hand side of the path and needs to turn left.

![Diagram](image)

**Figure 10:** The cross product of the path vector and the vehicle heading vector determines which direction the vehicle must turn to correct path deviation.

4. Apply a steering control effort proportional to distance between vehicle and path.

**C. Determine Heading Steering Effort.**

1. Determine the heading vector of the path, $h_p$, by looking at the last two points on the path segment.

2. Determine the heading angle of vehicle, $v$, by looking at the last two locations of the vehicle.

3. Determine which direction the vehicle needs to turn in order to correct its heading. The direction of steering effort is found by taking the cross product of the vehicle heading vector, $v$, and the vector between the path heading vector, $h_p$, as shown in the Figure 10 below. If $h_p \times v$ is positive, the vehicle needs to turn right. If $h_p \times v$ is negative, the vehicle needs to turn left.
Figure 11: The cross product of the path heading vector and the vehicle heading vector determine which direction the vehicle must turn to correct heading error.

4. Apply a steering control effort proportional to difference between the vehicle and path heading angles.

D. Add the results for B, C, and D to determine the total steering control effort.
Chapter 3 Analysis and Modeling

3.1 Overview

Certain analysis and modeling was necessary to support the algorithm presented in the previous section and facilitate its implementation in hardware. A kinematic vehicle model was developed so that desired turning rates could be converted to vehicle inputs. A vehicle control model was developed to choose controller gains used in the algorithm.

3.2 Vehicle Model

A kinematic vehicle model was needed for two reasons: to convert the desired heading change into a steering input and to determine the location of the vehicle based upon output of the vehicle sensors.

The following assumptions about the vehicle were necessary to proceed with the vehicle kinematic model.

1. The vehicle is front-wheel steered.
2. The vehicle has a parallel steering geometry.
3. There is no longitudinal slip in rear wheels.
4. A differential allows the left and right side rear wheels to turn at different rates.
5. Left and right rear wheel speed are known.

Figure 12 shows a diagram of the vehicle model.
Figure 12: The vehicle kinematic model allows for conversion between sensor outputs and vehicle location as well as desired vehicle location and actuator inputs.

The following equations are used to relate the vehicle speed, \( v \), and steering angle, \( \alpha \), to the left and right rear wheel speeds, \( v_l \) and \( v_r \), and turn radius, \( R \), where \( l \) is the track length and \( l_{wb} \) is the wheelbase of the vehicle [8]. The relation between path radius and left and right wheel speeds is

\[
R = \frac{l}{2} \left( \frac{v_r + v_l}{v_r - v_l} \right),
\]

(12)

where \( l \) is the track length of the vehicle. The relation between path radius and the commanded steering angle \( \alpha \) is

\[
R = \frac{l_{wb}}{\tan \alpha},
\]

(13)

where \( l_{wb} \) is the wheelbase length of the vehicle. The speed of the vehicle is
\[ v = \frac{(v_r + v_i)}{2}. \] (14)

If the steering angle and vehicle speed are known, the resulting left and right wheel speeds and turn radius can be found using (13) and the following two equations:

\[ v_i = v - \frac{l_i y}{2R}, \] (15)

\[ v_r = v + \frac{l_i y}{2R}. \] (16)

From (15) and (16), the vehicle speed and turn radius can be found. Using this information, the change in vehicle x position and y position, \( \Delta x \) and \( \Delta y \), over a given time interval, \( \Delta t \), can be found using (17) and (18),

\[ \Delta x = v \Delta t \cos \frac{v \Delta t}{R} \] (17)

\[ \Delta y = v \Delta t \sin \frac{v \Delta t}{R}. \] (18)

Solving (15) and (16) for \( v \) and \( R \), and substituting into (17) and (18), the following equations result:

\[ \Delta x = \frac{(v_r + v_i)}{2} \Delta t \cos \frac{v \Delta t}{l_i \left(\frac{v_r + v_i}{2} \right)} \left(\frac{v_r + v_i}{2} \right), \] (19)

\[ \Delta y = \frac{(v_r + v_i)}{2} \Delta t \sin \frac{v \Delta t}{l_i \left(\frac{v_r + v_i}{2} \right)} \left(\frac{v_r + v_i}{2} \right). \] (20)
3.3 Controller Design and Tuning

To choose the path and heading control gains, the following simplified vehicle and control system models were constructed as illustrated in Figure 13.

![Figure 13: The vehicle control model allows for adjustment of control gains.](image)

Table 2 lists the variables used in the model.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Distance in the x-direction from the origin to the vehicle.</td>
</tr>
<tr>
<td>XD</td>
<td>Distance in the x-direction from the origin to the path.</td>
</tr>
<tr>
<td>e</td>
<td>Distance in the x-direction between the path and the vehicle.</td>
</tr>
<tr>
<td>θ</td>
<td>Heading angle of the vehicle relative to x-axis.</td>
</tr>
<tr>
<td>F_heading</td>
<td>Heading correction force applied to the vehicle.</td>
</tr>
<tr>
<td>F_path</td>
<td>Path correction force applied to the vehicle.</td>
</tr>
<tr>
<td>ζ</td>
<td>Damping coefficient of the system.</td>
</tr>
<tr>
<td>M_p</td>
<td>Overshoot percentage of the system.</td>
</tr>
<tr>
<td>ωn</td>
<td>Natural frequency of the system.</td>
</tr>
<tr>
<td>t_s,1%</td>
<td>1% settling time of the system.</td>
</tr>
</tbody>
</table>
The model is used to describe the dynamic response of the vehicle. The input to the system is the desired path, $x_D$. The vehicle is assumed to initially be off the path. The distance between the vehicle and the path, $e$, causes a path control force, $F_{\text{path}}$, to be generated. The difference in heading between the vehicle and the path, $\theta$, causes a heading control force, $F_{\text{heading}}$, to be generated. The combination of these forces causes the vehicle position, $x$, to change. Only positions and forces perpendicular to the desired path are considered. The velocity of the vehicle, $v$, is assumed to be constant. A free-body diagram of the vehicle leads to

$$-(F_{\text{heading},x} + F_{\text{path},x}) = m\ddot{x},$$

where $m$ is the mass of the vehicle, $x$ is the distance to the path, and $F_{\text{heading}}$ and $F_{\text{path}}$ are forces perpendicular to the vehicle.

The control forces are

$$F_{\text{heading},x} = K_p(x - x_D),$$

and

$$F_{\text{path},x} = K_h \theta,$$

where $K_p$ and $K_h$ are controller gains. A block diagram of this control system model is shown in Figure 14.

**Figure 14:** Total steering force is made up of path and heading steering efforts.
The change in position of the vehicle is a function of its heading, which is given by

\[ \dot{x} = v \sin \theta \approx \frac{2}{\pi} v \theta, \quad (24) \]

where we have linearized sine in the first quadrant for ease of computation. As shown in Figure 15, the linearization results in a maximum deviation of 36% from the sine function.

![Linearized Plot of sin(Theta)](image1)

**Figure 15:** A linearization of the first quadrant of the sine of an angle is used to simplify computation.

Taking the Laplace transform of (21) leads to

\[ \frac{X(s)}{X_D(s)} = \frac{K_p}{ms^2 + \frac{nK_p}{2v} s + K_p}, \quad (25) \]

where \( \frac{X(s)}{X_D(s)} \) is the transfer function between a change in the position of the path and a change in position of the vehicle.

Standard linear control equations for a second order system were used to choose the controller gains [5]. The gains were chosen to meet time domain specifications for overshoot (\( M_p \)) and settling time (\( t_s \)).
The damping coefficient of the system, $\zeta$, is chosen by specifying the acceptable maximum overshoot of the system, $M_p$. $M_p$ is the ratio of overshoot to input step size. For instance, for $M_p = 0.5$, when the vehicle starts $1\text{m}$ from the path, it would overcorrect its position by $0.5\text{m}$. The damping coefficient is given by

$$\zeta = \sqrt{\frac{\ln(1/M_p)^2}{\pi^2 + \ln(1/M_p)^2}}. \quad (26)$$

After finding the damping coefficient of the system, the natural frequency of the system can be specified by determining the desired 1\% settling time, $t_{s,1\%}$. This is the amount of time that it would take the vehicle to correct its position to within a distance of 1\% of the input step from the path. The natural frequency of the system is given by

$$\omega_n = \frac{4.6}{\zeta t_{s,1\%}}. \quad (27)$$

Given the natural frequency and damping ratio of the system, the path and heading control gains, $K_p$ and $K_h$, can be found using the following equations.

$$K_p = m\omega_n^2 \quad (29)$$

$$K_h = \frac{4}{\pi} \zeta \omega_n \nu m \quad (31)$$
Chapter 4 Sensors and Signal Filtering

4.1 Overview

One concept for Throwbot implementation of this control system utilizes a vision system to allow both path planning and determination of the location of the vehicle relative to the path. Such a system would take images from the camera mounted on the vehicle and determine where certain objects (both goals and obstacles) are located. This information would then be translated into a path for the vehicle to follow. A system that can accomplish this task is not currently available, but a hardware demonstration of the path control algorithm is still desirable. A substitute sensor or set of sensors is needed in anticipation of the vision system. This chapter discusses the options available as well as the use of a Kalman filter to combine data from the different sensors which are eventually chosen.

4.2 Background

One requirement of the path following algorithm is that the vehicle must know its position relative to the path. Sensors are needed to determine both of these parameters. The accuracy of either may be limited, depending on the sensors chosen.

There are two general categories of navigation systems: off-board and on-board. Off-board navigation systems requires external hardware exists that vehicle communicates with in order to determine location. While this is not a practical solution for an indoor reconnaissance vehicle in hostile territory, it is currently used in AGV applications and would be realistic for demonstration purposes.
Examples of off-board navigation systems are inductive guidepaths, optical guidepaths, and reference beacons. Inductive guidepaths are commonly used in AGVs. Wire is placed beneath the floor. An AC current runs through the wire, creating an electromagnetic field. When a coil passes through this field, a voltage is generated which is proportional to the strength of the electromagnetic field. This voltage is used to determine the position of an AGV relative to the wire in the ground.

Optical guidepaths replace the wire in an inductive guidepath with painted or tape-marked path on the floor. A reflectance sensor on the vehicle can then recognize the difference between the marked path and the floor, and determine the position of the vehicle relative to the path. One advantage of this system is that it is more easily changeable than inductive guidepath systems.

Reference beacons use triangulation of three or more distance measurements to determine the location of the vehicle. The most commonly used reference beacon system is GPS. GPS satellites transmit data to a GPS receiver that indicates the satellite location and the current time. The signals arrive at the receiver at slightly different times because some satellites are farther away than others. The distance to the GPS satellites can be determined by estimating the amount of time it takes for their signals to reach the receiver. When the receiver determines the distance to a certain number of satellites, it can calculate the receiver position.

Examples of on-board navigation systems are inertial navigation, environment imaging, and dead-reckoning. An inertial navigation system uses a gyroscope with its axis parallel to the direction of motion of the vehicle. When the vehicle changes its direction of motion, the gyroscope senses the acceleration perpendicular to the path. This acceleration can be integrated
twice to determine position deviation from the path. Inertial navigation systems are flexible, however, the error that they produce cannot be corrected and their cost is high.

Imaging systems use visual data about the environment to identify goals and obstacles and to plan safe paths. At this time their accuracy is limited and cost is very high.

Dead reckoning uses the present location of the vehicle and measurements of the rotation of each drive wheel to estimate its new position [15]. The error using this system is cumulative like an inertial navigation system, however the cost is much lower. Another drawback to this system is that if slipping occurs between the wheels and the ground (i.e. the wheels rotate but do not produce motion of the vehicle), the new position estimate of the vehicle will be incorrect.

The set of sensors that were chosen are a GPS receiver which senses vehicle position in combination with wheel encoders which use dead reckoning to determine vehicle position. More than one type of sensor is used because the accuracy of GPS receivers are limited and their update rates are often slow. Therefore dead reckoning can be used to determine vehicle location between updates.

4.3 Signal Filtering

Because two different sensors are used to determine the position of the vehicle, the data from both must be combined to give a best estimate of the vehicle position. A Kalman filter was used to accomplish this task [5]. A Kalman filter is an optimal recursive data processing algorithm. It is statistically optimal in that it minimizes the mean-square estimation error of the set of measurements. It is recursive because it does not require all previous data to be kept in storage and reprocessed every time a new measurement is taken.

The two data sources are the GPS receiver, which gives the location of the vehicle, and the two wheel encoders, whose outputs can be converted to an estimate of the change in position.
The accuracy of each of these measurements is different, and a Kalman filter is useful because it is able to combine each of these measurements to arrive at an optimal estimate.

The requirements for the Kalman filter are a system model, a measurement model, and initial conditions. The system model represents how the values being measured are expected to change with each passing measurement and the confidence in this model. The measurement model represents how the measurements are actually related to the values being measured and the confidence in the measurements. The initial conditions represent any prior knowledge about the system and the confidence in this information.

4.3.1 System model

The following equation is the system model for the Kalman filter [5].

\[
x_k = \Phi_{k-1} x_{k-1} + w_{k-1}
\]

The Kalman state variables, \( x \), are errors in the x and y position of the vehicle. This is what we are trying to estimate. \( \Phi \) is the system model matrix that relates the estimated error from the last step to the error in the current step. In this case, because the error does not grow with time, \( F \) is a 2x2 identity matrix. The process noise, \( w \), is assumed to be a normally distributed white noise with variance \( Q \), which we have taken to be \( 0 \).

\[
\begin{bmatrix}
    x_{error} \\
    y_{error}
\end{bmatrix}
\]

\[
\Phi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]

\[
w = \text{Process Noise} \sim N(0,Q)(nx1)
\]
4.3.2 Measurement model

The following equation is the measurement model for the Kalman filter [5].

\[ \mathbf{z}_k = H_k \mathbf{x}_k + \mathbf{v}_k \]  

\[ \mathbf{z}_k = \begin{bmatrix} \mathbf{x}_{\text{model}} - \mathbf{x}_{\text{GPS}} \\ \mathbf{y}_{\text{model}} - \mathbf{y}_{\text{GPS}} \\ \mathbf{x}_{\text{model}} - \mathbf{x}_{\text{encoder}} \\ \mathbf{y}_{\text{model}} - \mathbf{y}_{\text{encoder}} \end{bmatrix} \]

\[ H_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \]

\[ \mathbf{v} = \text{Measurement Error} \sim \mathcal{N}(0, \mathbf{R})(1 \times 1) \]

The \( \mathbf{z} \) matrix is the difference between the expected measurements (based on the dynamic model of the vehicle) and the actual GPS and wheel encoder measurements. \( H \) relates the measurements to the Kalman state variables. The measurement noise, \( \mathbf{v} \), is assumed to be a normally distributed white noise with variance \( \mathbf{R} \), which is made up of the variances in the GPS and wheel encoder measurements.

4.3.3 Initial Conditions

The following equations are the initial conditions for the Kalman filter [5].

\[ E[\mathbf{x}(0)] = \hat{\mathbf{x}}_0 \]  

\[ P_0 = E[(\mathbf{x}(0) - \hat{\mathbf{x}}_0)(\mathbf{x}(0) - \hat{\mathbf{x}}_0)^T] \]
Two initial conditions are necessary to begin the algorithm: the initial estimates of the Kalman state variables ($\hat{x}$), and the initial error covariance matrix ($P_0$).

### 4.3.4 State Estimation

The following equations are used to estimate the new state of the vehicle [5].

\[
\begin{align*}
\hat{x}_k^{}(-) &= \Phi_{k-1} \hat{x}_{k-1}^{}(+) \\

P_k^{}(-) &= \Phi_{k-1} P_{k-1}^{}(+) \Phi_{k-1}^T + Q_{k-1} \\

\hat{x}_k^{}(+) &= \hat{x}_k^{}(-) + K_k^{} [z_k - H_k^{} \hat{x}_k^{}(-)] \\

P_k^{}(+) &= [I - K_k^{} H_k^{}] P_k^{}(-) \\

K_k^{} &= P_k^{}(-) H_k^T [H_k^{} P_k^{}(-) H_k^T + R_k^{}]^{-1} \text{ (nxl)} \\

\text{Estimated Position} &= \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} x_{\text{model}} \\ y_{\text{model}} \end{bmatrix} - \hat{x}_k^{}(+) 
\end{align*}
\]

The position estimate found using the Kalman Filter takes into account the variances in both GPS and wheel encoder measurements, both past and present, as well as a model of the vehicle dynamics to produce an optimal estimate.
Chapter 5 Simulation

5.1 Overview

Simulation was performed to test the algorithm described in Chapter 2. This process was important both in choosing parameters for the algorithm, such as controller gains, and in debugging the algorithm. The simulation was performed using MATLAB [19], and the code is shown in Appendix C. A simulation of the Kalman Filter used to interpret sensor data is also given.

5.2 Control System Simulation

Several different paths were used to test the control algorithm, two of which are shown below.

![Figure 16: Several test paths were used to test the algorithm.](image-url)
The inputs to the simulation include the time interval between control signals, the maximum velocity of the vehicle, the maximum lateral and longitudinal vehicle accelerations, the vehicle mass, and the path and heading controller gains. For the values shown in Table 3 below, the Figures 17-19 show sample outputs of the simulation.

Figure 17 shows the vehicle position at equally spaced time intervals. Figure 18 shows the deviation from the path by the vehicle. Notice how the maximum path error occurs during the curved segment of the path. Figure 19 shows the vehicle speed as the vehicle follows the path. While in path segment 1, the vehicle accelerates to the maximum speed possible while still allowing deceleration to the segment 2 speed. The vehicle maintains a constant speed in segment 2, and accelerates again in segment 3 until it must decelerate in preparation for stopping at the end of the segment.

### Table 3: Inputs to Path Control System Simulation

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time between control signals</td>
<td>0.05 s</td>
</tr>
<tr>
<td>Max velocity</td>
<td>10 m/s</td>
</tr>
<tr>
<td>Max lateral acceleration</td>
<td>7.35 m/s²</td>
</tr>
<tr>
<td>Max longitudinal acceleration</td>
<td>7.35 m/s²</td>
</tr>
<tr>
<td>Vehicle mass</td>
<td>10 kg</td>
</tr>
<tr>
<td>Path Control Gain</td>
<td>2000</td>
</tr>
<tr>
<td>Heading Control Gain</td>
<td>100</td>
</tr>
</tbody>
</table>
Figure 17: The vehicle closely follows the desired path.

Figure 18: Error between the vehicle and the path is greatest in the middle of the turn.
Figure 19: The vehicle first accelerates, then decelerates in preparation for the turn, accelerates coming out of the turn, and then stops at the end of the path.

5.3 Simulation of Kalman Filter

The Kalman Filter was simulated by modifying the MATLAB simulation discussed above. Random noise was added to the actual position of the vehicle to simulate the sensor outputs. Due to the limited accuracy of the GPS receiver, the noise added to create this signal had a larger distribution than the noise added to the encoder signal. The two signals were combined using the Kalman Filter, and the output of the filter was used in the vehicle control equations.

The filter behaves as expected, providing a best-fit estimate of the vehicle location. However, as the noise added to the measurements was increased, the response of the vehicle became quite erratic. The following two figures show the effect of increased measurement noise
on path following. This is most likely due to the steering controller overcompensating for errors not in actual position but in estimated position. In the case of inaccurate measurement devices, control gains need to be lowered to avoid this problem.

Figure 20: With 2cm noise in the encoder measurements and 1m noise in the GPS measurements, the vehicle is able to follow the path well.
Figure 21: If the noise in the encoder measurements is increased to 20cm, the vehicle response is more erratic.
Chapter 6 Hardware Implementation

6.1 Overview

Full implementation of the control algorithm on a test vehicle was not completed. However, design of the vehicle was completed as well as the collection of all necessary components and writing of the control code. This chapter summarizes the design of the vehicle including the sensors used, the actual vehicle chosen for implementation, and the control electronics.

6.2 Sensors

Two different types of sensors were selected to perform the position of the vehicle. A GPS receiver was selected to give the location of the vehicle, and optical wheel sensors were designed to sense the vehicle’s movement relative to the ground.

The GPS receiver chosen was the Novatel OEM4-RT2 2-Channel Sequencing Receiver. It is accurate to within 0.02m and updates at a 10Hz rate. However, this receiver was not obtainable, so a Novatel Millenium RT20S was used in its place. The accuracy of this receiver is 1m and it has an update rate of 4 Hz.

To measure the rotation of the rear wheels, slotted wheels were mounted to the rear axle on each side of the differential. Two photomicrosensors (Omron EE-SX1042) were mounted to the vehicle to detect the rotation of each slotted wheel. The signal from the sensor is low (0V) if there is nothing blocking its two halves, and high (5V) when there is something in between.
Figure 22: Wheel rotation was measured using a slotted disk and a photomicrosensor.

6.3 Vehicle

Several different vehicles were considered for testing the control algorithms. One of the main decisions was whether to use a steered or a skid-steered vehicle. A steered vehicle was chosen due to its simpler kinematics and smaller power consumption [16] although the mobility of the vehicle would not be as good. A radio-controlled hobby car (Associated RC10T3 Sport Truck) was selected as the test vehicle and is shown in Figure 21. The vehicle is electrically powered by a 7.2 V battery. Steering is controlled by servo which uses a linkage connected to the front wheels. Speed is controlled through a servo which powers a motor connected to the rear wheels. The vehicle has a four-wheel independent suspension and a rear-wheel differential, which allows for different rotational speeds of the left and right rear wheels. Table 4 provides the vehicle specifications.
Table 4: Vehicle Specifications

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>15.8 inches</td>
</tr>
<tr>
<td>Width</td>
<td>12.6 inches</td>
</tr>
<tr>
<td>Wheelbase</td>
<td>11.4 inches</td>
</tr>
<tr>
<td>Track Length</td>
<td>10 inches</td>
</tr>
<tr>
<td>Height</td>
<td>5.5 inches</td>
</tr>
<tr>
<td>Weight</td>
<td>1.655 kg</td>
</tr>
<tr>
<td>Front Wheel Diameter</td>
<td>2.2 inches</td>
</tr>
<tr>
<td>Rear Wheel Diameter</td>
<td>2.2 inches</td>
</tr>
</tbody>
</table>

Figure 23: The Associated RC10T3 Sport Truck was used as the test vehicle.

6.4 Electronics

A single board computer (SBC) in conjunction with several microprocessors was used to collect and analyze sensor data and send control signals to the steering and speed actuators. The inputs to the SBC are the position of the vehicle (from the GPS receiver) and the number of rotations of the left and right wheels (from the wheel encoders). The outputs of the SBC are the speed of the vehicle (sent to the speed servo) and the turn angle of the vehicle (sent to the
steering servo). The signals sent by the speed and steering servos are both pulse-width modulated (PWM) signals that are centered at 1.5ms.

The software to control the vehicle was written in C and is listed in Appendix B.
Chapter 7 Results and Conclusions

7.1 Summary of Results

A control algorithm was developed that minimizes time spent by a ground vehicle on a specified path. The algorithm allows for travel along curvilinear pathways and the adjustment of vehicle response based on mission objectives. A simulation of the control system was developed which aids verification, testing, and modification of the algorithm. A control model was used to assist in the selection of control parameters and a vehicle kinematic model was created to allow implementation of the algorithm on a vehicle. Plans for the implementation in hardware were developed using a 1/10th scale radio-controlled car and a single board computer (SBC) which processes the sensor measurements and controls the vehicle actuators.

7.2 Future Work

There is a large opportunity for future work in this area. Improvements can be made in both the vehicle control system and its implementation.

A proportional-derivative controller was used to regulate the vehicle’s distance from the path and heading. If a more sophisticated controller was used, it is likely that the simulation results would be improved. A nonlinear controller would be useful due to the nonlinear nature of both the vehicle steering and the tire-ground interaction.

Improvements in the sensors would definitely improve the performance of the vehicle. The GPS receiver is not adequate to add much value to the sensor data. It has an update rate of 4Hz and is only accurate to within 1m. This leaves the vehicle position being determined by the wheel encoder data, which relies on a good vehicle model to provide accurate position...
information. The position update rate used in simulation was 10Hz. When the update rate is decreased to 4Hz, the vehicle response becomes much more erratic.

In order for this type of system to work well, a better vehicle model needs to be developed. The current vehicle model is kinematic only and does not consider the complex interactions between the tire and the ground. The only way to construct an accurate dynamic vehicle model would be to conduct tests to determine the relationship between tire slip angle and lateral force as well as slip ratio and longitudinal force. However, the type of surface the vehicle is traveling over heavily affects the dynamics of the vehicle, and if the surface is not known, this would be a problem. In the case of a manufacturing environment where AGVs are commonly used, the ideal solution would be to use an indoor location support system, such as Cricket [14] so that accurate dead-reckoning of position is not as critical.

7.3 Conclusions

Although experimental results were not obtained, a great deal of information was revealed by the results of the simulation and much was learned assembling the hardware.

There are clear tradeoffs between path following ability and speed of the vehicle. If speed along the path is optimized, the ability of the vehicle to accurately track the desired path will be adversely affected.

There is also a tradeoff between heading and path control of the vehicle. If the vehicle needs to stay as close to the path as possible, it will often be correcting its course more frequently, which leads to poor heading control. It also affects the speed with which it is able to navigate the path. This tradeoff needs to be addressed when determining the objectives of the path-following vehicle.
The quality of the position sensing information is critically important to the path following algorithm. If there is a large variance in the position data, high feedback control systems cause the vehicle to act in an unstable manner. It is very important to use position sensors that provide information that is as accurate as possible. It is also beneficial to determine position using multiple sensors and methods and to combine this information using proper statistical estimation techniques.
Bibliography


Appendix A: Acronym Glossary

AGV – Automated Guided Vehicle

ALV – Autonomous Land Vehicle

DARPA – Defense Advanced Research Projects Agency

GPS – Global Positioning System

NAHSC – National Automated Highway System Consortium

RF – Radio Frequency

SBC – Single Board Computer

TMR – Tactical Mobile Robotics

UGV – Unmanned Ground Vehicle
Appendix B: Software Flow

Step 1: Create array describing N path segments
- Length (segment(n,1))
- Curvature (segment(n,2))
- Orientation (segment(n,3))

Step 2: Convert array into a set of discrete points on the path
- Determine total length of the path

\[
\text{total\_length\_of\_path} = 0; \\
\text{for } k = 1:N, \\
\quad \text{total\_length\_of\_path} = \text{total\_length\_of\_path} + \text{segment}(1,k); \\
\text{end}
\]

- Decide spacing between the points on the path (dels)
- At each point, determine x and y coordinates of path points, curvature at that point, angle of the path at that point, and within which segment that point resides.

\[
s=0; \\
i=1; \\
c = 2; \\
\text{xpath}(1) = 0; \\
\text{ypath}(1) = 0; \\
\text{pathangle}(1) = \text{segment}(3,1); \\
\text{while } s < \text{total\_length\_of\_path}, \\
\quad s = s + \text{dels}; \\
\quad \text{pathangle}(c) = \text{pathangle}(c-1) + \text{dels*segment}(2,i); \\
\quad \text{if } \text{pathangle}(c) > 2\pi \\
\quad \quad \quad \text{pathangle}(c) = \text{pathangle}(c) - 2\pi; \\
\quad \text{end} \\
\quad \text{curvature}(c) = \text{segment}(2,i); \\
\quad \text{pathsegment}(c) = i; \\
\quad \text{xpath}(c) = \text{xpath}(c-1) + \text{dels*cos(pathangle}(c)); \\
\quad \text{ypath}(c) = \text{ypath}(c-1) + \text{dels*sin(pathangle}(c)); \\
\quad \text{end} \\
\text{check which road segment the vehicle is in} \\
\text{length} = 0; \\
\text{for } j = 1:N, \\
\quad \text{length} = \text{length} + \text{segment}(1,j); \\
\quad \text{if } s < \text{length}, \\
\quad \quad \quad i = j; \\
\quad \quad \quad \text{break} \\
\quad \text{end} \\
\text{end} \\
\text{c} = c + 1; \\
\text{end}
Step 3: Set constants and initial conditions

- Set the vehicle properties
  - Maximum vehicle speed (vmax)
  - Maximum lateral acceleration (amaxlat)
  - Maximum longitudinal acceleration (amaxlon)
  - Mass
- Set the control gains
  - Proportional constant for distance from path (Kpath)
  - Proportional constant for heading (Kheading)
- Set the Kalman filter constants
- Set measurement variance matrix for GPS and encoder (rgps, rencoder)

\[
R = [\begin{bmatrix} rgps & 0 & 0 & 0 \\ 0 & rgps & 0 & 0 \\ 0 & 0 & rencoder & 0 \\ 0 & 0 & 0 & rencoder \end{bmatrix}];
\]

- Set process noise variance

\[
Q = [\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}];
\]

- Set system model

\[
Phi = [\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}];
\]

- Set measurement model

\[
H = [\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}];
\]

- Set initial conditions

\[
x_{\text{model}}(2) = 0;
y_{\text{model}}(2) = 0;
x_{\text{plus}}(:,2) = [0; 0];
y_{\text{plus}}(:,2) = [0.1; 0; 0.1];
\]

- Set the initial vehicle conditions

  - X-position (xact) – first 2
  - Y-position (yact) – first 2
  - Velocity – first 3
  - Heading (vehicleheading) – first 2
- Set the other initial conditions

  - Time interval between control signals (dt)
  - Path heading
  - Path error (error) – first 2
  - Total distance travel this far (distancetraveled)
- Counters

\[
i = 1;
s = 0;
k = 3;
j = 3;
\]
WHILE THE VEHICLE IS RUNNING...

Step 4: Determine which point on the path is closest to the vehicle
- Determine the distance between the vehicle and the point on the path that was closest to the last vehicle position.
  \[
  \text{error}(k) = \sqrt{(x_{path}(j) - x_{act}(k-1))^2 + (y_{path}(j) - y_{act}(k-1))^2};
  \]
- Determine the distance between the vehicle and the next point on the path
  \[
  \text{errornext}(k) = \sqrt{(x_{path}(j+1) - x_{act}(k-1))^2 + (y_{path}(j+1) - y_{act}(k-1))^2};
  \]
- If the next point is closer to the vehicle than the current point, check the next point...

\[
\begin{align*}
\text{while } & \text{error}(k) > \text{errornext}(k), \\
j &= j + 1; \\
\text{error}(k) &= \sqrt{(x_{path}(j) - x_{act}(k-1))^2 + (y_{path}(j) - y_{act}(k-1))^2}; \\
\text{errornext}(k) &= \sqrt{(x_{path}(j+1) - x_{act}(k-1))^2 + (y_{path}(j+1) - y_{act}(k-1))^2}; \\
\text{distancetraveled} &= \text{distancetraveled} + \text{dels}; \quad \% \text{This is used to determine how far along the current path segment the vehicle has traveled} \\
\text{if } & \text{pathsegment}(j-1) != \text{pathsegment}(j) \quad \% \text{If the path segment has changed (increased)} \\
& \text{distancetraveled} = 0; \quad \% \text{The counter resets.} \\
\text{end}
\end{align*}
\]

Step 5: Determine the distance between the vehicle and the path (This is NOT the distance between the vehicle and the closest path point).
- Compute the distance between the vehicle and the point on the path before the closest point on the path.
  \[
  \text{errorlast}(k) = \sqrt{(x_{path}(j-1) - x_{act}(k-1))^2 + (y_{path}(j-1) - y_{act}(k-1))^2};
  \]
- If the point after the closest point is closer than the point before the current point (if \(\text{errornext}(k) < \text{errorlast}(k)\)),
  \[
  \text{error}(k) = \sqrt{\text{error}(k)^2 - (\text{dels}^4 + 2*\text{dels}^2*\text{error}(k)^2 + 2*\text{dels}^2*\text{errornext}(k)^2 + \text{error}(k)^4 - 2*\text{error}(k)^2*\text{errornext}(k)^2 + \text{errornext}(k)^4)/(4*\text{dels}^2)};
  \]
- Else,
\[
\text{error}(k) = \sqrt{\text{error}_\text{last}(k)^2 - (\text{dels}^4 + 2\text{dels}^2 \text{error}_\text{last}(k)^2 - 2\text{dels}^2 \text{error}(k)^2 + \text{error}_\text{last}(k)^4 - 2\text{error}_\text{last}(k)^2 \text{error}(k)^2 + \text{error}(k)^4)/(4\text{dels}^2))}
\]

**Step 6:** Determine whether the vehicle needs to accelerate, decelerate, or keep its speed constant based on its current speed and its location on the path.

- Look at the current path length (\text{patharray}(1,*)), current path curvature (\text{patharray}(2,*)), current path orientation (\text{patharray}(3,*)), and calculate the current path maximum speed (\text{patharray}(4,*))

\[
\begin{align*}
\text{patharray}(1,1) &= \text{segment}(1,\text{pathsegment}(j)).\text{value}; \\
\text{patharray}(2,1) &= \text{segment}(2,\text{pathsegment}(j)).\text{value}; \\
\text{patharray}(3,1) &= \text{segment}(3,\text{pathsegment}(j)).\text{value}; \\
\text{if } \text{patharray}(2,1) &> \text{amaxlat}/(v_{\text{max}}v_{\text{max}}) \\
&\quad \text{patharray}(4,1) = \sqrt{\text{amaxlat}/\text{patharray}(2,1)}; \\
\text{else} \\
&\quad \text{patharray}(4,1) = v_{\text{max}};
\end{align*}
\]

- If this is the last segment, use the following values for the next path length, curvature, orientation, and maximum speed

\[
[m,n] = \text{size}(...)
\]

\[
\begin{align*}
\text{if } \text{pathsegment}(j) &= n \\
\text{patharray}(1,2) &= 0; \\
\text{patharray}(2,2) &= 0; \\
\text{patharray}(3,2) &= 0; \\
\text{patharray}(4,2) &= 0; \\
\text{else} \\
\text{patharray}(1,2) &= \text{segment}(1,\text{pathsegment}(j)+1).\text{value}; \\
\text{patharray}(2,2) &= \text{segment}(2,\text{pathsegment}(j)+1).\text{value}; \\
\text{patharray}(3,2) &= \text{segment}(3,\text{pathsegment}(j)+1).\text{value}; \\
\text{if } \text{patharray}(2,2) &> \text{amaxlat}/(v_{\text{max}}v_{\text{max}}) \\
&\quad \text{patharray}(4,2) = \sqrt{\text{amaxlat}/\text{patharray}(2,2)}; \\
\text{else} \\
&\quad \text{patharray}(4,2) = v_{\text{max}}; \\
\end{align*}
\]

- Else, look at the next path length, curvature, orientation, and calculate the next path maximum speed.

**Step 7:** Determine which velocity profile is applicable to the given situation

- Determine maximum speed of current and next path segments

\[
\begin{align*}
\text{max\_velocity\_current} &= \text{patharray}(4,1); \\
\text{velocity\_next} &= \text{patharray}(4,2);
\end{align*}
\]

- Determine distance needed to accelerate from max speed on current segment to max speed on next segment

\[
d\text{decel} = 0.5\times(\text{max\_velocity\_current}^2 - \text{velocity\_next}^2)/\text{amaxlon};
\]
• Determine distance needed to accelerate from current speed to max speed on current segment
  \[ \text{daccel} = 0.5 \times (\text{max}\_\text{velocity}\_\text{current}^2 - \text{velocity(k)}^2) / \text{amaxlon}; \]

• If the total distance exceeds the distance left in the current path segment, Case 1 exists
  \[ \text{if} \quad \text{daccel} + \text{ddecel} \geq \text{patharray}(1,1) - \text{distancetraveled}, \]
  Case 1; Else, Case 2 exists

Step 8: If case 1 exists,

if Case==1
  v0 = velocity(k);
  v2 = patharray(4,2);
  d2 = patharray(1,1) - distancetraveled;
  v1 = \sqrt{(\text{amaxlon} \times d2 + 0.5 \times v0^2 + 0.5 \times v2^2)};
  t1 = (v1 - v0) / \text{amaxlon};
  d1 = (v1 + v0) \times t1 / 2;
  t2 = (v1 - v2) / \text{amaxlon} + t1;
  if t1>0
    if isreal(t1) == 1
      velocitychange = 1;
    else
      velocitychange = -1;
    end
  else
    velocitychange = -1;
  end
end

Step 9: If case 2 exists,

if Case==2
  v0 = velocity(k);
  v1 = patharray(4,1);
  v2 = patharray(4,1);
  v3 = patharray(4,2);
  d3 = patharray(1,1) - distancetraveled;
  t1 = (v1 - v0) / \text{amaxlon};
  d1 = 0.5 \times (v1^2 - v0^2) / \text{amaxlon};
  d2 = d3 - 0.5 \times (v1^2 - v3^2) / \text{amaxlon};
  t2 = (d2 - d1) / v1 + t1;
  t3 = t2 + (v1-v3) / \text{amaxlon};
  if t1>0
    velocitychange = 1;
end
elseif \( t2 > 0 \)
    velocitychange = 0;
else
    velocitychange = -1;
end

Step 10: Update the vehicle speed and distance traveled based on the required change in velocity

\[
\text{velocity}(k+1) = \text{velocity}(k) + \text{amaxlon} \times \text{velocitychange} \times \text{dt};
\]
\[
\text{ds} = \text{velocity}(k+1) \times \text{dt};
\]

Step 11: Determine whether the vehicle is on the left or right side of the path
- Find the vector which gives the vehicle heading

\[
\text{vehiclevector}(1,1) = \cos(\text{vehicleheading}(k-1));
\]
\[
\text{vehiclevector}(1,2) = \sin(\text{vehicleheading}(k-1));
\]
\[
\text{vehiclevector}(1,3) = 0;
\]

- Find the vector starting at the vehicle and ending at the nearest path point

\[
\text{pathvector}(1,1) = \text{xpath}(j) - \text{xact}(k-1);
\]
\[
\text{pathvector}(1,2) = \text{ypath}(j) - \text{yact}(k-1);
\]
\[
\text{pathvector}(1,3) = 0;
\]

- Take the cross-product of the vectors. The sign of the cross product determines the direction of the steering force

\[
\text{c} = \text{cross(pathvector, vehiclevector)};
\]
\[
\text{forcesign} = \text{sign}(\text{c}(1,3));
\]

Step 12: Determine the control effort needed to steer the vehicle closer to the path

\[
\text{F}_{b-y-path}(k) = -1 \times \text{forcesign} \times \text{Kpath} \times \text{error}(k);
\]

Step 13: Determine the control effort needed to steer the vehicle to make its heading parallel to the path
- Determine the heading of the path

\[
\text{pathheading}(k) = \text{atan2}((\text{ypath}(j) - \text{ypath}(j-1)), (\text{xpath}(j) - \text{xpath}(j-1)));
\]
\[
\text{if} \ \text{pathheading}(k) < 0
\]
\[
\text{pathheading}(k) = 2 \times \pi + \text{pathheading}(k);
\]
end

- Determine the direction that the vehicle needs to steer to correct its heading
  - Create a vector describing the vehicle heading

\[
\text{vehiclevector}(1,1) = \text{xact}(k-1) - \text{xact}(k-2);
\]
\[
\text{vehiclevector}(1,2) = \text{yact}(k-1) - \text{yact}(k-2);
\]
\[
\text{vehiclevector}(1,3) = 0;
\]

- Create a vector describing the path heading
pathvector(1,1) = xpath(j) - xpath(j-1);
pathvector(1,2) = ypath(j) - ypath(j-1);
pathvector(1,3) = 0;

- Find the cross product of the vectors. The sign determines the direction of steering effort

\[
\text{crossproduct} = \text{cross} (\text{vehiclevector}, \text{pathvector});
\text{headingforcesign}(k) = \text{sign} (\text{crossproduct}(1,3));
\]

- Determine the control effort

\[
\text{headingerror}(k) = \text{abs}(\text{pathheading}(k) - \text{vehicleheading}(k-1));
\]

\[
\text{if } \text{headingerror}(k) > \pi, \\
\quad \text{headingerror}(k) = 2\pi - \text{headingerror}(k);
\text{end}
\]

\[
F_{b\_b\_y\_heading}(k) = \text{headingforcesign}(k)*K_{heading}*\text{headingerror}(k);
\]

---

**Step 14: Determine the nominal steering effort**
- Determine the current path curvature

\[
\text{pathcurvature}(k) = \text{curvature}(j);
\]

- Use this to determine the necessary steering effort

\[
F_{b\_b\_y\_nominal}(k) = \text{mass}^*\text{velocity}(k+1)^2*\text{velocity}(k+1)^2*\text{pathcurvature}(k);
\]

---

**Step 15: Determine the total steering effort**

\[
F_{b\_b\_y} = F_{b\_b\_y\_path}(k) + F_{b\_b\_y\_heading}(k) + F_{b\_b\_y\_nominal}(k);
\]

---

**Step 16: Filter Measurement data (xgps, ygps, xencoder, yencoder)**
- Use vehicle model to estimate position

\[
d\theta = F_{b\_b\_y}^*ds/(\text{mass}^*\text{velocity}(k+1)^2); \\
\text{vehicleheading}(k) = \text{vehicleheading}(k-1) + d\theta;
\]

\[
\text{if } \text{vehicleheading}(k) < 0 \\
\quad \text{vehicleheading}(k) = 2\pi + \text{vehicleheading}(k);
\text{end}
\]

\[
dx = ds^*\cos(\text{vehicleheading}(k));
\text{dy} = ds^*\sin(\text{vehicleheading}(k));
\]

\[
\text{xmodel}(k) = \text{xmodel}(k-1) + dx;
\text{ymodel}(k) = \text{ymodel}(k-1) + \text{dy};
\]

- Find best estimate of vehicle position (xact)
\[ z(1,k) = x_{\text{model}}(k) - x_{\text{gps}}(k); \]
\[ z(2,k) = y_{\text{model}}(k) - y_{\text{gps}}(k); \]
\[ z(3,k) = x_{\text{model}}(k) - x_{\text{encoder}}(k); \]
\[ z(4,k) = y_{\text{model}}(k) - y_{\text{encoder}}(k); \]
\[
x_{\text{minus}}(:,k) = \Phi x_{\text{plus}}(:,k-1); \]
\[
P_{\text{minus}}(:,:,k) = \Phi P_{\text{plus}}(:,:,k-1)*\Phi' + Q; \]
\[
K(:, :, k) = P_{\text{minus}}(:, :, k)*H'*(H*P_{\text{minus}}(:, :, k)*H' + R)^{-1}; \text{ % Kalman gain matrix} \]
\[
P_{\text{plus}}(:,:,k) = (1-K(:, :, k)*H)*P_{\text{minus}}(:, :, k); \]
\[
x_{\text{plus}}(:, k) = x_{\text{minus}}(:, k) + K(:, :, k)*(z(:, k) - H*x_{\text{minus}}(:, k)); \]
\[
x_{\text{act}}(k) = x_{\text{model}}(k) - x_{\text{plus}}(1, k); \]
\[
x_{\text{act}}(k) = y_{\text{model}}(k) - x_{\text{plus}}(2, k); \]

**Step 17: Update counters**

\[ s = s + ds; \]
\[ k = k + 1; \]
Appendix C: Complete Matlab Simulation Code

clear;

% Segment Arrays

N = 3; % Total Number of Segments

% Segment 1
segment(1,1) = 10;
segment(2,1) = 0;
segment(3,1) = 0;

% Segment 2
segment(1,2) = 2*pi*2/4;
segment(2,2) = 0.5;
segment(3,2) = 0;

% Segment 3
segment(1,3) = 10;
segment(2,3) = 0;
segment(3,3) = pi/2;

% Divide up the given path into discrete points

total_length_of_path = 0;
for k = 1:N,
    total_length_of_path = total_length_of_path + segment(1,k);
end

s=0;

pathangle(1) = segment(3,1);
i=1;
c = 2;

xpath(1) = 0;
ypath(1) = 0;

dels = 0.05; % Spacing between points on the path

while s < total_length_of_path,
    s = s + dels;

    pathangle(c) = pathangle(c-1) + dels*segment(2,i); % Calculate the path angle
    if pathangle(c) > 2*pi
        pathangle(c) = pathangle(c) - 2*pi;
    end

    curvature(c) = segment(2,i); % Check the road curvature

    pathsegment(c) = i;
\[
\text{xpath}(c) = \text{xpath}(c-1) + \text{dels} \cdot \cos(\text{pathangle}(c));
\]
\[
\text{ypath}(c) = \text{ypath}(c-1) + \text{dels} \cdot \sin(\text{pathangle}(c));
\]

%Check which road segment the vehicle is in

length = 0;
for \( j = 1:N \),
  length = length + segment(1, j);
  if \( s < \text{length} \),
    i = j;
    break
  end
end

c = c + 1;
end

%Constants
\[
\text{dt} = 0.1; \quad \text{\#Time interval between control signals}
\]
\[
\text{vmax} = 8; \quad \text{\#Maximum vehicle velocity on straight segments}
\]
\[
\text{amaxlat} = 7.35; \quad \text{\#Maximum lateral vehicle acceleration (3/4 g)}
\]
\[
\text{amaxlon} = 7.35; \quad \text{\#Maximum longitudinal vehicle acceleration (3/4 g)}
\]
\[
\text{Kpath} = 100; \quad \text{\#Proportional control constant for distance from path}
\]
\[
\text{Kheading} = 10; \quad \text{\#Proportional control constant for heading}
\]
\[
\text{mass} = 1.75; \quad \text{\#Vehicle mass in kg}
\]

%Initial Vehicle Conditions
\[
\text{xactlast} = 0;
\]
\[
\text{yactlast} = 0;
\]
\[
\text{vehicleheading} = 0;
\]
\[
\text{xactlastlast} = 0;
\]
\[
\text{yactlastlast} = 0;
\]

\[
\text{xact} = 0;
\]
\[
\text{yact} = 0;
\]

\[
\text{pathheading} = 0;
\]
\[
\text{velocity} = 0;
\]
\[
\text{error}(2) = 0;
\]
\[
i = 1;
\]
\[
\text{distancetraveled} = 0;
\]
\[
s = 0;
\]
\[
k = 3;
\]
\[
j = 2;
\]

while \( 2-=3 \), \text{%Big while loop that does everything.}

%Check which point on the path is closest to the vehicle
\[
\text{error} = \sqrt{(\text{xpath}(j) - \text{xactlast})^2 + (\text{ypath}(j) - \text{yactlast})^2};
\]
\[
\text{errornext} = \sqrt{(\text{xpath}(j+1) - \text{xactlast})^2 + (\text{ypath}(j+1) - \text{yactlast})^2};
\]
while error > errornext,
  j = j + 1;

  error = sqrt((xpath(j) - xactlast)^2 + (ypath(j) - yactlast)^2);
  errornext = sqrt((xpath(j+1) - xactlast)^2 + (ypath(j+1) - yactlast)^2);

  distancetraveled = distancetraveled + dels; %This is used to determine how far along the current path segment the vehicle has traveled
    if pathsegment(j-1) ~= pathsegment(j) %If the path segment has changed (increased)
      distancetraveled = 0;
    end %The counter resets.
end

errorlast = sqrt((xpath(j-1) - xactlast)^2 + (ypath(j-1) - yactlast)^2);

%This computes the distance between the vehicle and the path (as opposed to the distance between the vehicle and the closest point on the path)
  if errornext < errorlast,
    error = sqrt(abs(error^2 - (dels^4+2*dels^2*error^2-2*dels^2*errornext^2+errornext^4)/(4*dels^2)));
    errorout(k) = error;
  else
    error = sqrt(abs(errorlast^2 - (dels^4+2*dels^2*errorlast^2-2*dels^2*error^2+errorlast^4-2*errorlast^2*error^2 + error^4)/(4*dels^2)));
    errorout(k) = error;
  end

%This function determines if the vehicle needs to accelerate, decelerate, or keep its speed constant based on the next two segments in the path. This summarizes the important information about the next two segments in the array "patharray". If the current segment is the last segment (no next segment) the vehicle must slow down and stop

  patharray(1,1) = segment(1,pathsegment(j)); %current path segment length
  patharray(2,1) = segment(2,pathsegment(j)); %current path segment curvature
  patharray(3,1) = segment(3,pathsegment(j)); %current path segment orientation
    if patharray(2,1) > amaxlat/(vmax*vmax)
      patharray(4,1) = sqrt(amaxlat/patharray(2,1)); %current path segment maximum speed
    else
      patharray(4,1) = vmax;
    end

[m,n] = size(segment);
  if pathsegment(j) == n %next path segment length
    patharray(1,2) = 0;
  else %next path segment curvature
    patharray(2,2) = 0;
  end %next path segment orientation
    if pathsegment(j) == n %next path segment maximum speed
      patharray(3,2) = 0;
    else
      patharray(3,2) = 0;
    end
    if pathsegment(j) == n
      patharray(4,2) = 0;
    else
      patharray(4,2) = 0;
    end

75
else
    patharray(1,2) = segment(1,pathsegment(j)+1);  %path segment length
    patharray(2,2) = segment(2,pathsegment(j)+1);  %path segment curvature
    patharray(3,2) = segment(3,pathsegment(j)+1);  %path segment orientation
    if patharray(2,2) > amaxlat/(vmax*vmax)
        patharray(4,2) = sqrt(amaxlat/patharray(2,2));  %path segment maximum speed
    else
        patharray(4,2) = vmax;
    end
end

%Check which velocity profile is applicable to the given situation. This
function determines which of three velocity profile cases exist based on
current speed, max speed of the current segment, max speed of the next
segment, acceleration possible and length of current path segment.

max_velocity_current = patharray(4,1);
velocity_next = patharray(4,2);

ddecel = 0.5*(max_velocity_current^2 - velocity_next^2) / amaxlon;
daccel = 0.5*(max_velocity_current^2 - velocity^2) / amaxlon;

if daccel + ddecel >= patharray(1,1)-distancetraveled,
    Case = 1;
else
    Case = 2;
end

if Case==1
    v0 = velocity;
    v2 = patharray(4,2);
    d2 = patharray(1,1) - distancetraveled;
    v1 = sqrt(amaxlon*d2 + 0.5*v0^2 + 0.5*v2^2);
    t1 = (v1 - v0)/amaxlon;
    d1 = (v1 + v0)*t1 / 2;
    t2 = (v1 - v2)/amaxlon + t1;
    if t1>0
        % if isreal(t1) == 1
        velocitychange = 1;
    % else
    %    velocitychange = -1;
    % end
    else
        velocitychange = -1;
    end
end

if Case==2
    v0 = velocity;
    v1 = patharray(4,1);
    v2 = patharray(4,1);
    v3 = patharray(4,2);
\[ d_3 = \text{patharray}(1,1) - \text{distancetraveled}; \]
\[ t_1 = (v_1 - v_0)/a_{maxlon}; \]
\[ d_1 = 0.5*(v_1^2-v_0^2)/a_{maxlon}; \]
\[ d_2 = d_3 - 0.5*(v_1^2 - v_3^2)/a_{maxlon}; \]
\[ t_2 = (d_2 - d_1)/v_1 + t_1; \]
\[ t_3 = t_2 + (v_1-v_3)/a_{maxlon}; \]
\[ \text{if } t_1 > 0 \]
\[ \quad \text{velocitychange} = 1; \]
\[ \text{elseif } t_2 > 0 \]
\[ \quad \text{velocitychange} = 0; \]
\[ \text{else} \]
\[ \quad \text{velocitychange} = -1; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{velocity} = \text{velocity} + a_{maxlon}\text{velocitychange}\text{dt}; \]
\[ ds = \text{velocity}\text{dt}; \]

%This determines if a path point is on the right or the left of a vehicle given the location of a vehicle and its heading. The program returns +1 if the vehicle is to the left of the path, 0 if the vehicle is on the path, and -1 if the vehicle is to the right of the path.

\[ \text{vehiclevector}(1,1) = \cos(\text{vehicleheading}); \]
\[ \text{vehiclevector}(1,2) = \sin(\text{vehicleheading}); \]
\[ \text{pathvector}(1,1) = \text{xpath}(j) - \text{xactlast}; \]
\[ \text{pathvector}(1,2) = \text{ypath}(j) - \text{yactlast}; \]
\[ \text{c} = (\text{vehiclevector}(1,1) \times \text{vehiclevector}(1,2)) - (\text{vehiclevector}(1,1) \times \text{pathvector}(1,2)); \]
\[ \text{forcesign} = \text{sign(c)}; \]

%Path Control Effort
\[ F_{b_{-}b_{-}y\_\text{path}} = -1*\text{forcesign}*\text{Kpath}\_\text{error}; \]

%Heading Error Calculations
\[ \text{pathheading} = \text{atan2}((\text{ypath}(j) - \text{ypath}(j-1)),(\text{xpath}(j) - \text{xpath}(j-1))); \]
\[ \text{if } \text{pathheading} < 0 \]
\[ \quad \text{pathheading} = 2*\pi + \text{pathheading}; \]
\[ \text{end} \]
\[ \text{vehiclevector}(1,1) = \text{xactlast} - \text{xactlastlast}; \]
\[ \text{vehiclevector}(1,2) = \text{yactlast} - \text{yactlastlast}; \]
\[ \text{pathvector}(1,1) = \text{xpath}(j) - \text{xpath}(j-1); \]
\[ \text{pathvector}(1,2) = \text{ypath}(j) - \text{ypath}(j-1); \]
\[ \%\text{crossproduct} = \text{cross(vehiclevector, pathvector)}; \]
\[ \text{crossproduct} = (\text{vehiclevector}(1,1)\times\text{pathvector}(1,2)) - (\text{vehiclevector}(1,1)\times\text{vehiclevector}(1,2)); \]
\[ \text{headingforcesign} = \text{sign(crossproduct)}; \]
\[ \text{headingerror} = \text{abs(pathheading} - \text{vehicleheading}); \]
%This compensates for errors due to transition of heading from slightly less than 360 to slightly more than 0.
if headingerror > pi,
    headingerror = 2*pi - headingerror;
end

%Heading Control Effort
F_b_b_y_heading = headingforcesign*Kheading*headingerror;

%Nominal Steering Effort
pathcurvature = curvature(j);
F_b_b_y_nominal = mass*velocity*velocity*pathcurvature;

%Total Steering Effort
F_b_b_y = F_b_b_y_path + F_b_b_y_heading + F_b_b_y_nominal;

%Vehicle Model
dtheta = F_b_b_y*ds/(mass*(velocity)^2);
vehicleheading = vehicleheading + dtheta;
dx = ds*cos(vehicleheading);
dy = ds*sin(vehicleheading);

xact = xact + dx;
yact = yact + dy;

xactlastlast = xactlast;
yactlastlast = yactlast;

xactlast = xact;
yactlast = yact;

s = s + ds;
k = k + 1;

totalsec(k-1) = s;

plot(xact,yact,'s',xpath,ypath);
axis([-l 14 -2 14])
drawnnow

end
Appendix D: Complete Vehicle Control C Code

// vehiclemain.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define VMAX 8.0 /* Maximum vehicle velocity on straight segments */
#define AMAXLAT 7.35 /* Maximum lateral vehicle acceleration (3/4 g) */
#define AMAXLON 7.35 /* Maximum longitudinal vehicle acceleration (3/4 g) */
#define KPATH 100.0 /* Proportional control constant for distance from path */
#define KHEADING 10.0 /* Proportional control constant for heading */
#define MASS 1.655 /* Vehicle mass in kg */
#define PI 3.14159
#define PATHSIZE 500 /* Size of arrays in path variable */
#define WHEELBASE 0.2794 /* Vehicle wheelbase in m */
#define TRACKLENGTH 0.254 /* Vehicle track length in m */
#define WHEELDIA 0.05588 /* Vehicle tire diameter in m */

int velocityschange(double velocity, double patharr[3][3], int pathseg[500], double disttrav, int j, int N);
double angle(double rho, double hfsign, double eh, double pfsign, double ep, double v);
void speed_out(double v);
void steering_out(double ang);
int left_encoder(void);
int right_encoder(void);

int main (int argc, char* argv[ ])
{

    FILE *fhandle;
double test = 3.2034;
double temp;

    int N=3; /* Number of segments in the path */
double patharr[3][3]; /* Array containing information about the path */
double totlen=0.0; /* Total length of the path */
int k; /* Counter */

double s=0.0; /* Path length */
double xpath[PATHSIZE]; /* X locations of points on the path */
double ypath[PATHSIZE]; /* Y locations of points on the path */
double pathang[PATHSIZE]; /* Angle of the path wrt x axis at each point on the path */
double curve[PATHSIZE]; /* Path curvature at each point on the path */
int pathseg[PATHSIZE];
double dels=0.05; /* Spacing between points on the path */
double length=0.0; /* Length of certain number of path segments */
int i=0; /* Counter */
int c=1; /* Counter */
int j; /* Counter */

double velocity; /* Vehicle velocity */
double xact; /* X position of vehicle */

79
double yact;    /* Y position of vehicle */
double xactlast;    /* Previous x position of vehicle */
double yactlast;    /* Previous y position of vehicle */
double xactlastlast;    /* X position of vehicle previous to previous position */
double yactlastlast;    /* Y position of vehicle previous to previous position */
double vheading;    /* Vehicle heading */
double pheading;    /* Path heading */
double disttrav;    /* Distance traveled */
double error;    /* Distance between vehicle and closest path point */
double errornext;    /* Distance between vehicle and next path point */
double errorlast;    /* Distance between vehicle and last path point */
int velchnge;    /* Velocity change sign */
double vvector[3];    /* Vector showing vehicle’s heading */
double pvector[3];    /* Vector showing path’s heading */
double pfsign;    /* Sign of path control force */
double hfsign;    /* Sign of heading control force */
double heading_error;    /* Heading error */

// double Fbbypath;    /* Path control force in the y direction */
// double Fbbyhead;    /* Heading control force in the y direction */
// double Fbbynorn;    /* Nominal control force in the y direction */
// double Fbby;    /* Total control force in the y direction */

double total_angle;    /* Desired steering angle */
double left_wheel_distance;    /* Distance the left wheel has traveled since last checked */
double right_wheel_distance;    /* Distance the right wheel has traveled since last checked */

double crossproduct;

double rho;    /* Curvature of vehicle’s path */
double dx;    /* Change in x position */
double dy;    /* Change in y position */
double dtheta;    /* Change in theta */
double ds;    /* Change in position */
double dt=0.05;    /* Time interval between control signals */

/****PATH INFORMATION****************************/
for(k=0; k<=(N-1); k++) {
    totlen = totlen + patharr[0][k];
}

/* Divide up the given path into discrete points */
pathang[0] = patharr[2][0];
xpath[0] = 0;
ypath[0] = 0;

while(s < totlen) {
    s = s + dels;

    pathang[c] = pathang[c-1] + dels*patharr[1][i]; /* Calculate the path angle */
    if (pathang[c] > 2*PI) {
        pathang[c] = pathang[c] - 2*PI;
    }

    curve[c] = patharr[1][i]; /* Record the road curvature at this path point*/
    pathseg[c] = i; /* Record the path segment at this path point */

    xpath[c] = xpath[c-1] + dels*cos(pathang[c]);
ypath[c] = ypath[c-1] + dels*sin(pathang[c]);

    /* Check which road segment the vehicle is in */
    length = 0;
    for(j=0; j<=(N-1); j++) {
        length = length + patharr[0][j];
        if (s < length) {
            i = j;
            break;
        }
    }
    c = c + 1;
}

/***END PATH INFORMATION***************************************************************************/

/* Initial Vehicle Conditions */
velocity = 0;
xact = 0;
yact = 0;
vheading = 0;
vheading = 0;
xactlast = 0;
yactlast = 0;
xactlastlast = 0;
yactlastlast = 0;
phheading = 0;
rho = 0;

time = 0;

i = 0;


disttrav = 0;
s = 0;
k = 2;
j = 1;

while(j<=PATHSIZE-50) {
    /* Check which point on the path is closest to the vehicle */
    error = sqrt(temp);

    while (error > errornext) {
        j++;
        error = sqrt(temp);

        disttrav = disttrav + dels;  /* This is used to determine how far along the current path segment the
vehicle has traveled */
        if (patharr[j-1] != patharr[j]) {  /* If the path segment has changed (increased) */
            disttrav = 0;  /* The counter resets. */
        }
    }


    /* This computes the distance between the vehicle and the path (as opposed to the distance between
the vehicle and the closest point on the path) */
    if (errornext < errorlast) {
        temp = error*error - (pow(dels,4)+2*dels*dels*error*error-2*dels*dels*errornext*errornext+pow(error,4)-2*error*errornext*errornext + pow(errornext,4))/(4*dels*dels);
        error = sqrt(fabs(temp));
    } else {
        temp = errorlast*errorlast - (pow(dels,4)+2*dels*dels*errorlast*errorlast-2*dels*dels*error*error+pow(errorlast,4)-2*errorlast*errorlast*error*error + pow(error,4))/(4*dels*dels);
        error = sqrt(fabs(temp));
    }

    velchnge = velocity_change(velocity, patharr, pathseg, disttrav, j, N);  /* Determines whether vehicle
needs to accelerate or decelerate */

    velocity = velocity + AMAXLON*velchnge*dt;  /* Determine the desired velocity */
    speed_out(velocity);  /* Send the speed signal to the speed controller */
}

velchnge = velocity_change(velocity, patharr, pathseg, disttrav, j, N);  /* Determines whether vehicle
needs to accelerate or decelerate */

velocity = velocity + AMAXLON*velchnge*dt;  /* Determine the desired velocity */
speed_out(velocity);  /* Send the speed signal to the speed controller */

//  ds = velocity*dt;

/* This determines if a path point is on the right or the left of a vehicle given the location of a vehicle and its heading. The program returns +1 if the vehicle is to the left of the path, 0 if the vehicle is on the path, and -1 if the vehicle
is to the right of the path. */

vvector[0] = cos(vheading);
vvector[1] = sin(vheading);

pvector[0] = xpath[j] - xactlast;

crossproduct = pvector[0]*vvector[1] - vvector[0]*pvector[1];

if(crossproduct == 0)
    pfsign = 1.0;
else
    pfsign = crossproduct / fabs(crossproduct);
}

/* Path Control Effort */
// Fbbypath = -1.0*pfsign*KPATH*error;

/* Heading erroror Calculations */
pheading = atan2((ypath[j] - ypath[j-1]),(xpath[j] - xpath[j-1]));
if (pheading < 0) {
    pheading = 2*PI + pheading;
}

vvector[0] = xactlast - xactlastlast;
vvector[1] = yactlast - yactlastlast;

pvector[0] = xpath[j] - xpath[j-1];
pvector[1] = ypath[j] - ypath[j-1];

crossproduct = vvector[0]*pvector[1] - pvector[0]*vvector[1];

if(crossproduct == 0) {
    hfsign = 1.0;
} else {
    hfsign = crossproduct / fabs(crossproduct);
}

heading_error = fabs(pheading - vheading);
/* This compensates for errors due to transition of heading from slightly less than 360 to slightly more than 0. */
if (heading_error > PI) {
    heading_error = 2*PI - heading_error;
}

/* Heading Control Effort */
// Fbbyhead = hfsign*KHEADING*heading_error;

/* Nominal Steering Effort */
// Fbbyn = MASS*velocity*velocity*curve[j];

/* Total Steering Effort */
// Fbby = Fbbypath + Fbbyhead + Fbbyn;
total_angle = angle(rho, hfsign, heading_error, pfsign, error, velocity); /* Compute the desired steering angle */
steering_out(total_angle); /* Send the steering signal to the steering servo */

/* Convert wheel encoder counts to wheel travel length */
left_wheel_distance = left_encoder() * PI * WHEELDIA / 4;
right_wheel_distance = right_encoder() * PI * WHEELDIA / 4;

/* Vehicle Model */
// dtheta = Fbby * ds / (MASS * (velocity * velocity));
rho = 2 * (right_wheel_distance - left_wheel_distance) / (TRACKLENGTH) * (left_wheel_distance + right_wheel_distance);
dtheta = rho * ds;
vheading = vheading + dtheta;

ds = (left_wheel_distance + right_wheel_distance) / 2;

dx = ds * cos(vheading);
dy = ds * sin(vheading);

xact = xact + dx;
yact = yact + dy;

xactlastlast = xactlast;
yactlastlast = yactlast;

xactlast = xact;
yactlast = yact;

s = s + ds;
k++;

disttrav = s;

// printf("%d\n", k);

// fhandle = fopen("data.txt", "a");
// fprintf(fhandle, "%d %d", k, j);
// fputc(' ', fhandle);
// fprintf(fhandle, "%f %f %f %f", xpath[j], ypath[j], xact, yact);
// fputc('n', fhandle);
// fclose(fhandle);

}

return 0;

}

double angle(double rho, double hfsign, double eh, double pfsign, double ep, double v)
{
    double angle_nom;
    double angle_path;
    double angle_heading:
double total_angle;

angle_nom = WHEELBASE * rho;
angle_path = -1*pfsign*KPATH * ep * WHEELBASE / (MASS*v*v);
angle_heading = KHEADING * e * WHEELBASE / (MASS*v*v);

total_angle = angle_nom + angle_heading + angle_path;

return total_angle;
}

void speed_out(double v)
{
}

void steering_out(double ang)
{
}

int left_encoder(void)
{
    int left_encoder_counts;

    return left_encoder_counts;
}

int right_encoder(void)
{
    int right_encoder_counts;

    return right_encoder_counts;
}

int velocity_change(double velocity, double patharr[3][3], int pathseg[500], double disttrav, int j, int N)
{
    /* This function determines if the vehicle needs to accelerate, decelerate, or keep its speed constant based on the next two segments in the path. This summarizes the important information about the next two segments in the array "pathdata". If the current segment is the last segment (no next segment) the vehicle must slow down and stop */

double pathdata[4][2];
double ddecel;
double daccel;
int Case; /* Velocity profile case */
double v0; /* Variables used in speed control */
double v1;
double v2;
double v3;
double t1;
double t2;
double t3;
double d1;
double d2;
double d3;
int velchnge;

pathdata[0][0] = patharr[0][pathseg[j]]; /* Current path segment length */
pathdata[1][0] = patharr[1][pathseg[j]]; /* Current path segment curvature */
pathdata[2][0] = patharr[3][pathseg[j]]; /* Current path segment orientation */
if (pathdata[1][0] > AMAXLAT/(VMAX*VMAX)) {
    pathdata[3][0] = sqrt(AMAXLAT/pathdata[2][0]); /* Current path segment maximum speed */
} else {
    pathdata[3][0] = VMAX;
}
if (pathseg[j] == N) {
    pathdata[0][1] = 0; /* next path segment length */
    pathdata[1][1] = 0; /* next path segment curvature */
    pathdata[2][1] = 0; /* next path segment orientation */
    pathdata[3][1] = 0; /* next path segment maximum speed */
} else {
    pathdata[0][1] = patharr[0][pathseg[j]+1]; /* path segment length */
    pathdata[1][1] = patharr[1][pathseg[j]+1]; /* path segment curvature */
    pathdata[2][1] = patharr[3][pathseg[j]+1]; /* path segment orientation */
    if (pathdata[1][1] > AMAXLAT/(VMAX*VMAX)) {
        pathdata[3][1] = sqrt(AMAXLAT/pathdata[1][1]); /* path segment maximum speed */
    } else {
        pathdata[3][1] = VMAX;
    }
}

ddecel = 0.5*(pathdata[3][0]*pathdata[3][0] - pathdata[3][1]*pathdata[3][1]) / AMAXLON;
daccel = 0.5*(pathdata[3][0]*pathdata[3][0] - velocity*velocity) / AMAXLON;

if ((daccel + ddecel) >= (pathdata[0][0]-disttrav)) {
    Case = 1;
} else {
    Case = 2;
}

if (Case==1) {
    v0 = velocity;
    v2 = pathdata[3][1];
    d2 = pathdata[0][0] - disttrav;

    v1 = sqrt(AMAXLON*d2 + 0.5*v0*v0 + 0.5*v2*v2);
    t1 = (v1 - v0)/AMAXLON;
    d1 = (v1 + v0)*t1 / 2;
    t2 = (v1 - v2)/AMAXLON + t1;

    if (t1>0) {

86
if (isreal(t1) == 1) { *
  velchngge = 1;
/*
} else {
  velchngge = -1; */
/*
} */
else {
  velchngge = -1;
}

if (Case==2) {
  vO = velocity;
  v1 = pathdata[3][0];
  v2 = pathdata[3][0];
  v3 = pathdata[3][1];
  d3 = pathdata[0][0] - distrav;
  t1 = (v1 - vO)/AMAXLON;
  d1 = 0.5*(v1*v1-vO*vO)/AMAXLON;
  d2 = d3 - 0.5*(v1*v1 - v3*v3)/AMAXLON;
  t2 = (d2 - d1)/v1 + t1;
  t3 = t2 + (v1-v3)/AMAXLON;
  if (t1>0) {
    velchngge = 1;
  } else if (t2>0) {
    velchngge = 0;
  } else {
    velchngge = -1;
  }
}

return velchngge;