5-1-2001

Design and Implementation of a Path Planning and Path Following Car-Like Robot

Daniel Houy
Trinity University

Andrew Leininger
Trinity University

Jeff Liddle
Trinity University

Michael Sutton
Trinity University

Executive Summary

May 1, 2001

This document describes the design, implementation, testing, and results of a mobile robot called $\mathcal{MoRoN}$ (Mobile Robotic Path Planning and Navigation). The work herein, is part of Trinity University's design series and is in partial fulfillment of the group member's requirements for a degree in Engineering Science.

First the document gives a brief overview of the problem of path planning and the objectives considered during the design of $\mathcal{MoRoN}$. A description is given of the main different components of the final design and how they interact. These main components include the odometry system, the path planner, and the local path planner. The odometry system keeps track of the location of the robot. The planner finds a path between two points in complicated environments. The local path planner determines the radius of curvature and distance between two points as would be followed by a car-like robot. Next the testing methods for the overall robot as well as some of the critical components are detailed including expected results from each test. The results of the tests are provided and discussed. $\mathcal{MoRoN}$ met *all* of the objectives that the group set in the beginning of the design process. The performance of the final implementation of $\mathcal{MoRoN}$ was significantly better than the initial design specification. The initial design specification was an error no greater than 20% of the distance traveled by the robot. The final test showed $\mathcal{MoRoN}$ performing with an error almost as low as 10%. The results also shows $\mathcal{MoRoN}$'s ability to plan paths in complicated environments. The path planner is able to take into account the fact that the robot is a car like robot and can only move in straight lines or arcs (like a car, it cannot parallel park by moving directly to the right or left). An overall conclusion and recommendation for the project is given that explains the success of the robot.

Additionally, several appendices are included. A user's manual is attached that describes in detail how to construct and operate the robot. Also included are summaries of the individual group member's contributions to the project and a time line of the entire project. For the year the project used \$896.25 of the allotted \$1000 budget and the group members sent a total of 922.5 hours (at an average cost per hour of \$22.50 this would be a total cost of \$20,756.25). The final portion of this report contains all of the source code for the project and a specification sheet for the hardware components.

Design and Implementation of a Path Planning and Path Following Car-Like Robot

Daniel Houy
Andrew Leininger
Jeff Liddle
Michael Sutton

Trinity University
Department of Engineering Science
Engineering Design VIII
Dr. Kevin Nickels, Advisor

May 1, 2001

Abstract

This report gives a description of the Mobile Robot Path Planner and Follower project. This description contains documentation for individual components as well as documentation for the whole mobile robot. These component descriptions are accompanied by appropriate theory and background. Also documented here are the tests performed on each component, as well as a test of the system as a whole. The results of those tests are given including a comparison to expected results. The final robot system performed excellently, obtaining an error level half that required of the system. Finally, some suggestions regarding the continuation of this project are presented. Additionally, six appendices are included: Construction Manual, Operation Manual, Group Member Contributions, Time-line, Costs, and Component Specifications. The entire source code is included as a separate document.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

This report provides a description of the final design, implementation, testing, and test results of the Mobile Robotic Navigator ($\mathcal{MoRoN}$). It also contains a number of ancillary documents relating to the mobile robot senior design project.

Path planning is an important problem in the field of mobile, autonomous robotics. Autonomous robots have the potential to benefit society in a number of ways. For example, robots that could work in hazardous environments (part of which consists of planning paths in those environments) could be extremely useful. More examples include courier robots that deliver mail or robots that vacuum floors or mow lawns (all of which would need some type of path planning). Path planning is critical to the manufacturability, sustainability, reliability, and marketability of any of these types of robots. If the path planning fails on any of these robots, the overall task of the robot would fail.

There has been extensive research into the task of path planning, yielding many different planning algorithms. Latombe's book on path planning presents a good description of the most common approaches to solving the path planning problem [7]. Path planning is the process of finding a path between an initial position and a final goal position when given information regarding the environment to be traversed. While often implemented using a robot, path planning in its purest sense need not involve a robot. When a planned path is applied to a mobile robot, several factors arise that influence the success of following

the planned path. For example, regardless of what path planning algorithm is used, it is necessary for a robot to be able to determine its location as it travels. The simplest way to do this is to use odometry[1], which is what *MoRoN* uses. However, when a robot uses only odometry for localization, small errors (such as wheel slip) are magnified as the robot travels greater distances. This can cause huge discrepancies between the actual location of the robot and the perceived location of the robot. Even when the robot has perfectly *planned* a path, it still may fail to follow the path because of localization difficulties. Additionally, the motor and steering must be actuated in accordance with the desired path. This project address three fundamental issues: planning, control (of motor and steering), and localization. The end result is a robot that can plan sophisticated paths and can move along those paths using a simple localization scheme.

1.1 System Overview

The *MoRoN* system has the following components, as shown in Figure 1.1: Global Path Planner (GPP), Segmenter, Localizer, Executive, Local Path Planner (LPP), and Local Path Follower (LPF). These components interact in the following way, and are described in the order of top to bottom in Figure 1.1. First, the user inputs initial and goal points to the GPP. *MoRoN* implements the Approximate Cell Decomposition (ACD) method for global path planning, an algorithm described in section 2.3. The GPP outputs a series of points that define a path through the environment to be traveled. Those points are inputs for the Segmenter, a module that interpolates points between the points from the GPP. Parallel to the Segmenter, the Localizer receives odometry information and calculates the robot's location. Next, the Executive module takes the Segmenter's points, combines them with the Localizer's points and sends those points to the Local Path Planner (LPP). The points sent by the Executive are the Localizer's point indicating where the robot is located,

[1]One example of this is to put a shaft encoder on the wheels of the robot. Accurate localization is a significant problem. More accurate techniques quickly become complicated and difficult to implement.

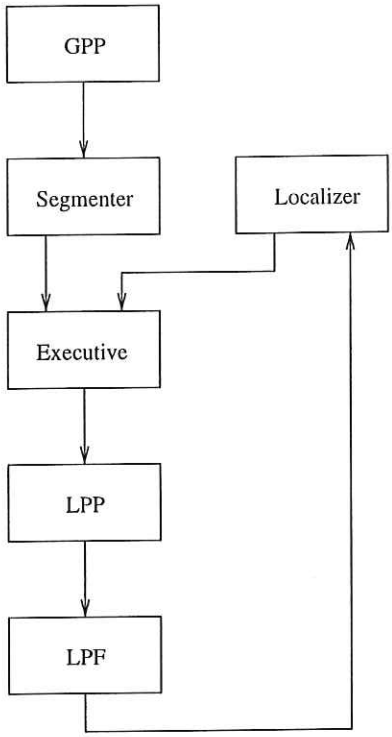

Figure 1.1: Mobile Robotic Path Planner architecture.

and the next point needed to be reached, given by the Segmenter. The LPP computes an arc between these points, and sends steering and drive motor commands to the Local Path Follower (LPF), which in turn sets the robot's steering servo and drive motor such that the robot moves along those curves.

1.2 Software Overview

All of the system components mentioned in Section 1.1 are software-based. The GPP, Segmenter, Localizer, Executive, and LPP are stored on and execute from a laptop computer on the robot. The LPF software is stored on and executes from a Handyboard[9], a microcontroller created by M.I.T. that uses a Motorola 6811 microprocessor and has an array of input/output (I/O) ports. It is also on-board the robot.

1.3 Hardware Overview

The robot design details two types of hardware for the robot. The first set of hardware components includes the laptop and the Handyboard. These components are the robot's computational and operational center. The second set of hardware on the robot includes the individual components that allow for path following. This component set includes the digital compass, two optical encoders, steering servo and drive motors, and a switch that allows for the user to take control of the robot. With the exception of the manual control switch, all of these components are software driven. The final robot design differs slightly from the initial design due to changes made after the testing of the prototype. The final design is documented fully in Chapter 3.

1.4 Chapter Summary

The first few weeks of the project were spent developing reasonable objectives for the project. Detailed here is a summary of those specifications.

1. The robot has complete *a priori* knowledge of the environment to be traversed.
2. The robot does not encounter moving objects; there will be no mobile objects in the environment.
3. A resolution complete, non-holonomic path is produced.
4. If no path exists within the current resolution, the planning module notifies the user that no path exists.
5. The planning algorithm is able to plan a path on an order of complexity sufficient for traversing MEB room 305.
6. There is a switch that enables manual, joystick-based control or software (self) control of the robot.

7. The Handyboard issues commands to the motor and steering servos to follow the planned path.

8. The laptop and Handyboard have bidirectional communication.

9. The location error of the robot must be within 20% of the planned path length at all times.

10. The robot must have a switch to kill power to the motors.

Chapter 2

Component Descriptions

This chapter contains detailed descriptions of the hardware components and software modules that comprise MoRoN's system. First, the robot's physical description is presented.

2.1 Robot Hardware

Basic hardware was inherited from a previous design group, including the R/C vehicle, Handyboard, compass, and encoder. A complete list of parts and their specifications are in Appendix F.

Figure 2.1 shows the side and front views of the constructed prototype, respectively. Additionally, Figure 2.2 shows a photo of the robot. The construction is detailed in Appendix A.

2.2 Odometry System

To accurately follow a given path, the robot must be able to determine its current location. An odometry system comprised of hardware and software components is used to generate, record, and compute location information.

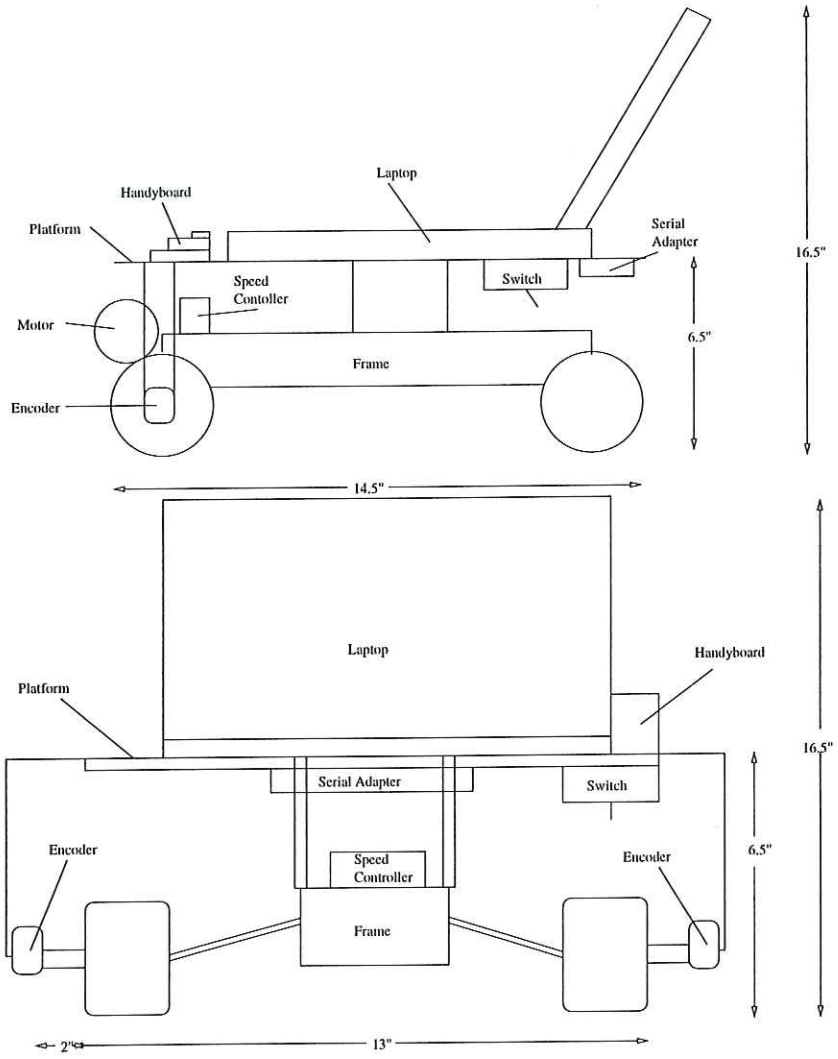

Figure 2.1: Front and side view of the final robot

2.2.1 Hardware

The initial design of the odometry system hardware includes a digital compass and an optical shaft encoder. These two pieces of hardware and the Localizer determine the robot's position. However, when the robot is used in the Moody Engineering Building, the compass readings are inaccurate due to the magnetic fields in the building that interfere with the Earth's magnetic fields. To accurately determine the robot's position inside, a second design for the odometry replaced the initial design. The final version of the odometry system consists of two optical encoders, one on each of the rear wheels. This odometry system accurately

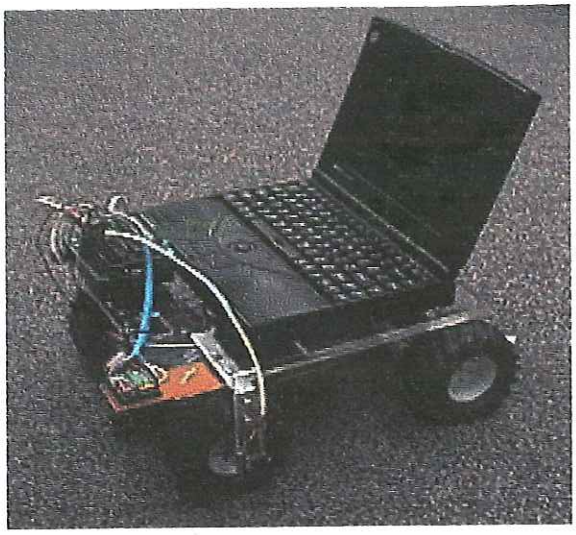

Figure 2.2: Photograph of MoRoN

determines the robot's position inside and outside.

Compass and Encoder

The compass and encoder system provides enough information to determine how far the robot travels and in which direction. The compass is mounted on the rear right corner of the robot. It measures the angle of the robot based on the magnetic fields of the Earth. The optical encoder consists of a shaft and optical disk with 50 transparent slots. The shaft of the encoder is connected to the axle of a wheel attached to a trailer behind the robot. For each full revolution of this wheel, the encoder generates 50 ticks. The information from the compass and encoder are sent to the Localizer in pairs. From these pairs, the position of the robot is determined.

Dual Encoders

The second design of the odometry system consists of mounting optical encoders on the back two wheels of the robot. These two encoders are identical to the encoder used in the first design. The distance each back wheel travels is measured independently of the other wheel and is used to determine the robot's position through dead reckoning. This system provides

knowledge about the robot's course and speed over a period of time, therefore, the position of the robot can be determined. Although slight errors could exist, the encoders generally provide a satisfactory estimate of displacements for the each back wheel. The Localizer uses these displacement values to calculate the robot's position. The only problem with this odometry system is that the initial robot orientation needs to be entered manually, and there is no way the robot can correct itself if it is aligned slightly askew in the beginning of path following.

2.2.2 Software

The Localizer is the software that receives as input the odometry information from the Handyboard, and computes the robot's location. The prototype used a single encoder and a compass. While testing the prototype indoors, the magnetic interference made the system inadequate. A dual encoder odometry system was used in the final robot to eliminate the dependence on the compass and inaccuracy of the robot indoors.

Compass and Encoder

In the first design, the Localizer inputs are the previous location of the robot and an array of encoder and compass reading pairs. To find the new location of the robot, the Localizer finds a small vector of movement from each encoder and compass pair. Each pair gives a distance traveled and an angle of orientation. The sine and cosine of the angle are multiplied by the difference in the current and previous encoder values, yielding the y and x components of movement for each pair. These values are then added to the previous position of the robot to formulate the current location. The new position can be computed with the following equations.

$$\theta = (\theta_1 + \theta_2)/2, \tag{2.1}$$
$$x = \bar{s}\cos\theta + x_o, \tag{2.2}$$
$$y = \bar{s}\sin\theta + y_o. \tag{2.3}$$

$\bar{s}$ is the distance traveled between two pairs of data.

Dual Encoders

In the second design, the Localizer takes as input the previous location of the robot and an array of the two encoder pairs. These two values are then used in a few simple equations to determine the distance that the robot traveled and its new angle of orientation. The premise of these equations is the simple geometry shown in Figure 2.3. Lucas[8] goes through a

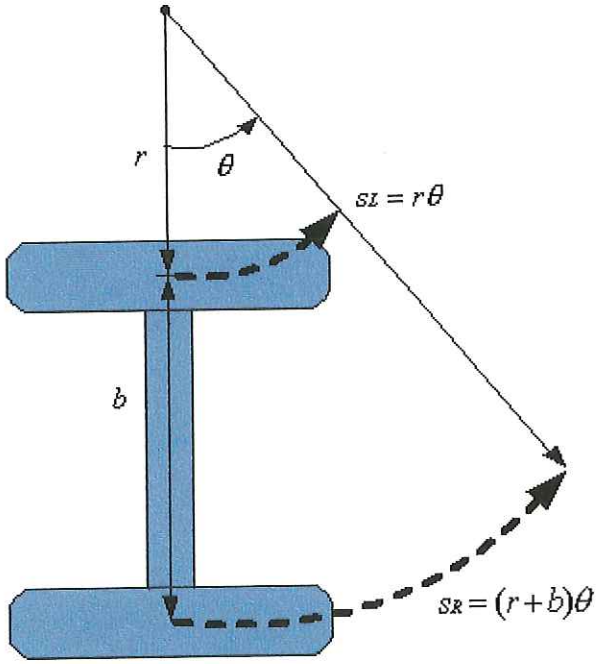

Figure 2.3: Dual Encoder Geometry [8].

detailed derivation of the kinematics equations governing the motion of the robot shown in Figure 2.3. He presents the following set of equations that approximate the motion of the robot:

$$\bar{s} = (s_r + s_l)/2, \tag{2.4}$$
$$\theta = (s_r - s_l)/b + \theta_o, \tag{2.5}$$
$$x = \bar{s}\cos\theta + x_o, \tag{2.6}$$
$$y = \bar{s}\sin\theta + y_o. \tag{2.7}$$

The initial position of the robot is described by x_o, y_o, and θ_o, and these equations determine the new position of the robot, given by x, y, θ. The distance traveled forward by the robot is $\bar{s}$; s_r and s_l are the distances traveled by the right wheel and left wheel, respectively; b is the distance between the two wheels. These approximations are accurate provided the distance traveled is relatively short. In the Localizer, these values are calculated for every pair of encoder data collected by the Handyboard. Generally, there is less than 10 ticks (which represents approximately 7 cm) between each pair of encoder values. For this distance, the error between these equations and the actual position is less than 1 tick.

2.3 Global Path Planner (GPP)

The following is a description of the approximate cell decomposition (ACD) method described by Latombe[7]. ACD determines a path for a robot by breaking the free configuration space (C-space) into discrete rectangloids of space, called cells. Once the empty cells in the configuration space[1] are found, they can be put into a graph where adjacent cells are connected. This graph can then be searched to find a path from the robot's initial position to the goal position. In approximate cell decomposition, the configuration space is divided up in equal sized cells. Each cell can be labeled as open, mixed, or full. A open cell is a cell that is completely in free space. A mixed cell is a cell that is partially occupied by a C-space obstacle. A full cell is a cell that lies completely inside an obstacle. Full cells and empty cells make the path planning process relatively easy. Mixed cells, on the other hand, are not so easy. They may be obscured only at the corner and for the most part free; they could provide the shortest path to the goal, but be unusable because of a small filled portion. Some methods incorporate breaking mixed cells down further until a free path is found. In this project, mixed cells are considered full for planning purposes. Therefore, the paths that are found are resolution complete paths. In other words, if a path exists within the resolution of the cells, it will be found. However, this does not guarantee finding a path if it exists.

[1]For a more complete description of configuration space, or C-space, see [12].

2.3.1 Path Planning Algorithm Overview

The configuration space for $\mathcal{MoRoN}$ has three dimensions. This means that the robot has three degrees of freedom and can translate and rotate in the workspace. The first two dimensions are x and y, and the third is θ_{robot}, which corresponds to the angular orientation of the robot. The steps to create the navigation algorithm are:

1. Determine the configuration space from the workspace for discrete orientations corresponding to the number of cells desired in the θ_{robot} direction.
2. Determine the adjacencies (see Section 2.3.2).
3. Construct a search graph using the adjacency information. Each free cell becomes a node in the graph. The nodes are connected based on the adjacency information.
4. Determine the solution by using a graph searching technique (see Section 2.3.4).

2.3.2 Generating Adjacencies

$\mathcal{MoRoN}$ is nonholonomic, or car-like. A nonholonomic[2] robot is a robot that *cannot* move instantly in any direction. For example, a car can not rotate without moving. ACD must take this into account.

Kinematic Constraints

Latombe ([7] - Chapter 9) derives the nonholonomic constraints for a car-like robot in great detail. This section will only list those constraints that will be of interest for this project.

Consider the robot shown in Figure 2.4. This robot's motion is constrained by:

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0 \tag{2.8}$$

[2]See [12] for more details on holonomic versus nonholonomic robots.

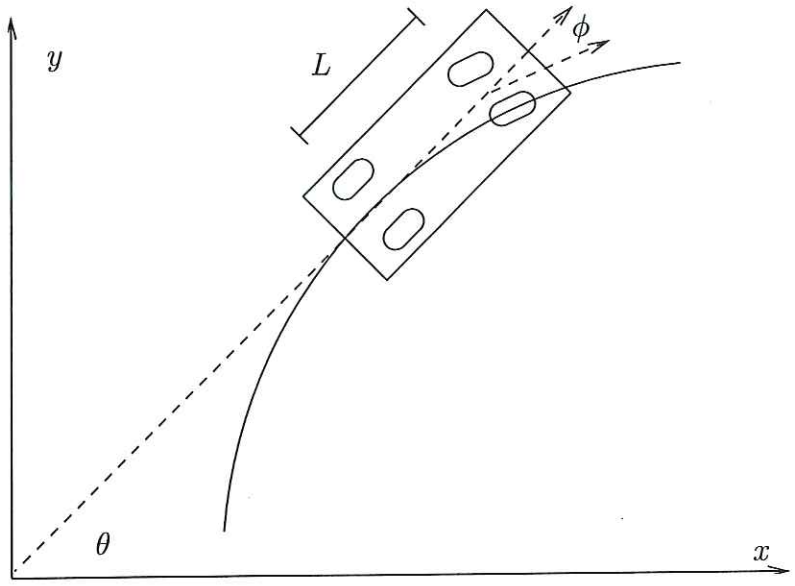

Figure 2.4: A car-like nonholonomic robot

This shows that the path must be either straight lines or arcs. However, the maximum steering angle is not taken into account by this constraint. This robot has some steering angle ϕ such that:

$$|\phi| \leq \phi_{max} < \frac{\pi}{2} \tag{2.9}$$

Likewise, the minimum turning radius ρ_{min} drawn out by a point between the back two wheels is:

$$\frac{1}{\rho_{min}} = \frac{1}{L} \tan \phi_{max} \tag{2.10}$$

where L is defined as shown in Figure 2.4. The steering angle constraint given in (2.9), can also be written as:

$$|\dot{\theta}| \leq \frac{|v|}{\rho_{min}} \tag{2.11}$$

where $|v|$ is the speed of the robot. This can also be expressed as:

$$\dot{x}^2 + \dot{y}^2 - \rho_{min}^2 \dot{\theta}^2 \geq 0 \tag{2.12}$$

These constraints must be satisfied at all times, because $\mathcal{MoRoN}$ is a nonholonomic robot. This can be done by taking care in generating the adjacencies used for the path planning. The adjacencies are computed by looking at each cell individually and testing which cells the robot can move to. The robot's motion must meet the nonholonomic constraints. For a car type robot, the nonholonomic constraint is a minimum turning radius. If in moving from one cell to the next the turning radius is greater than the minimum and it is a "short" distance away, it can be said to be adjacent. A short distance means a distance that takes the robot out of the current cell and no further than the neighboring cell.

Adjacencies were determined in the following fashion; for each cell, the algorithm:

1. Calculates the position that would result by moving straight forward a short distance from the cell.
2. Finds the cell that corresponds to this new position and adds it to the adjacency list.
3. Calculates the position that would result by moving forward at the maximum *right* turning angle a short distance from the cell.
4. Finds the cell that corresponds to this new position and adds it to the adjacency list.
5. Calculates the position that would result by moving forward at the maximum *left* turning angle a short distance from the cell.
6. Finds the cell that corresponds to this new position and adds it to the adjacency list.

This approach may not find *all* possible adjacencies because it is possible that there is a cell that would be adjacent if it were to turn less than the maximum. This can be addressed by checking for adjacencies at various turning angles, which has the disadvantage of making the search graph larger and thus making the time to find a path longer. It would be interesting for future work to experiment with these variables.

2.3.3 Marking Cells

Each cell can be marked empty, full, or mixed. An empty cell is completely free of any obstacles, a full cell is completely filled with an obstacle or obstacles, and a mixed cell contains some free space and at least some part of an obstacle. A modified Cohen-Sutherland Clipping Algorithm [1] is used to mark the cells. This algorithm determines if a line is within a certain clipping region while reducing the number of line intersections that need to be calculated. Each line segment of every polygonal obstacle is looked at with respect to this clipping algorithm. The following is an example of the algorithm to determine open, full, and mixed cells:

1. Determine if all line segments making up the polygonal obstacles are outside the cell.
2. If a line goes through the cell, the cell is known to be mixed.
3. If all lines are outside, there are two possible cases, empty or full.
   (a) Full: At least one point describing the polygon in question must be above AND at least one point must be below AND at least one point must be to the right AND at least one point must to the left.
   (b) Empty: otherwise.

2.3.4 Searching the Graph

A graph is data structure that contains nodes connected by vertices. In this case, the nodes are the cells and the vertices are determined by the adjacency information. There are several different methods to search a graph. Some methods involve finding a solution and not attempting to find the least cost solution.[3] One search algorithm that is easy to implement is depth-first search. The simplest way to express depth-first search is shown in Figure 2.5.

[3]The paths connecting graphs can be generalized to other uses than mobile robot motion planning–hence the term least cost. In different applications, it might be desirable to minimize something else while moving through the graph. In our case we are interested in finding the shortest distance, which means the cost function is distance traveled.

With this type of algorithm, the cost is not necessarily minimized between nodes. However

```
DFS(Node n) {
    visit and mark Node n;
    while ( n has unmarked adjacent nodes ) {
        DFS(n.getAdjacent());
    }
}
```

Figure 2.5: Depth-first search algorithm

it is possible to minimize the cost with a recursive depth-first search, which is found to add considerable complexity to the algorithm. Figure 2.6 shows the algorithm used for MoRoN. Once this function completes, the path can be generated by tracing back from the goal node, which will eventually lead to the initial position or to a null parent pointer. If a null pointer is encountered that is not the initial node then it is known that no path was found for the given resolution. If this happens, the entire process can be rerun with a smaller grid size.

2.4 Segmenter

For the robot to function it must first be able to plot a nonholonomic path between the endpoints. This plot and corresponding x, y, θ values are then sent to the LPP. The plot must resemble a smooth curve with wide enough turns for the robot to navigate. One method for developing these curves is by using a spline function [2, 4]. The function must be able to plot a path through a series of points in the desired angle of orientation. For these constrains, a spline function is the best option, because it is easier and more applicable to our problem. The spline function is implemented in the Segmenter component to output the desired intermediate points to the LPP.

The Segmenter was used in the prototype to smooth the curves. In the final design it is no longer needed, because the output of the path planning algorithm will be smooth enough for the robot to follow. For the prototype the points were manually entered. These points

```
DFS(Node n, float distTraveled) {
    if ( n is goal node )
        return;
    for ( all nodes adjacent to n ) {
        Node a = adjacent node;
        if ( a has been visited ) {
            if ( dist from n to a + dist < the distance stored in a )
{
                // reset the data in node a
                a.setDist(dist + (distance from n to a));
                a.setParent(n);
                // A recursive call here finds the minimum cost
                // path however, in so doing it adds a good deal of
                // complexity to the algorithm.
                DFS(a, newDistance);
            }
        } else {
            // has not been visited before
            a.setDist(dist + (distance from n to a));
            a.setParent( n );
            DFS(a, newDistance);
        }
    }
}
```

Figure 2.6: Depth-first search applied to path planning

x, y, θ define a path from a starting point to a final goal point. The Segmenter takes these points and fits a smooth curve to them, it then outputs an array of intermediate points. This smooth curve is produced by using a Hermite cubic function [10, 14, 6, 3, 5]. This function defines the positions and tangents at the curves endpoints. At each coordinate of points, the control points and their corresponding derivatives are defined. If the start point is given as $P_S = (1, 2, \pi)$ and the goal point as $P_G = (3, 4, \pi/2)$ then $P_{Sx} = 1$, $P_{Sy} = 2$, $P_{S\theta} = \pi$, $P_{Gx} = 3$, $P_{Gy} = 4$, and $P_{G\theta} = \pi/2$. The control points would be defined as the position coordinate plus the cosine of the angle for that point. For the first coordinate the controlling point would be

$$P_{SCx} = P_{Sx} + cosP_{S0}. \tag{2.13}$$

It is the same for each P_S and P_G point. These control points help to specify a derivative for the curve that is used to help determine the final path between the two points. To find the derivative for start position x, y, the difference between the control point and the initial point is found and is divided by D where $D = 1/L$ and $L > 0$ and determines the position of the control points from the endpoints. Therefore the derivative of the first coordinate is

$$P'_{Sx} = (P_{SCx} - P_{Sx})/D. \tag{2.14}$$

A row vector T of powers of t is also needed for the function, where t defines the number of intermediate segments between the endpoints. Using T, the endpoints, and the derivatives, the characteristic equations for each coordinate can be found. The equation is a four by four matrix multiplied by a column vector of the coordinates of the endpoints and their derivatives [6]. For the x coordinate the equation is

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{Sx} \\ P_{Gx} \\ P'_{Sx} \\ P'_{Gx} \end{bmatrix}. \tag{2.15}$$

The x position for the curve is then found by the equation

$$x = TC_x. \tag{2.16}$$

Repeating this and solving for the corresponding y values gives the other coordinate necessary to create the curve. After the x and y values are know then a plot of x versus y can be made to give the Hermite curve.

Now that a smooth curve can be fit between two points, an array of points along that curve can be found. These points are taken and input into the LPP where the LPP attempts to fit an arc between the segmented points.

2.5 Local Path Planner (LPP)

The Local Path Planner plays a very important role in the robot's execution of a path. The LPP is given the robot's current position from the Executive in the form: (x,y,θ). The first two points are the Cartesian x and y coordinates of the robot. The third point is the angle of orientation of the robot, where the angle increases counterclockwise from 0 on the positive x-axis. The LPP is then given the robot's next goal position. Even though the point is in the same format as the current location of the robot, the LPP uses only the x and y coordinates. This is because the first point including orientation and the second point excluding orientation is enough to define an arc for the robot to follow. Using the current location and orientation, and the destination point, the LPP constructs a curve with a constant radius. It then sends the information about the distance and radius of curvature of the path to the Handyboard.

To determine the equations for the LPP, sets of parametric equations were solved simultaneously. A basic circle is described by the two following equations:

$$\begin{aligned} x &= r\sin\alpha + x_o \\ y &= r\cos\alpha + y_o \end{aligned} \tag{2.17}$$

where x_o, y_o is the center of the circle, r is the radius of the circle, and α is the angular position on the circle in radians. For example, $\alpha = 0$ is always the rightmost point of the circle. The input to the LPP is the current position (x_1, y_1, θ_1) and the next position (x_2, y_2). This yields two pairs of parametric equations:

$$\begin{aligned} x_1 &= r\sin\alpha_1 + x_o \\ y_1 &= r\cos\alpha_1 + y_o \end{aligned} \tag{2.18}$$

and

$$\begin{aligned} x_2 &= r\sin\alpha_2 + x_o \\ y_2 &= r\cos\alpha_2 + y_o \end{aligned} \tag{2.19}$$

There are four equations and four unknowns. The unknowns are α_2, x_o, y_o, and r. The relationship between α_1 and θ_1 is: $\alpha_1 = \pi - \theta_1$. Solving these equations for α_2 and r it is possible to provide the direction to turn, the distance, and the turning radius. Maple was used to solve the simultaneous equations and produced the following equations[4]:

$$\begin{aligned} \alpha_2 = \arctan[&-(-2x_2y_2\cos(\theta_1) + 2x_2y_1\cos(\theta_1) + \sin(\theta_1)x_2^2 - 2\sin(\theta_1)x_2x_1 + 2x_1y_2\cos(\theta_1) \\ &- 2x_1y_1\cos(\theta_1) + \sin(\theta_1)x_1^2 - \sin(\theta_1)y_2^2 + 2\sin(\theta_1)y_2y_1 - \sin(\theta_1)y_1^2)/ \\ &(y_2^2 - 2y_2y_1 + x_1^2 + x_2^2 + y_1^2 - 2x_2x_1), \\ &-(-y_2^2\cos(\theta_1) + 2y_2y_1\cos(\theta_1) + 2y_2x_2\sin(\theta_1) - 2y_2x_1\sin(\theta_1) - y_1^2\cos(\theta_1) - 2y_1x_2\sin(\theta_1) \\ &+ 2y_1x_1\sin(\theta_1) + \cos(\theta_1)x_1^2 + \cos(\theta_1)x_2^2 - 2\cos(\theta_1)x_2x_1)/ \\ &(y_2^2 - 2y_2y_1 + x_1^2 + x_2^2 + y_1^2 - 2x_2x_1)] \end{aligned} \tag{2.20}$$

$$r = -\frac{1}{2}\frac{y_2^2 - 2y_2y_1 + x_1^2 + x_2^2 + y_1^2 - 2x_2x_1}{-y_2\cos(theta1) + y_1\cos(\theta_1) + x_2\sin(\theta_1) - x_1\sin(\theta_1)} \tag{2.21}$$

Given α_2 and r, the distance, d, can be computed by:

$$d = r\,d\theta \tag{2.22}$$

where $d\theta$ is the acute angle between α_1 and α_2.

[4]Note that the arctan function is the two argument function in this case.

2.6 Local Path Follower (LPF)

The LPF behavior described here can be seen in Figure 2.7. The first event in the Local Path Follower is the reception of a move command, via a serial link, sent to the Handyboard from the laptop. The radius of curvature and drive motor instructions are extracted from this move command, and the radius of curvature is translated to a steering servo setting through a look up table. Next, the LPF activates both the steering servo and the drive motors, causing the robot to move in the desired direction. During this time, the LPF is monitoring the odometry components and collecting data to send back up the serial link to the Localizer. The monitoring process involves reading the amount of encoder ticks that have accumulated, and disengaging the drive motors after 40% of the desired distance has been traveled. This behavior is introduced in an attempt to reduce the effect of coasting on the localization systems. The actual percentage was arrived at experimentally. The data collection process involves reading and buffering encoder ticks and preparing them for transportation back to the Localizer. A variable for each encoder is incremented by an Interrupt Service Routine (ISR) on each rising edge from the encoder[5]. This allows the LPF to simply read a variable that is being incremented in the background to learn the current encoder information.

2.7 Chapter Summary

This chapter discusses the hardware and software components that comprise the robot. Each component is shown in detail, including any necessary theory and mechanics. This chapter serves to illustrate the current implementation of the initial robot design, and to show how the robot evolved over the course of designing, building, and testing.

[5]The encoders are connected to the Input Capture (IC) mechanisms on the Handyboard's Motorola 6811, and since those mechanisms are configured to request an interrupt on each rising edge of the signal feeding the IC pins, each time an encoder sends a signal indicating another tick, an interrupt is requested and the encoder ISR increments a counter for that encoder.

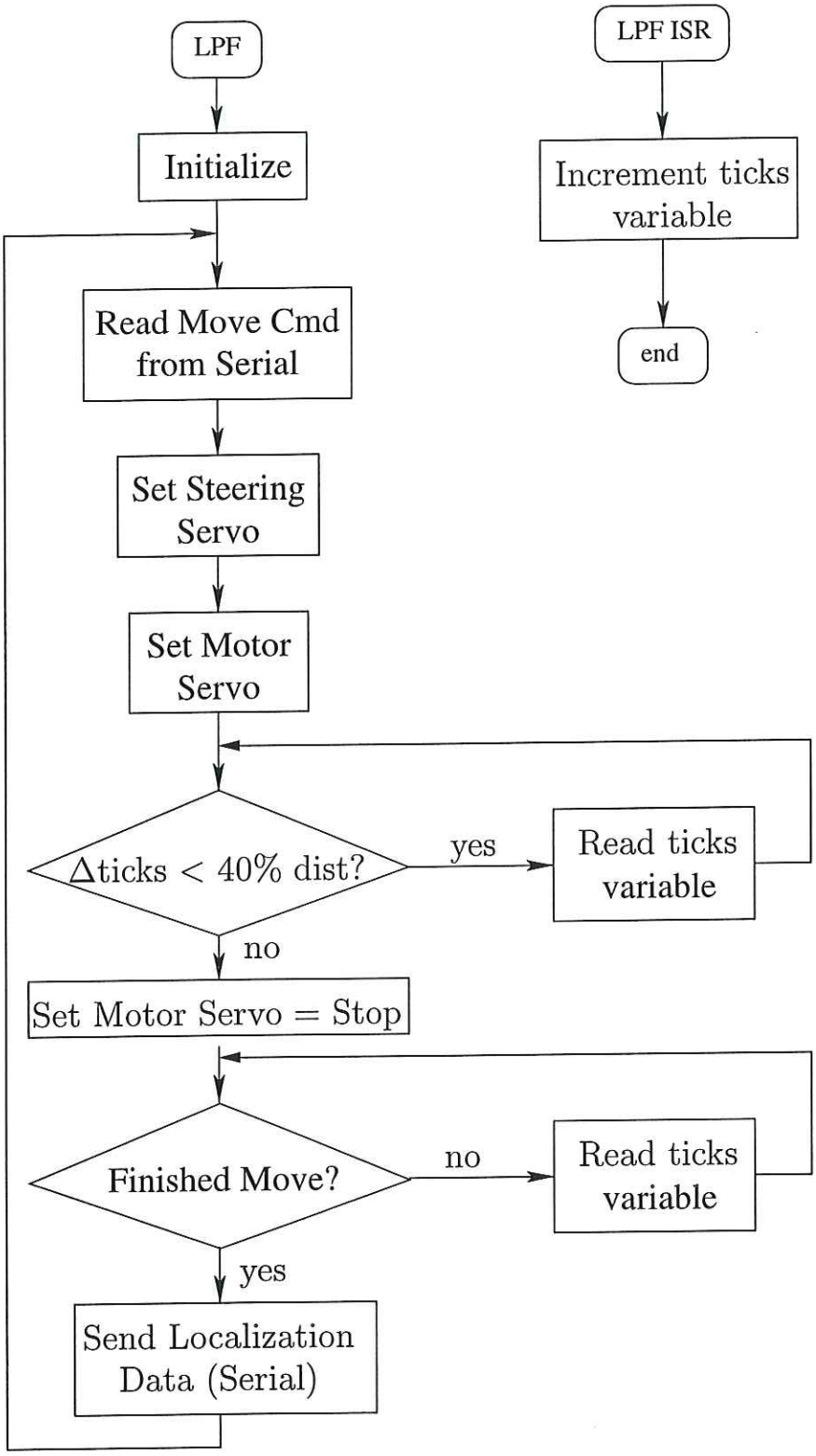

Figure 2.7: LPF Algorithm

Chapter 3

Testing Methods

This chapter describes the test methods for the components of the mobile robot described in Chapter 2. These tests are conducted with the intention of verifying the proper operation of each component. These tests are designed to verify the operation of the system as a whole and to determine the success of failure of the project.

3.1 Overall System

The overall system test of the robot is a "black box" test that includes all system components. A environment similar to Figure 3.1 is constructed using temporary obstacles that are drawn out on the floor with masking tape. This environment is drawn into xfig, a graphics drawing program, for input into the global path planner (GPP). The robot is then placed within the environment and the user inputs the goal position. The test is judged by whether the robot meets the 20% specification. The grid size is chosen to be approximately the size of the footprint of the robot with a resolution in the angular direction of 10 degrees. Several overall tests are performed to show the repeatability and consistency of the design.

3.2 Odometry System

To evaluate the odometry system of the robot tests are run for the compass and encoder system as well as the dual encoder system. For each system, the same test is used. The test consists of the robot moving in an arc through 15 different points. Each set of hardware is tested outdoors in the same location. A chalk mark is made when the robot stops at each of the 15 points. These chalk marks are then compared to the desired location of the robot and the position that is reported by the robot. Both the Encoder/Compass and Dual Encoder setup are run multiple times, to determine if the robot is consistent.

3.3 Global Path Planner (GPP)

The GPP has a series of internal parts that are tested along with the entire component. Much of the development on the internal parts of this module were done in a previous project and tested at that time. The tests included here are overall tests that show the functionality of the GPP as a whole.

The main concerns when testing the GPP are: 1) it should return a path from the specified initial position to the goal position if it exists 2) the path must avoid all obstacles and 3) the resulting path must take into account nonholonomic constraints. All of these can be visually observed in the output of the GPP.

Figure 3.1 shows an interesting path planning problem. Clearly, it could not be solved by "simple connect the dots" i.e. visibility graphs [7] and requires some intelligent planning. The planning area in this scenario is 5000 ticks by 5000 ticks (approximately 36 by 36 meters). This test will be able to use a relatively large grid size of 17 by 17 cells in the workspace and 36 possible angular orientations. A turning angle of 6° is used in finding the adjacencies. Figure 3.2 is an extension to the first test. This test is clearly more complicated than the first. It is expected for this to require a denser grid than the first test. However, it is uncertain whether the same turning angle can be used and the same number of cells in

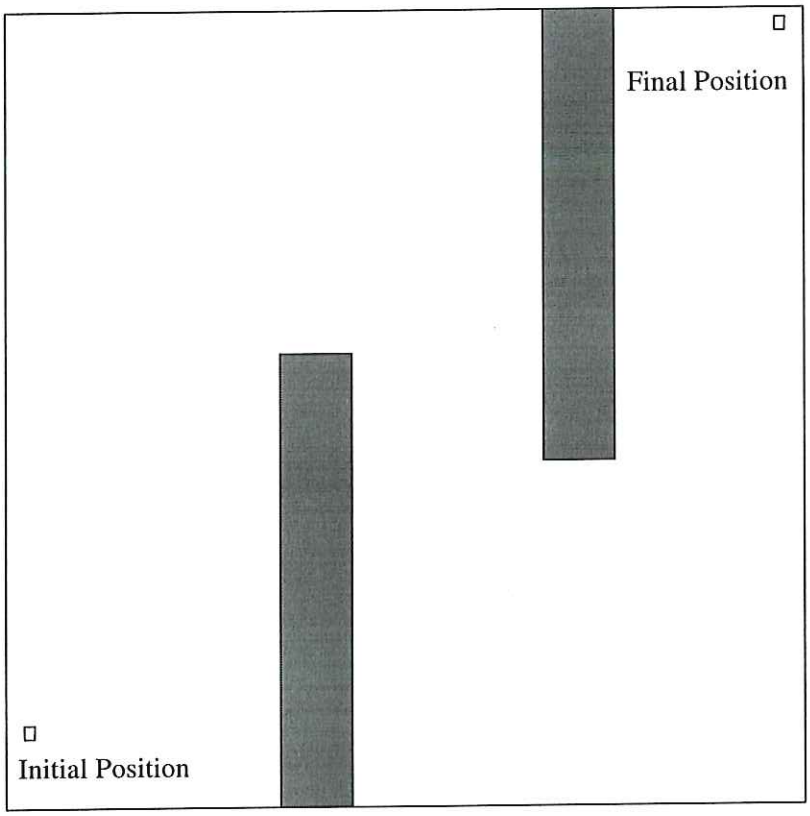

Figure 3.1: Test #1 input for GPP

the angular direction can be used. This is because the robot will be required to do tighter turning.

In this testing, several parameters can be adjusted, including the grid size (both in workspace and in angular rotation) and the turning angle used in finding the adjacencies. Most of these are set by trial and error methods in an attempt to find a trade off between performance (finding paths in tighter spaces) and time required to find a path (as the grid grows, the time required to compute grows exponentially).

3.4 Segmenter

The first test for the Segmenter receives the points (0, 0, 0) and (5, 5, $\pi/2$). These points are input into the Segmenter and a smooth curve connecting the points together should be

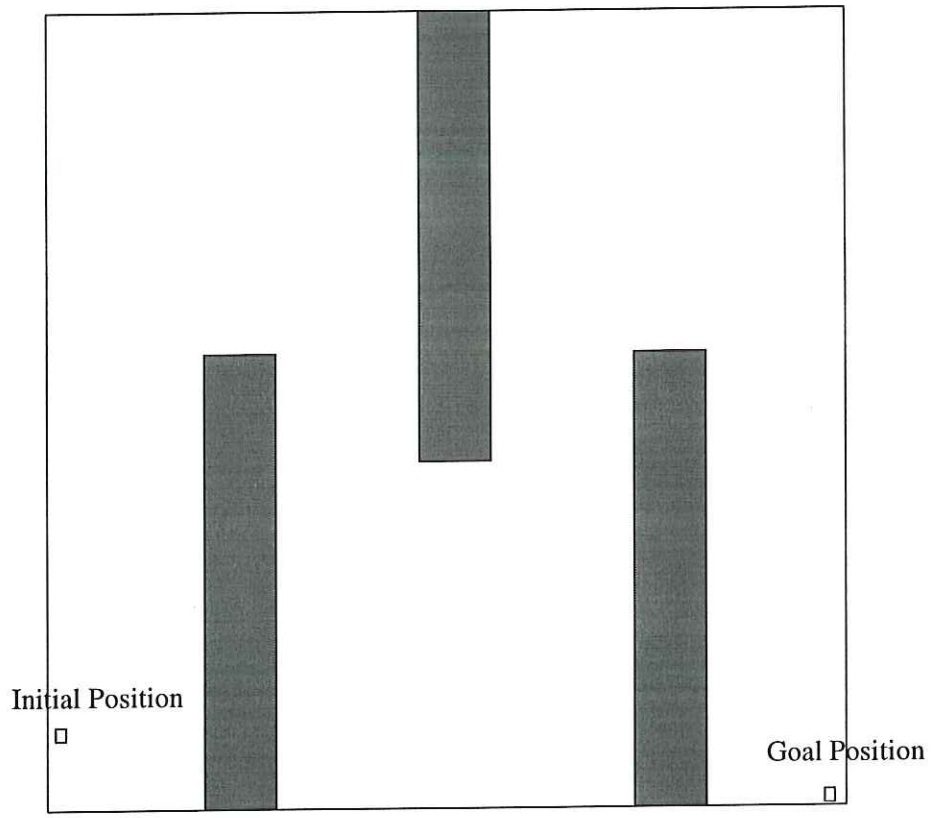

Figure 3.2: Test #2 input for GPP

produced, along with an array of x, y, θ values for points along the curve. The output array is expected to start at the given starting point and end at the given goal point and all intermediate points should lie along the curve connecting those two points. Additional tests for the Segmenter includes a straight-line path, a parallel parking path where two consecutive points have the same angle of orientation, and a path where the two points are close to each other (one unit or less) but have different angles of orientation. For each of these tests the expected output should be a smooth curve connecting the points with a corresponding array of points. The straight line test should produce a simple straight line between the points, the second should resemble a sine curve, and the third should be a curve that is considerably longer then the distance between the points and has a large curve or loop.

3.5 Local Path Planner (LPP)

The LPP is tested using a variety of point pairs. The initial point always consists of an (x, y) coordinate pair and the angle of orientation of the robot. The goal point consists only of an (x, y) pair. The turning radius and distance that the robot is to travel are then given as outputs. These numbers are then verified by hand calculations. The results of this test is found in Section 4.5.

3.6 Chapter Summary

In this chapter, the test methods for the robot are discussed. For each component of the robot that requires individual testing, a test is developed, executed, and documented here. An overall system test, also known as a black box test, is designed here as well. Included in these test descriptions are expected results of the tests. The actual results generated by the tests are shown in the next chapter.

Chapter 4

Results

This chapter contains a presentation of the results of the tests described in Chapter 3. The actual results are compared against the expected results for each test. Also included in this chapter is a discussion of the current status of the robot project.

4.1 Overall System Test Results

Several overall tests were run with considerable success. Initial tests were done for the prototype report[13]. At that time, it was found that use of the robot was restricted to magnetically clean areas, areas that are free of magnetic variations. The inside of the Moody Engineering building has considerable magnetic anomalies, which make use inside impossible. However, there is satisfactory performance outdoors. Figure 4.1 shows the result of the outdoor path using the compass and a single encoder. The path in this test is very simple and only uses the Segmenter because the test was performed before the GPP was incorporated into the design.

The design was changed as explained in the previous sections to use two encoders, one on each back wheel. This allowed for the elimination of the compass and thus the operation of the robot indoors. Additionally, the GPP was integrated into the system. Figure 4.3 shows the results of one of the overall tests performed. The obstacles shown (gray areas),

the input position, and the goal position are entered into the GPP via xfig. The GPP then generates a path connecting these two points, which is shown by the blue path with stars in Figure 4.3. The obstacles are drawn on the ground with masking tape. As the robot moves, the localizer continually updates the position, which is shown in the figure by the red line. Additionally, hand measurements are taken as the robot moves, shown by the black x's.

Overall, the robot performed within the project specifications. More than anything, the test shows all of the pieces working together. The GPP was able to find a path through a complicated set of obstacles while incorporating the constraints of the nonholonomic robot. The LPP, Localizer, and software on the Handyboard (in addition to the other components such as serial communication) worked together to move the robot along the planned path. All of these components had to be working for the results that are seen in Figure 4.3. It was found that for the path generated by the GPP it is unnecessary to use the Segmenter for additional smoothing between the points of the GPP. However, the capabilities of the Segmenter are much more sophisticated than what was originally anticipated. The Segmenter can be used when there are few planned points, especially in areas with few obstacles.

Additionally, the robot met the specification of being within the 20% of the path length specification. Figure 4.4 shows the error in the robot's position as it travels along the path. The error is determined by calculating the Euclidean distance between the hand-measured location of the robot and the points generated by the GPP, which represent where the robot should be at any given time. Additionally, the 20% of path length specification is shown by the dashed line. Except for a single point towards the beginning of the path, all of the points fall well below the specification. The single point that does exceed the specification can be explained by the method used to calculate the error. The error was approximated by taking the distance between the robot at the end of each move command and the point it was moving toward. The 20calls for the shortest perpendicular distance from the intended path. The error shown is a result of undershoot or overshoot. Overshoot or undershoot of points is not part of the 20used was significantly easier to compute and it still showed the performance of the robot. As the robot moved further along the path, it had no trouble

keeping within the specification. Most of the time, the robot was within 10% of the path length.

The robot remained clear of the obstacles except for clipping the corner of the second obstacle while making a turn. This result is very acceptable. The limitations of an encoder system such as this is widely known. The robot is "dead reckoning" as it moves and has no way to update its location. Inevitably, the wheels slip along the ground (since they are far from the idealized point contact wheels off of which the localizer calculations are based) and cause the calculated position to be in error. As the robot moves by the second obstacle, it "thought" it was clearing the obstacle as indicated by the red line in Figure 4.3. However, without a more sophisticated localization system to update the position, it is never able to correct for the error in the encoders. The following sections discuss the results of tests performed on individual parts of the design.

4.2 Odometry System

The tests of the odometry system were successful. The following results for the prototype and final design show that the robot works within specifications using both versions of the odometry system. Although both work very well outside, only the dual encoder design is within specifications when tested inside.

4.2.1 Digital Compass and Optical Encoder

The first iteration, which used a digital compass and encoder, was successful when tested outside. As seen in Figure 4.1, the robot was very close to the desired path the entire time of the test. This result meets the specification since the final position is within the circle on the graph. The final position is approximately 100 ticks away from the desired final position, which is much less than the maximum allowable error of 180 ticks. This result is very acceptable considering the following various errors. First, there is inherent error in the hand measured positions of approximately ± 1 inch. Additionally, there is the

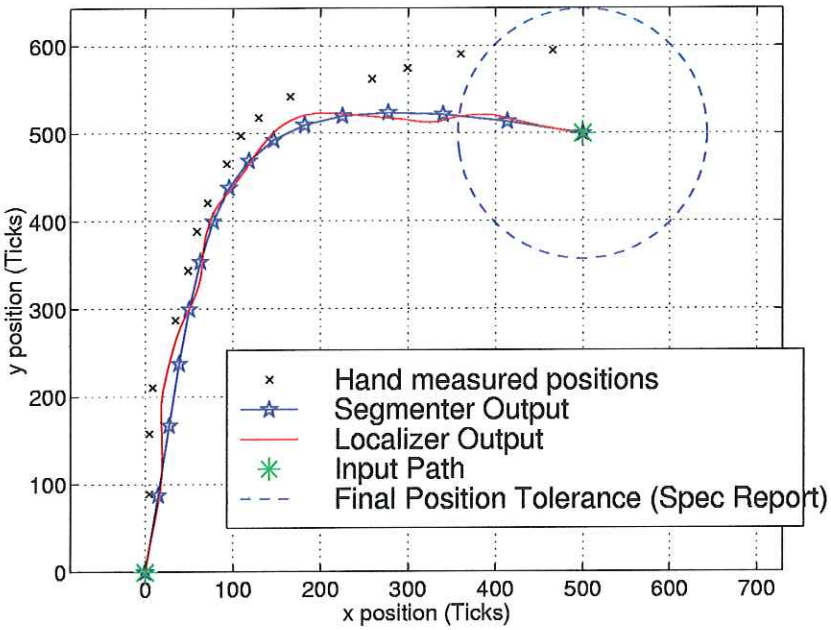

Figure 4.1: Results of overall test using compass

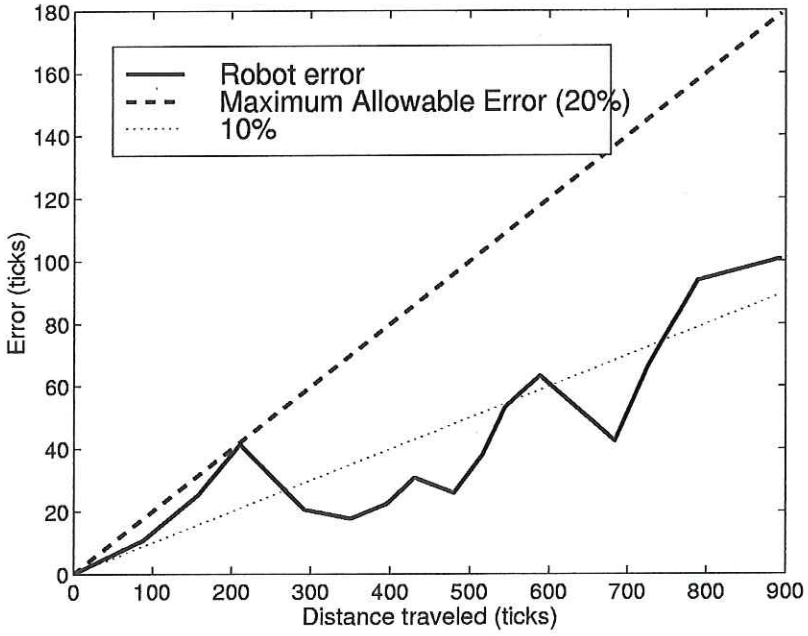

Figure 4.2: Comparison of actual error to maximum allowable error for compass/encoder

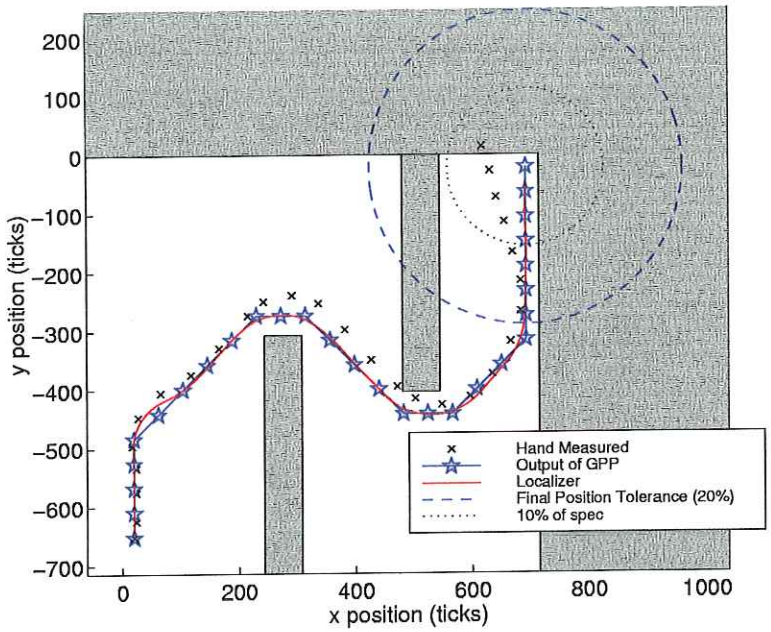

Figure 4.3: Result of final test using dual encoder system

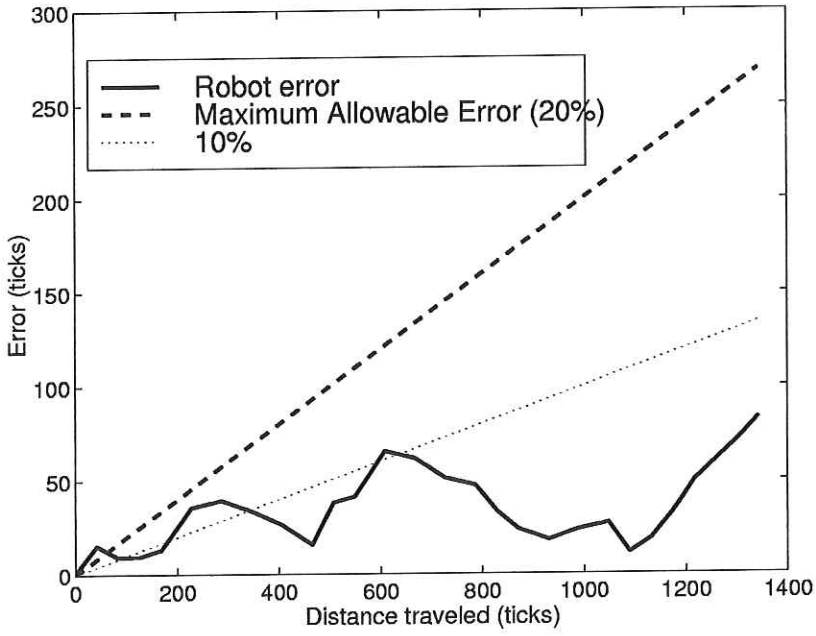

Figure 4.4: Distance of actual path from desired path in final test

uncertainty in the orientation of the entire coordinate system that was drawn out on the ground. The uncertainty of the orientation of the coordinate system was on the order of $\pm 2°$. Although this may seem to be a small error in orientation, it can cause a large error in final position measurements. The final position error due to orientation error can be estimated by multiplying the length of the straight line distance to the final position by the error in orientation (in radians.) For this test, the straight line distance was $\sqrt{500^2 + 500^2} = 707.1$ ticks. Given the uncertainty of orientation of 2°, this results in a position error of about 25 ticks. However, considering the error in the compass, this error grows to 148 ticks. When analyzed further, the robot was completely within the 20% error mark for the entire path, which can be seen in Figure 4.2. Although it comes very close to crossing the maximum error line at the third point, Figure 4.1 shows that it didn't go far enough to the third point, but it is still following the correct path. (See the discussion of the error metric in Section 4.1.)

4.2.2 Dual Encoders

The second iteration of the odometry system, which consists of dual encoders, was also successful. Figure 4.5 shows that it was close to the desired path for the entire test. This result also met the specification. The final position of the robot in this test was also approximately 100 ticks away from the desired final position. This was also much less than the maximum allowable error of 180 ticks. This test also had some error in it, including the hand measured positions and the uncertainty in the coordinate system. Upon closer inspection, the robot was well within the specified 20% error for the entire path. This can be seen in Figure 4.6. The dual encoder system completely eliminates the dependency of the compass, which allows $\mathcal{MoRoX}$ to be operated in noisy magnetic environments such as indoors Moody Engineering Building. For this reason, the second iteration of the odometry system remains on the robot.

4.3 Global Path Planner (GPP)

Satisfactory performance for the GPP was achieved. Figures 4.7 and 4.8 show the results of the tests discussed in Section 3.3. Test #1 was successful using a 17 by 17 grid. The planning took less than a minute to compute. The path avoids all obstacles and is therefore successful in that respect. Additionally, the path appears to be nonholonomic, although there are a few strange jumps, especially in the middle of the two obstacles. This is likely because the rotation must be divided into discrete steps. For example, if the angular rotation is divided into 36 elements (as done in this case) that implies 10 degrees between each cell. There would be a layer with the robot heading 0°, 10°, etc. If the resulting path from the planner indicates straight ahead at a 5^o heading, the path on paper will appear disjointed, much like it does in Figure 4.7.

Test #2 was successful after the cell size was decreased. It failed with the initial grid size of 17 by 17. It was increased to 30 by 30 and completed successfully in 149 seconds.[1] It was possible to use the same maximum turning angle in determining the adjacencies as in the first test of 6°.

4.4 Segmenter

To verify that the Segmenter functions properly, the test plan from Section 3.4 is used and the values compared to the expected values. The first simulation is a constant arc with the starting point given to be (0, 0, 0) and the goal at (5, 5, $\pi/2$). When this test is run the Segmenter produces the curve in Figure 4.9. This corresponds to the expected output from the Segmenter.

The next simulation for the Segmenter is the parallel parking path. In this simulation the starting and goal y and θ positions are the same but the x value has been shifted over, it is assumed that the prototype will only travel in the forward direction. For the simulation

[1]This process was run on `moody`, a computer running Linux 2.2, with a P II 400 MHz processor, and 256 MB of memory.

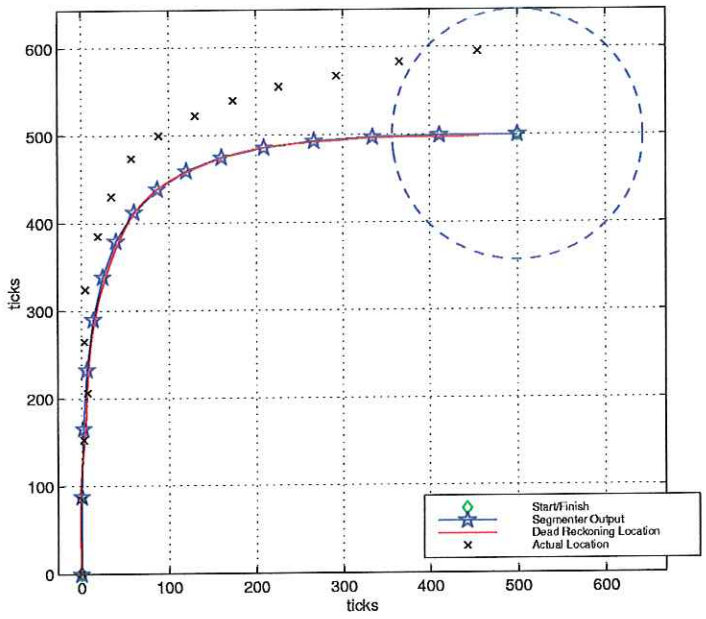

Figure 4.5: Second Iteration Test Results

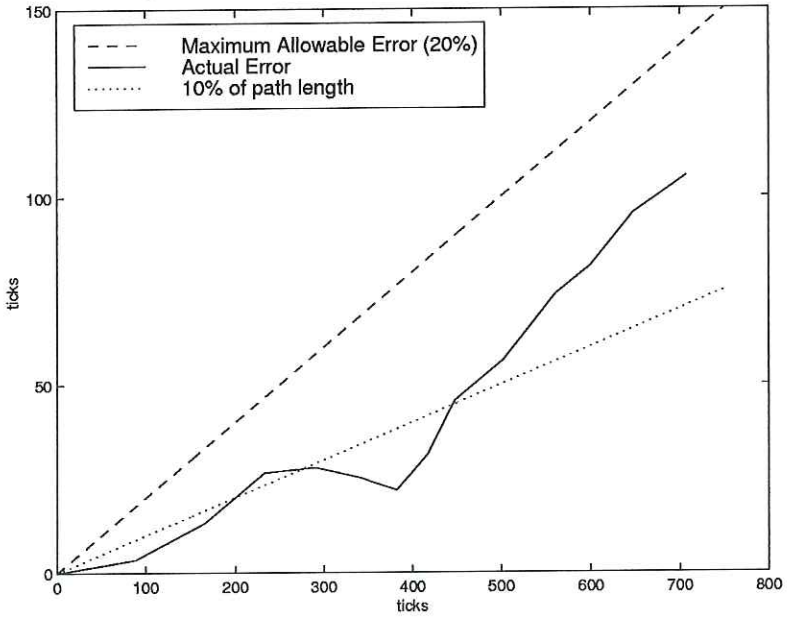

Figure 4.6: Second Iteration Error

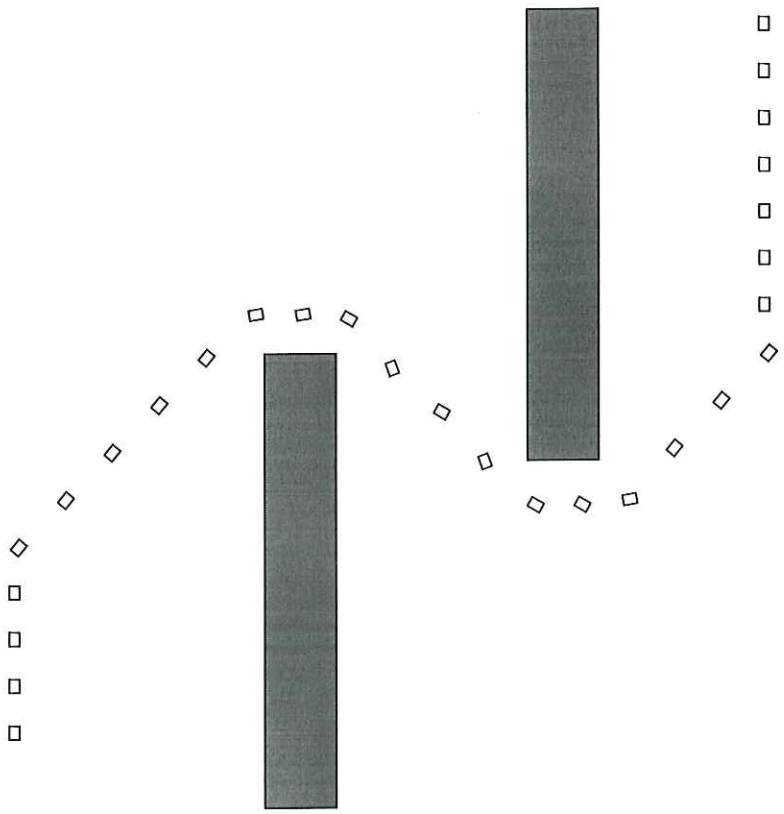

Figure 4.7: Results for test #1 of GPP

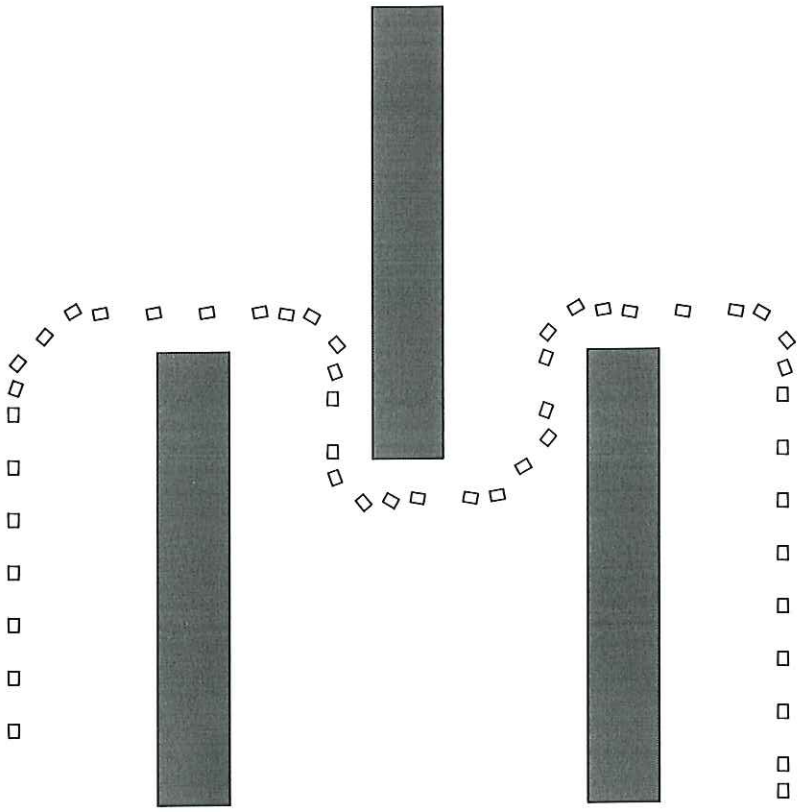

Figure 4.8: Results for test #2 of GPP

in Figure 4.11, the same points will be used as in the first simulation but the angles will be changed to $\pi/2$. This figure matches the expected output from the Segmenter.

The third simulation will test the performance for two points close to each other (one distance unit or less) but at different angles of orientation. For this simulation shown in Figure 4.12, the points (0, 0, $\pi/2$) and (1, 0.5, $\pi/4$) were used. The figure corresponds to what was expected from the Segmenter.

Simulations for failure plots are worthwhile to perform. The reason for the failure are most likely due to the position of the control points defined by L. By manually varying the value of L the problems of sharp corners and tight radius of curvature can be avoided but L cannot be too large that it plots a path outside of the workspace.

In the first failure simulation, shown in Figure 4.13, all variables are held constant with the exception of the starting angle of orientation. The first three figures are feasible paths for the prototype but the forth path contains a sharp corner at (3,5) and the prototype would be unable to follow this path.

A plot that involves three points can also produce up to three failures, sharp corners, a tight radius of curvature, or paths outside the boundary. In this simulation shown in Figure 4.14, three points are plotted (0, 0, 0), (-4, 10, $\pi/4$), and (8, 2, $\pi/2$) for $L = 1, 10, 50, 100$. All four of these paths display a tight radius of curvature and the third plot includes a sharp corner.

The number of segments to calculate can also lead to failure. So far all of the simulations were run taking 100 segments. This in a way defines the resolution of the plot. Three points were used (0, 0, π), (1, 1, $\pi/2$), and (2, 2, π) and all points and L were held constant but the number of segments varied from 100 to 2 in Figure 4.15. As the number of segments gets smaller the path becomes jagged and is less probable that the prototype can follow the path.

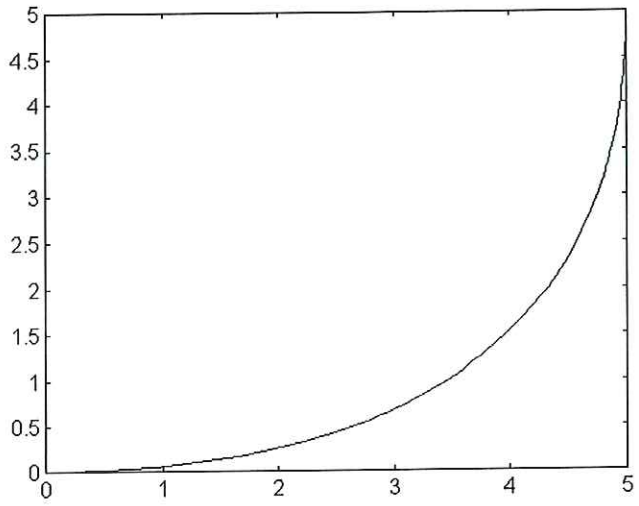

Figure 4.9: Constant radius of curvature.

4.5 Local Path Planner (LPP)

The LPP was tested as stated in Section 3.5. It worked well for many combinations of possible points. The results can be seen in Table 4.1. The turning radius is positive for right turns and negative for left turns.

4.6 Chapter Summary

Shown in this chapter were the results of the tests described in Chapter 3. The results generated from each test were compared against the expected results. Overall the experimental results closely resembled the expected results. In the cases where the results differed, the test were examined to determine the reasons behind the non-similarity and to develop a method to fix the problem. Those comparisons allowed for an evaluation of the robot components, and the robot as a whole, based on criteria specific to each test. The design team produced results early in the project that were used to better the final design and meet all project objectives and specifications.

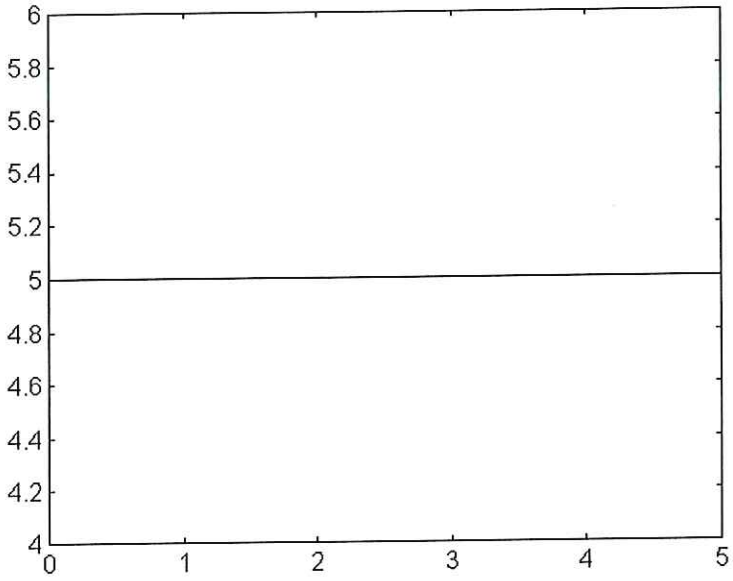

Figure 4.10: Straight line path.

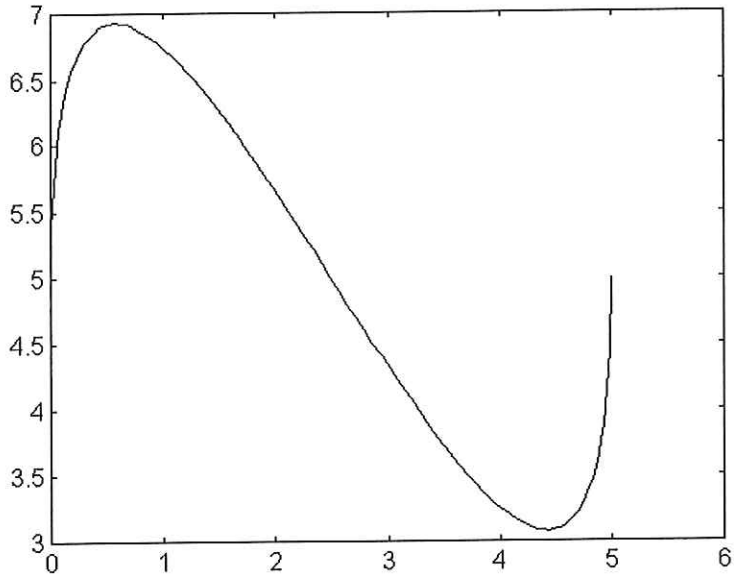

Figure 4.11: Parallel parking path.

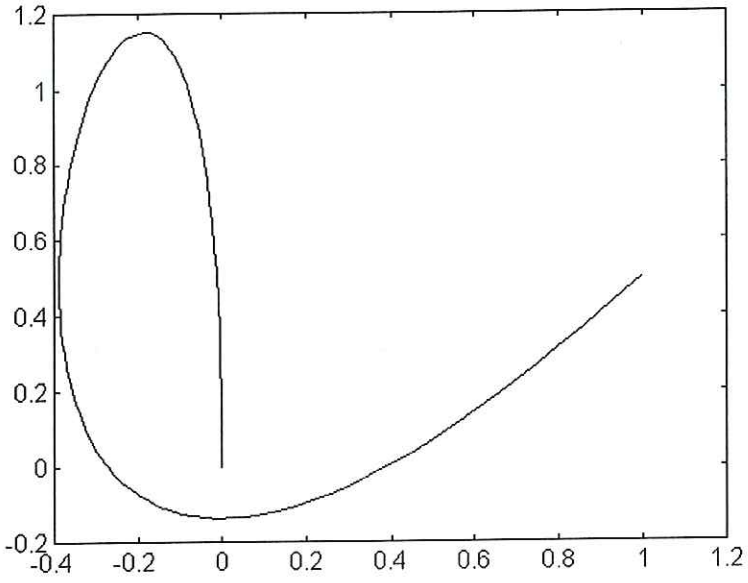

Figure 4.12: Two close points.

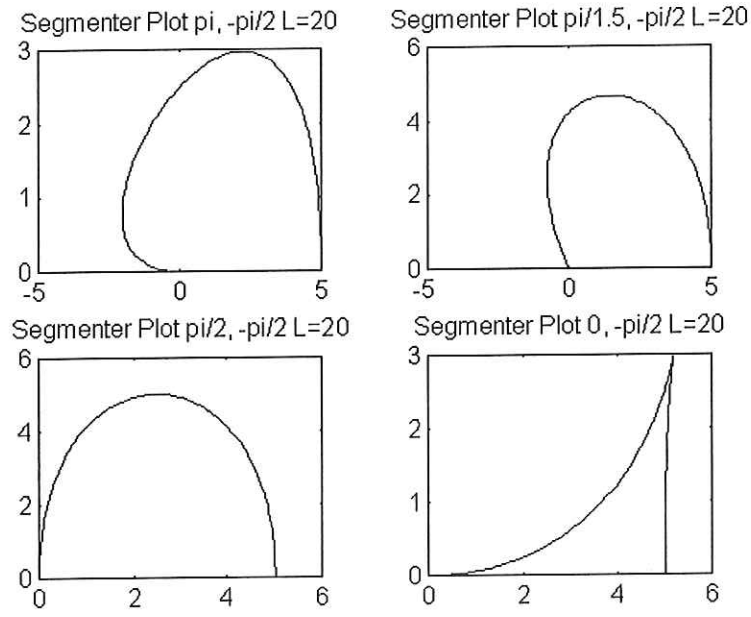

Figure 4.13: Segmenter failure simulation 1.

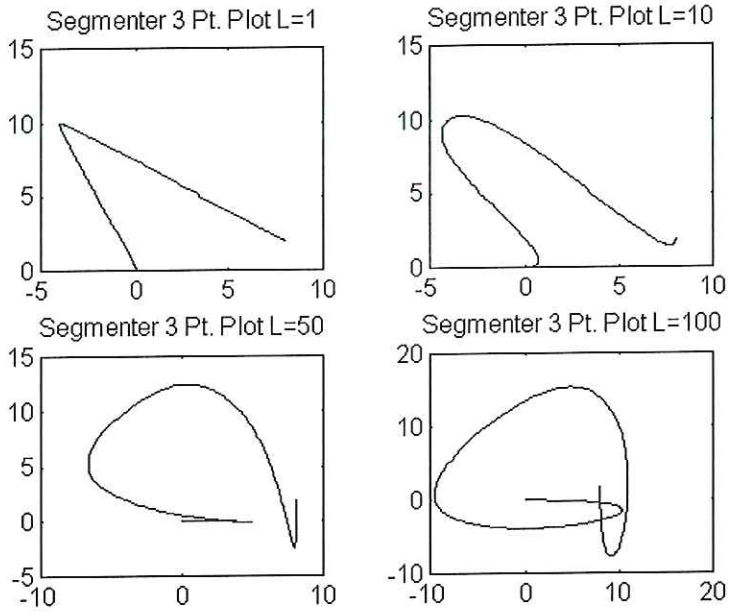

Figure 4.14: Segmenter failure simulation 2.

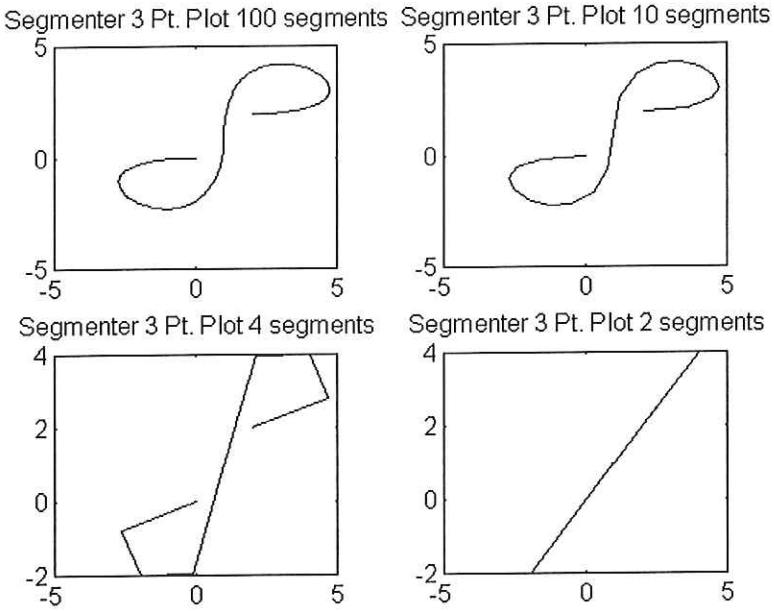

Figure 4.15: Segmenter failure simulation 3.

Table 4.1: Simulation Results For LPP

Input Conditions		Test Result	
Point 1	Point 2	Radius	Distance
(1,0,$\pi/2$)	(0,1)	-1	1.5708
(1,1,$\pi/2$)	(0,1)	-0.5	1.5708
(0,1,$\pi/2$)	(0,2)	-8.16E15	1
(0,1,$\pi/4$)	(1,0)	0.7071	2.2214
(-1,-1,$\pi/4$)	(1,-0.5)	2.0035	2.1654
(1,-0.5,$3\pi/4$)	(-1,-1)	-1.2021	2.4772
(-1,-1,$3\pi/2$)	(0,-2)	-1	1.5708
(-0.5,-0.5,$5\pi/6$)	(-1,2)	1.6971	2.8843
(-1,1,0)	(1,-0.5)	2.0883	2.6813
(-1,1,0)	(1,-1)	1	3.1416
(-1,-1,0)	(-1,1)	-1	3.1416
(0.5,-0.75,π)	(-0.5,0.25)	1	1.5708
(0.5,-0.5,π)	(-1,-0.25)	4.625	1.5276
(0.5,-0.5,π)	(-10,-0.25)	220.625	10.5
(0.5,-2.5,π)	(-10,-0.25)	25.625	10.819
(2,2,-$\pi/2$)	(0,0)	2	3.1416

Chapter 5

Project Summary and Recommendations

This report describes the $\mathcal{M}o\Re o\aleph$ project. Included in the report are descriptions of the system operation, both overall and component wise. Also, tests of the entire system and individual components are presented, along with the test results. The robot produces results that match the expected values for each test. Each component is designed and tested separately first, then incorporated into the overall system. A prototype design involves an initial attempt at solving the path planning and following problems, and improvements to this prototype design comprise the final robot design. The report also includes a budget summary of both the time and parts used during the creation of the robot. A user's manual is included in the appendix that describes the construction and operation of the $\mathcal{M}o\Re o\aleph$.

The project is a success for a number of reasons. One is that the goals established in the requirements analysis phase of the project are met. These specifications include that the robot takes two points in the work space as input and plans a path, between those points, in the free space. The robot follows the planned path, avoiding all obstacles, to within 20% of the total distance traveled. The robot shows that it is capable of planning a wide variety of paths in the free space and following them within the given allowable deviation.

The next phase of $\mathcal{M}o\Re o\aleph$ development could include several upgrades and features that

will make the $\mathcal{MoRoN}$ system more aesthetically and functionally mature. The laptop and Handyboard could be replaced with an embedded system architecture that would reduce the overall size and weight of the robot and make it easier to fit within a shell (covering). However, if an embedded system is used, a user interface would have to be remotely located. A digital camera installation was attempted earlier in the project, however excessive hardware and software conflicts proved this task too difficult to accomplish within the time allotted for the project. A camera mounted on the robot would allow the user to control the robot from any computer with access to the the Trinity LAN. Since the $\mathcal{MoRoN}$ system already contains wireless Ethernet capabilities, a camera would greatly improve the overall usefulness of the robot. Final suggestions for improvements to $\mathcal{MoRoN}$ include modifying the robot so that it can successfully navigate more complicated paths.

Bibliography

[1] Edward Angel. *Interactive Computer Graphics.* Addison-Wesley, Reasing, Massachusetts, 1997.

This book list various types of interative computer graphics. It was used in the development of the spline function.

[2] Richard Bartels. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann Publishers Inc., Los Altos, CA, 1987.

This gives the basic of using splines in computer graphics. It was used in the development of the spline function.

[3] Richard Bruden. *Numerical Analysis.* PWD-KENT Publishing Company, Boston, MA, 1989.

This book provides a theory for spline functions.

[4] Paul Dierckx. *Curve and Surface Fitting with Splines.* Clarendon Press, Oxford, 1993.

This book details the theory of splines and how to fit them to various curves. It was used in the development of the Segmenter.

[5] J. Foley and A Van Dam. *Fundamentals fo Interactive Computer Graphics.* Addison-Wesley Publishing Co., Reading, MA, 1982.

This book gives a background on spline implimentation in computer graphics. It was used in the development of the Segmenter.

[6] SG Hoggar. *Mathematics for Computer Graphics.* Cambridge University Press, Great Britain, 1992.

This books gives formulas for spline creation. It was used in the development of the Segmenter.

[7] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991.

This book contains the information needed to understand the concept of non-holonomic path planning.

[8] G.W. Lucas. A tutorial and elementary trajectory model for the differential steering system of robot wheel actuators. http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html, April 1 2000.

This site has information that was used for the dual encoder implimentation.

[9] Fred G. Martin. The handy board technical reference. MIT Handyboard Webpage, http://mevard.www.media.mit.edu/groups/el/projects/handy-board/techdocs/index.html, February 7 1999.

This has all of the basic operations of the Handyboard including schematics, pinouts, basic IC help, etc.

[10] Martin Schultz. *Spline Analysis.* Princeton Hall Inc., Englewood Cliffs, NJ, 1973.

This book was used as a spline reference manual for generating different types of spline functions.

[11] M. Sutton, D. Houy, A. Leininger, and J. Liddle. Source code for mobile robotic planning and navigation. Source listing of path planner, laptop executable, and Handyboard programs.

[12] M. Sutton, D. Houy, A. Leininger, and J. Liddle. Alternative solutions for mobile robotic path planning. Second milestone paper for project, October 2000.

This paper dicusses different path planning software and hardware methods.

[13] M. Sutton, D. Houy, A. Leininger, and J. Liddle. Mobile robotic path planner prototype. Third milestone paper for project, February 2001.

This paper discusses the prototype in detail.

[14] S. Teukolsky and et.al. *Numberical Recipes in C.* Cambridge University Press, USA, 1992.

This book helped with the writing of the spline function code.

Appendix A

Construction Manual

This section explains how to completely assemble the robot. It is assumed that the all of the parts are available, in working condition, and that no part preparation is needed (i.e. holes drilled, wire runs made, etc). All dimensions are given as L x W x H for flat plates and as L x D for all screws, bolts, and tubes. It is recommended that the parts are present when using the construction manual. (This aids in the visualization of $\mathcal{M}o\Re o\aleph$ construction.)

A.1 Modify Radio Controlled Car

The robot is built on top of a R/C car such as the Traxxas Stampede or similar model. The R/C kit used for $\mathcal{M}o\Re o\aleph$ has a very springy suspension. The robot needs a stable platform to mount the laptop and other components. Therefore, the suspension springs need to be replaced with four (4) aluminum tubes, 2" x $\frac{3}{4}$". These tubes need to be placed over the shocks to prevent any travel in the suspension pieces.

A.2 Speed Controller

Instead of using the factory supplied servo connected to a potentiometer to adjust the speed of the motor, an electronic speed control is used[1]. The speed controller in Figure A.1 is a Duratax - Spike Electric Speed Controller #1040. It is a small rectangular shaped piece, 2.5" x 1" x 1" with four sets of wires. Use the following steps to install the speed control:

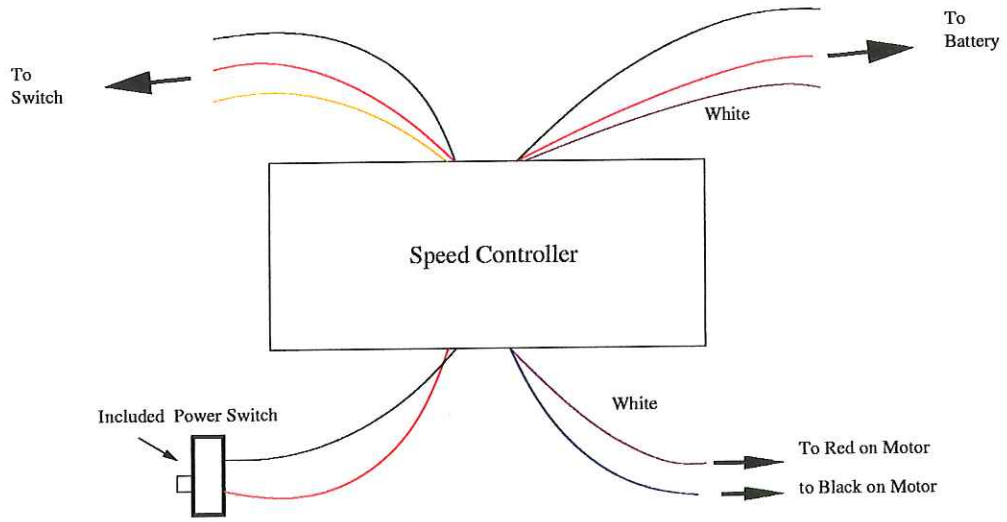

Figure A.1: Speed Controller

1. Remove the factory supplied motor servo from its location; place the speed control in its place.

2. Attach the speed controller permanently to the frame with an adhesive.

3. Connect the the black, red, and white wires to the battery connector of the R/C car.

4. Connect the single white wire to the red wire from the motor.

5. Connect the single blue wire to the black wire from the motor.

6. The On/Off switch is placed on the rear of the motor casing with adhesive.

[1]The speed controller is used for better control of the robot and gives better test results.

7. The black, red, and yellow wires will be connected to the manual switch described in Section A.5.

A.3 Mounting Bracket for Plexiglas Platform

The platform mounting bracket holds and supports the Plexiglas platform. For this part of the construction the following items are needed:

- 2 aluminum plates, 6" x 3.5" x $\frac{1}{8}$"
- 4 bolts, 5" x $\frac{1}{4}$"
- 12 hex nuts, $\frac{1}{4}$"
- 4 flat washers $\frac{1}{4}$"
- Adjustable wrench

Once these items are assembled the following steps will explain how to build the mounting bracket. Refer to Figure A.2 for a depiction of the mounting bracket.

1. Place one plate on the top and one plate on the bottom of the R/C car frame in the center, where the battery is stored, so that the 6" length of the plates lie perpendicular to the frame (and parallel to the axles).
2. Thread the bolts through each of the four holes on both plates.
3. Attach a nut to the bottom of each bolt, so that the nut is on the outside of the bottom aluminum plate. Be sure and leave only a small part of the bolt showing on the bottom. The extra length will be needed for the top part of the platform.
4. Attach a nut on the outside of the top plate and tighten with a wrench. The two plates should now be firmly attached to the car frame.

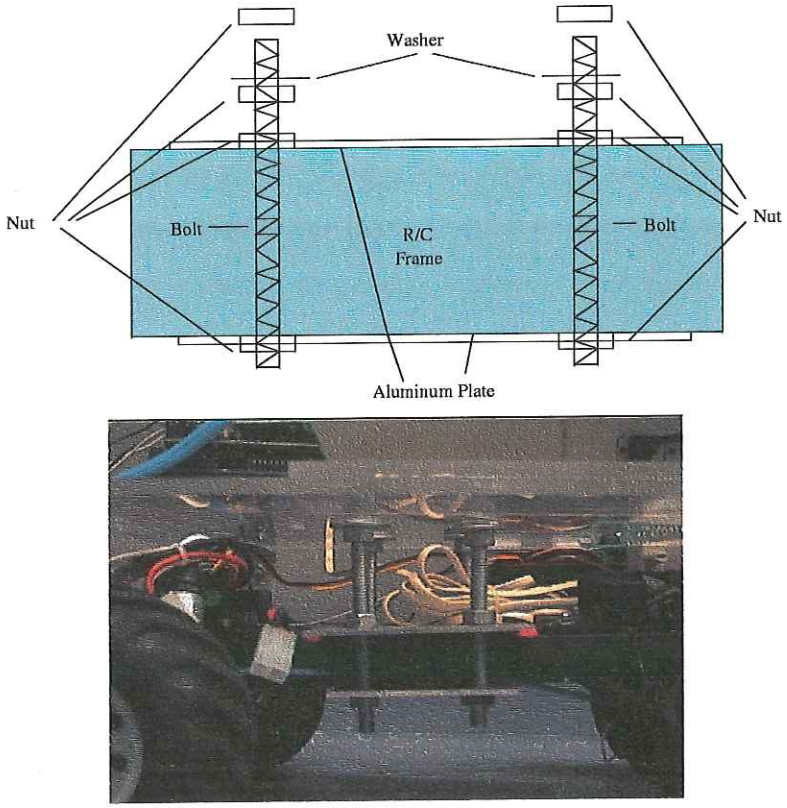

Figure A.2: Mounting Bracket Side View

5. Thread a nut down each bolt approximately 1 inch from the top.

6. Place a washer on top of each nut.

A.4 Platform

The platform in Figure A.3, supports the laptop and the Handyboard. It is a piece of Plexiglas 13.25" x 12.5" x $\frac{1}{4}$". There are also two spacers attached to give the laptop an even surface to sit on. Four $\frac{1}{4}$" nuts will be needed for this step.

1. Place the bolts from the mounting bracket through the 4 holes in the center of the platform. Make sure that the spacers are facing up and that the platform is oriented correctly.

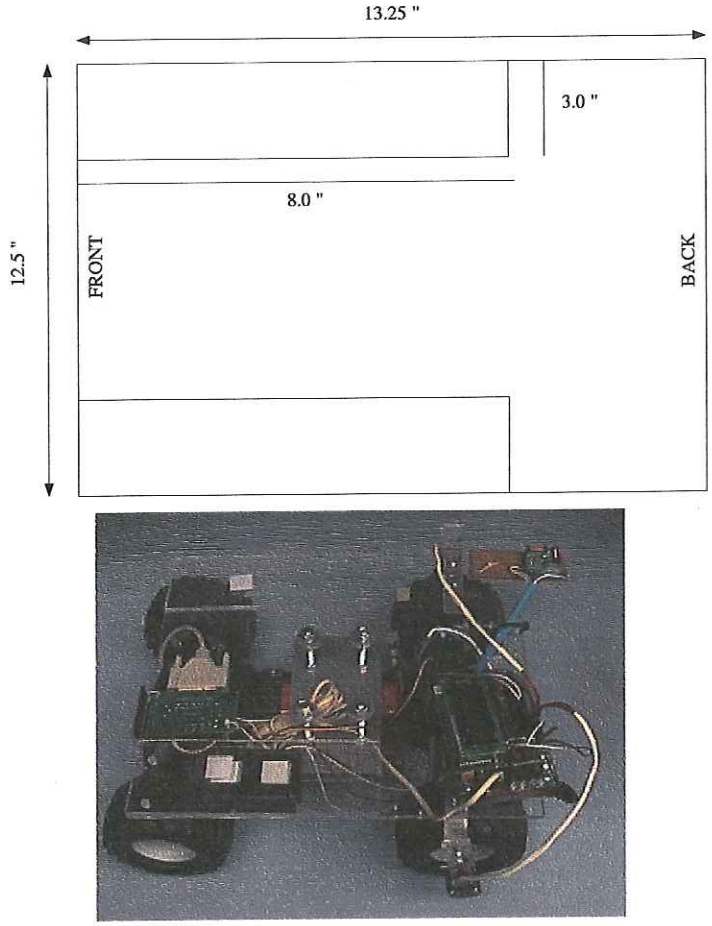

Figure A.3: Robot Platform

2. Use the nuts to fasten the platform with a wrench on the mounting bracket. There should be just enough of the bolt sticking out for the nut to fit over.

A.5 Manual Switch

The manual switch, shown in Figure A.4, allows the user to change between joystick control and autonomous control. To attach the switch to the robot the following items are needed:

- Switch circuit (pre-manufactured double pull, single throw switch; with jumper wire connections from switch to input and output leads)
- 2 wire runs with a 6 pin connector at one end and two 3 pin connectors at the other

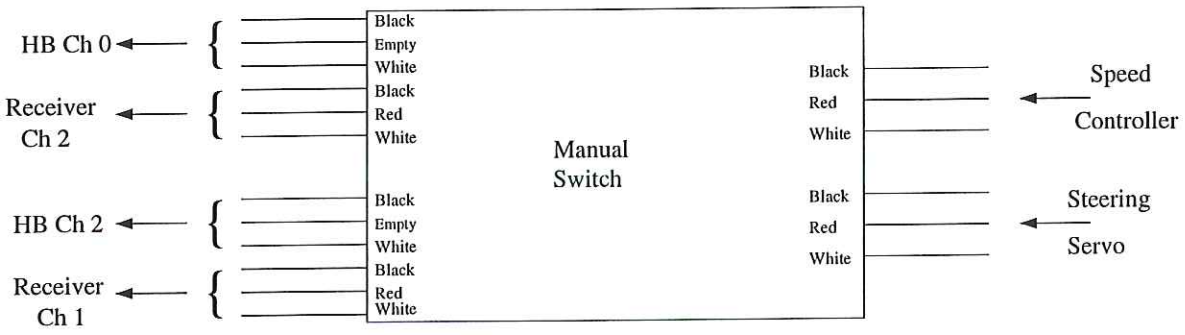

Figure A.4: Manual Switch Pin Connections

end.

- Black plastic box with metal top3" x 2" x 1"
- Rubber grommet

The wire connectors from the speed controller and the steering servo will also be needed. In addition, it may be easier to remove the platform while connecting the switch.

1. Looking from the top, orient the switch circuit as shown in the figure with the 6 pin leads on the left and the 3 pin leads on the right.
2. Locate the wire run with the white dot on the 6 pin connector.
3. Place this wire run on the top 6 pin lead set. Make sure that the wire colors correspond.
4. Place the other wire run on the bottom 6 pin lead set, being careful to match the wire colors correctly.
5. Take the 3 pin connectors and run them from the inside of the box through the hole and out.
6. Place the switch circuit inside the box.
7. Attach the grommet around the hole.
8. Take the wire connector from the speed controller through the hole to the switch.

9. Connect it to the top 3 pin lead set, matching the wire colors.
10. Take the wire connector from the steering servo through the hole to the switch.
11. Connect it to the bottom 3 pin lead set, matching the wire colors.
12. Place the metal cover on the box and tighten with washer and nut from switch.
13. Find the 3 pin connector with the white dot and 3 wires (black, red, white).
14. Connect it to Channel 2 on the receiver with the white wire on top.
15. Fine the other 3 pin connector with the 3 wires, (black, red, white).
16. Connect it to Channel 1 on the receiver with the white wire on top.
17. The other two wire connectors will be attached to the Handyboard once it is installed.
18. Mount the switch on the left, bottom side of the platform with .

A.6 Serial Communicator

The serial communicator is a circuit board with RS-232 and RJ-45 connectors.

1. Mount the serial communicator in the front, center of the platform on the bottom with zip ties. Make sure that the RS-232 connector is facing to the left, when the robot is looked at from the front.
2. Connect the RS-232 cable to the serial communicator.
3. Connect the RJ-45 cable to the serial communicator.
4. The connections for the other ends are discussed in Sections A.8 and A.9.

A.7 Encoders

The encoders are two small boxes with a metal shaft and pin leads on the top. They are mounted to the rear wheels. In order to mount them the following items are needed:

- 2 US Digital E2-50-250 encoders
- 8 bolts, 1" x $\frac{1}{8}$"
- 8 nuts, $\frac{1}{8}$"
- 2 aluminum 90^o brackets
- 2 Plexiglas frame strips, 6" x 1" x $\frac{1}{4}$"
- 2 rubber tubing 2" x $\frac{1}{4}$"
- length of black wire
- 2 Shrink wrap wire run with orange, red, blue, and black wire.

Refer to Figure A.5 for construction help as needed.

1. Thread wire from rear wheel spokes through small holes in rubber tubing [2].
2. Place tubing around wheel nut and secure by tightening wire.
3. Bolt 90^o bracket to back, bottom side of platform and secure with nuts.
4. Attach Plexiglas strip frames to outside of brackets with bolts and nuts. Making sure the large hole is pointing down and even with the center of the wheel.
5. Place shaft of encoder though frame hole and into tubing.
6. Use an epoxy to secure encoder onto Plexiglas.

Now that the encoders are mounted, the pins need to be connected. Figure A.6 shows the pin outs and wire connections for the encoder.

1. Connect the wires of the wire run to the appropriate pin as shown in the figure.
2. Make sure that the pin connector is pushed down all the way to avoid any loose connections.
3. Once the Handyboard is mounted the other end will connect to an I/O port.

A.8 Laptop Mounting

The laptop is the main component that sits on the platform.

1. Locate the laptop, a Dell Latitude XP [3].
2. Place laptop on top, front of platform with the screen facing the back of the platform using the velcro.
3. Check to make sure that the laptop is resting on the spacers evenly and that no part of the bolts are in the way.
4. Connect the RS-232 cable from the serial communicator to the serial port in the back of the laptop.

A.9 Handyboard Mounting

The Handyboard is the last item to be attached to the robot. A general picture of the Handyboard is shown in Figure A.7.

[2]This is to hold the tubing in place so that it turns with the wheel.

[3]A similar laptop can be used as long as there is an RS-232 port, Linux, and network capabilities. For laptop specifications see Appendix F.

1. Make sure the expansion board and LCD display are connected to the Handyboard.
2. Mount the Handyboard in the back, top, left corner of the platform with velcro.
3. Connect the RJ-45 cable from the serial communicator to the RJ-45 port on the Handyboard.
4. Connect the left encoder wire run into the digital I/O port 8 as shown in Figure A.8.
5. Connect the right encoder wire run into the digital I/O port 7 the same way as the previous encoder.
6. Find the 3 pin connector with the white dot from the manual switch.
7. Connect it into SERVO pin 0 on the expansion board as shown in Figure A.9.
8. Take the other 3 pin connector from the manual switch and connect it into SERVO pin 2 on the expansion board the same way as before.

A.10 Final Comments

The robot is now constructed and ready for operation. Be sure to double check all connections, both mechanical and electrical, before proceeding to the operation section of the user's manual. All parts should now be used. If there are parts left over, go back through the construction procedure and locate where that part is used.

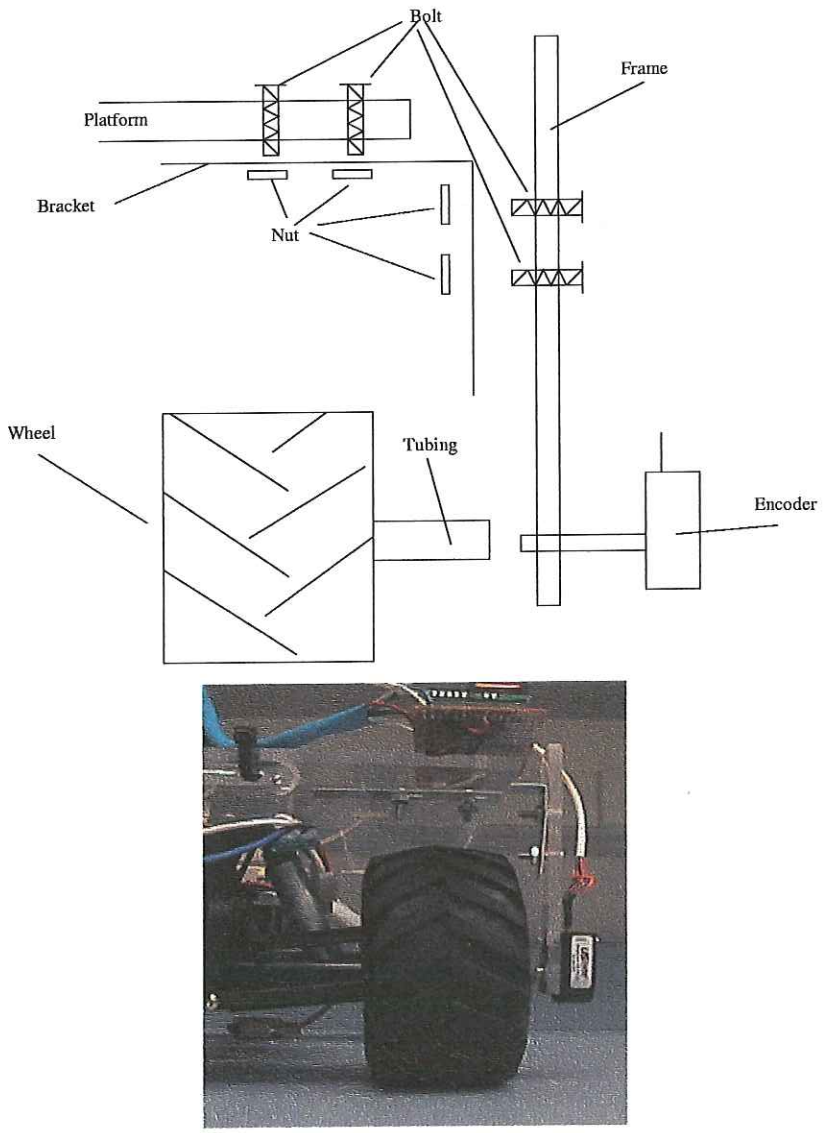

Figure A.5: Encoder Mounting Diagram from back right rear

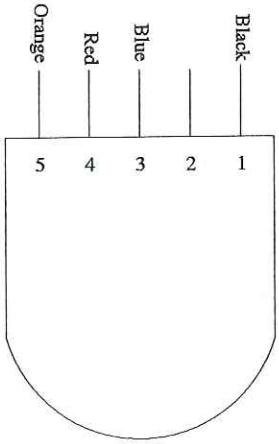

Figure A.6: Encoder Pin Connection Diagram

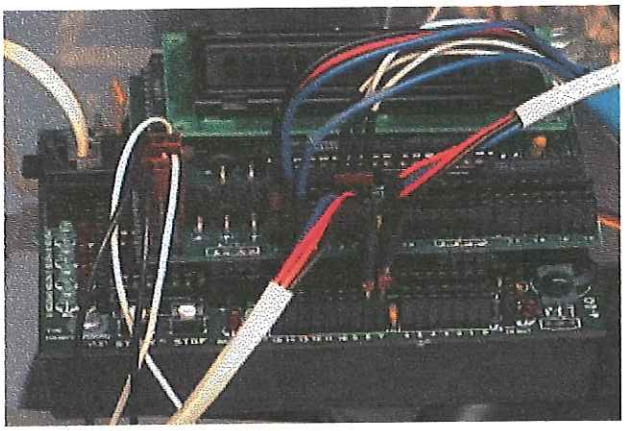

Figure A.7: Picture of the Handyboard with connections

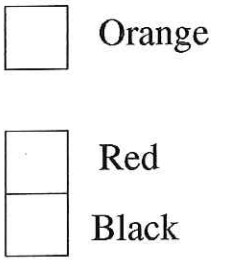

Figure A.8: Digital I/O Pin Connection on Handyboard

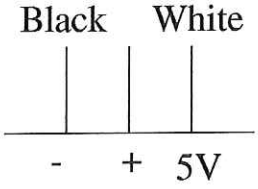

Figure A.9: Servo Pin Connection on Handyboard

Appendix B

Operation Manual

This manual outlines how to operate the robot. This is an experimental platform and has not been refined for the use of the general public.

B.1 Getting Started

Several components need to be working to get the robot working. First the Handyboard must have its operating system up and running. The OS tends to disappear in a relatively short amount of time when the battery charger is not connected to the battery. Page 4 of the Handyboard manual [9] discusses this how to load the OS onto the Handyboard. It involves using some type of a downloader to transfer the `pcode_hb.s19` file (the Handyboard's OS) to the Handyboard. If using Linux, both a downloader and a version of interactive-C (the Handyboard's on board C language environment) exist. However, a working binary of interactive-C was not found and it was necessary to build the binary from its source form. The downloader and the source for interactive-C were found on the M.I.T. Handyboard Web site (`http://www.handyboard.com`). The downloader is on the MoRoN project CD, in `/ic-src-2.860-beta/dl/` .

Once the Handyboard is up and running, it is necessary to install (download) several files on the Handyboard. This process is accomplished by use of the interactive-C program (gener-

ally invoked by typing `ic` from a shell). Interactive-C allows the user to connect to the Handyboard to load and execute code[1]. The following files must be loaded onto the Handyboard: `lib_hb.c`, `lib_hb.icb`, `libexpbd.idb`, `expsens.c` `expservo.icb`, `explego.icb` (these are all files that come with the Handyboard or the Handyboard expansion board) `quad.icb`, `quad.c`, `serial_isr.icb` and `serial_isr.c` (these are specific to the robot project[2]). Once these files are loaded, the Handyboard is ready to go. Simply reset it by cycling the power. At this point, the display on the Handyboard should read, "Initialized... Hit Start to Continue." After hitting the start button, the Handyboard is ready to receive commands from the laptop.

B.2 Path Planner

The path planner is a self contained process; it is not directly integrated into the laptop executable[3]. The primary executable is `search`. It reads the initial and goal position from the text file `position-info.dat`. This file must contain six numbers. The first three represent the x, y, θ in units of ticks of the initial position and the next three represent the final position. The map is read from the xfig (developed with xfig 3.2.3 patchlevel beta-1) file `figs/input.fig`. The software is designed to use a 5000 by 5000 unit working area. If the size of the working area needs to be changed, code must be modified in `PathPlanner.cpp` in the generateAdjNonHol() function. (See the code for further information.) When reading the input fig file, the search program only pays attention to polygons. Obstacles are denoted by polygons that are filled. The fill settings are Fill Color set to White, Fill Style set to Filled and Fill Intensity set to 50%. The robot must also be drawn in the fig file. The location of the robot has no significance, the program only needs to determine the relative size of the robot with respect to the obstacles. The program assumes the robot is oriented with the

[1] Although in general it is possible to execute code from within interactive-C, this is not possible with the robot because the Handyboard's serial link is used with the laptop for other communication purposes.

[2] All of the source code is included in a separate document [11].

[3] It would not be difficult to incorporate the path planner into the laptop executable. It was not incorporated because the laptop being used for the robot has a slow processor.

front pointing toward the right of the page. The robot is denoted by a polygon with the fill settings of: Fill Color set to White, Fill Style set to Filled, and Fill intensity set to 100%.

The path planner is run by simply typing `search` at the command prompt. The program will then inform the user when it is complete. If it was unable to find a path, the user will be informed. If it was successful in generating a path, the path is written to the file `out-pts.dat`. Additionally, the path is written to a fig file `figs/testout.fig`. This fig file shows the path as well as the obstacles.

B.3 Laptop executable

The main executable is `laptop`. This program reads in the input path from the file `input-path.dat`. However, it can easily be configured to read an input path from a fig file (code to do this has been written but is currently commented out). The final implementation by default does not use the Segmenter. `laptop` must be run with the `-s` argument for the Segmenter to be used.

Operation procedure:

1. Verify all connections as discussed in the construction manual (Appendix A).
2. Make sure the desired path is in the file `input-path.dat`.
3. Make sure the Handyboard is up and running as discussed in Section B.1. The display on the Handyboard should read "Ok."
4. Power the robot on.
5. Type `laptop [options]`.
6. As the robot moves, it will write the files: `local.dat`, `point.dat`, and `segmenter.dat` . `local.dat` is the output from the Localizer, `point.dat` contains the points read from `input-path.dat` , and `segmenter.dat` contains the points generated by the Segmenter (if applicable).

`[options]` : There are only two options that can currently be passed to the laptop executable. They are "-p" and "-s". The "-p" option causes the robot to stop and wait for a key to be pressed before issuing the next move command. The "-s" command forces the Segmenter to be used, otherwise, the Segmenter it *not* used.

Appendix C

Group Member Contributions

Each Member on the team contributed considerable time and effort to this project. Overall, the team spent 922.5 hours of time during the year on the project. The number of hours that each person spent are shown in Figure C.1. The average number of hours spent per week was 9.89 by Mike, 8.37 by Jeff, 9.38 by Daniel, and 7.85 by Andrew. A graph of the difference between the expected and actual hours spent per week can be seen in Figure C.2. On average, the entire group was only a half of an hour short of the recommended 36 hours per week.

Since this project was such a massive undertaking, each member worked on different aspects of the project. The following sections describe each person's contribution to the project.

C.1 Daniel Houy

Daniel started out the first semester by being the group leader. As group leader he worked on making sure tasks were assigned and being performed, scheduled regular meetings with the group members as well as with the advisor. During his term as group leader he made sure the Specifications report and presentation were completed on time an gave an account of his accomplishments and project views to the the project advisor. As group leader Daniel

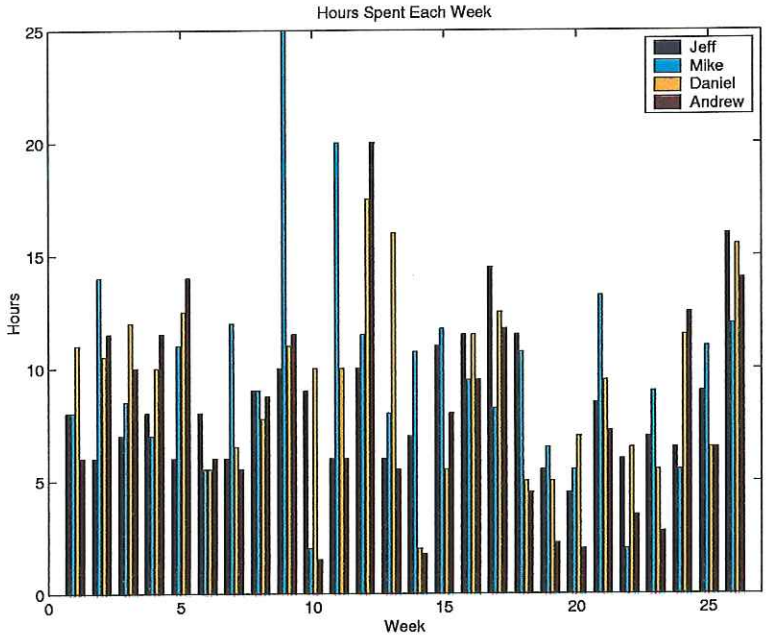

Figure C.1: Hours Spent Each Week for the Entire Year

logged a total of 36.25 hours from September 1-21, 2000.

Next Daniel began researching road maps for the white paper. After the report was finished he began preparing for the white paper presentation that Dr. Nickels had organized. After it was decided what methods to use he began working on the Localizer. For this milestone Daniel spent 28 hours from September 22 through October 18, 2000.

Next he began working on the Segmenter. By the end of this milestone Daniel had a working Segmenter in the most basic form. From October 19 though November 8, 2000 Daniel logged 24.75 hours.

The last milestone was the most intensive for him. Daniel was continuing to work on the Segmenter as well as preparing for the Prototype Design and Test Plan report. For this report he worked on the overall system test plan and the Segmenter test plan. He also ran simulations on the Segmenter and reported those conclusion in the simulation results section. After the report was finished he began working on the presentation. Daniel presented on the prototype test plan and Segmenter results. From November 9 to December 4, 2000 Daniel has spent a total of 44.5 hours.

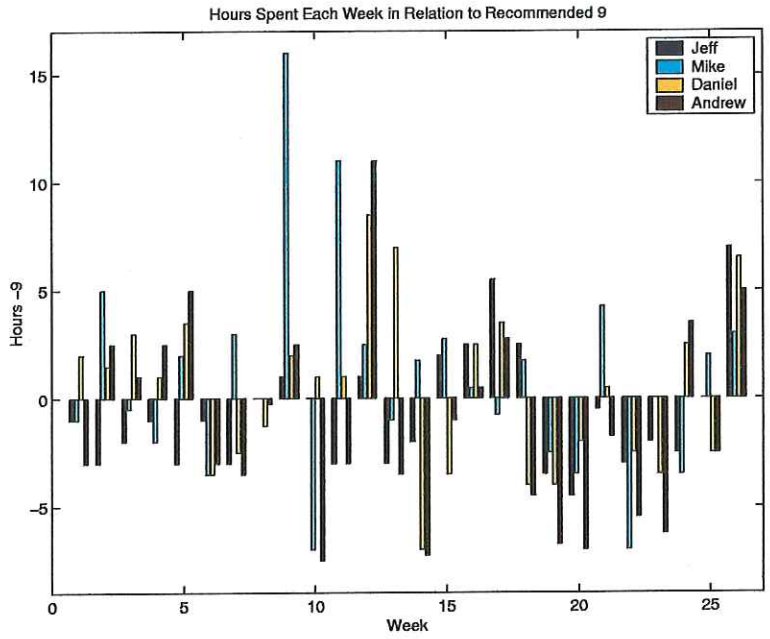

Figure C.2: Hours Above or below 9 hours a Week for the Entire Year

After Christmas break the group had a little over three weeks to get the prototype ready for the presentations and to finish the paper for this milestone. His main focus during this time was helping to get the to the finishing touches on the robot. Daniel fixed the wheel alignment and also made some B-field measurement in Moody 305. The rest of his time was spent writing the report, making corrections to it, and preparing for the presentation. During the month of January Daniel logged a total of 23.5 hours.

In February Daniel continued to work on the paper and the presentation for the first week. After that time his task was the robot clean up. This included taking out all of the old solid core wires that connected different hardware and replace them with braided wire. He also used this time to look into a shell for the robot. For this month Daniel spent a total of 21.0 hours on the project.

Daniel was the group leader for the month of March. He continued to work on the clean up of the robot by building a new manual switch. After the switch was installed and all of the clean up down the group presented a demonstration of the prototype to KLRN TV of San Antonio for use in an upcoming show. Towards the end of the month he started working

on the outline and format for the final paper. In March Daniel worked a total of 25.5 hours on the project.

In April Daniel spent the majority of his time working on the final paper. His main contribution was with construction section of the user's manual. He has also written other various subsections and helped with the editing of the drafts. Daniel worked with Andrew and Mike in the making of the video that will be used for recruitment. For the month of April, he has worked a total of 44.5 hours on the project. For the year Daniel has logged a total of 248.0 hours for senior design.

C.2 Andrew Leininger

Andrew Leininger's initial contributions to the $\mathcal{M}o\Re o\aleph$ project involved path planning research. Following this, he assisted the team in developing the requirements and specifications of the project. He contributed to the creation of the first deliverables, the specifications report and the corresponding presentation. During the alternative solutions research phase of the project, Andrew looked into a fluid dynamics based path planning algorithm, and into an embedded system based robot control hardware. He contributed to the whitepaper deliverable, and the whitepaper presentation. Andrew aided the group in deciding which solutions to use in the project, and then began learning the Handyboard. Once he had a familiarity with the Handyboard, Andrew began leaning about and interfacing the digital compass. This work resulted in a device driver for the compass with simple and efficient interfaces for the higher level software routines. After taking over as group leader, Andrew performed the more administrative oriented tasks involved with the project such as time and task management, advisor liaison, and deliverables responsibility. During this time, the group was creating the $\mathcal{M}o\Re o\aleph$ prototype. Andrew worked mostly on odometry hardware for the prototype. He also worked towards the completion of written and presented deliverables concerning the prototype. Near the close of the semester, Andrew's work involved tying up loose ends and preparations for the coming semester, such as part requisition and task

assignments. The beginning of the spring semester found the group working towards cleaning up and finalizing the prototype. Andrew's contribution to this effort included porting the Segmenter module from its then current Matlab implementation to a C implementation so that it could be merged into the main robot software source tree. As the current group leader, Andrew's administrative and organizational tasks continued. The work on the prototype lead to two more deliverables, a report and presentation on the testing and status of the prototype. Andrew contributed to both of these deliverables. Once the prototype was complete, work on creating the final $\mathcal{M}o\Re o\aleph$ system began. Andrew had many tasks during this phase, including: finding, ordering, and interfacing a video camera for the robot; working with the Handyboard software (6811 assembly) for handling the two encoder system; obtaining and installing a new speed control; and creating a departmental demonstration video of the robot. As a final task for the semester, Andrew has spent his project time working on the creation and perfection of this document. He also contributed to and participated in the presentation of this document and the $\mathcal{M}o\Re o\aleph$ project.

C.3 Jeff Liddle

Jeff worked on many aspects of the robot project during the entire year of Senior Design. The first thing was path planning research for both hardware and software. The next thing that he worked on was a report for the project description and specifications. Along with that report came preparation for the first presentation. After the presentation he looked into fixing the RC receiver that was mysteriously fried. The next step in the project was to work on the white papers, which were a description of alternative solutions. He then worked on editing the white papers and preparing for a presentation. After the solution had been determined, he began working on the code for the local path planner. He also worked on the code for the localizer. During this time, he also replaced the old RC receiver with a new one. After working on the localizer and local path planner for a while, he began work on the prototype report. He also spent time revising the report and preparing for the technical

presentation on the prototype. At the beginning of the next semester it was time to begin testing the components of the prototype while finishing the prototype. He worked on testing the local path planner and realized that it had to be completely redone. Once it was done he ran a thorough test of it. He ran many more tests on the prototype having to do with turning radius to servo measurements, and also ran a test to determine if the encoder was outputting the correct distance. Once these were done, he created graphs of them and began working on the new prototype paper. Along with this paper again came more editing and preparation for a presentation. He then began working on fixing the odometry system of the robot. The new solution was a dual encoder system. So, he implemented the hardware for this solution. He also wrote the software for the new localizer, edited it, and tested it when it was complete. After completing this, he helped to setup and test the final product. Once the final product was tested, he began working on the final report. Once he is done with this report, he will work on editing it and preparing for the final presentation.

C.4 Michael Sutton

Michael Sutton contributed to several aspects of the $\mathcal{MoRoN}$ project. His previous research in robotics was helpful in determining project specifications and looking over potential solutions to the problem. He looked over the hardware from the previous year's design group determining what had already been accomplished and what was reusable (as it turned out, virtually nothing was used except the hardware from the previous year's design group). His previous research experience was most useful in working on the proposed path planning solution, which is the solution that was eventually used. He had already written the basics of the path planner that was used in this project. The majority of the time in the initial stages of the project was spent in researching and writing up papers to meet the project deadlines set by the project administrator. The period lasted up through about the middle of the first semester.

Beyond the initial stage of determining specifications and working on potential solutions,

he moved towards working on the design and implementation of the robot. He worked extensively with the Handyboard determining ways to interface the encoder and the compass. Additionally, he wrote some of the basic software allowing the Handyboard to communicate with the laptop. This provided a platform to build the rest of the design. He then played a significant role it writing software for the Handyboard and the laptop. He also worked on bringing together several pieces of code written by different authors and integrating and debugging them all. Towards the end of the semester, the third paper of the semester was his main focus. The majority of Michael's time during the first semester was spent on writing and editing reports.

The second semester allowed for more time to be spent working on $\mathcal{MoRoN}$. A significant amount of time was spent debugging and working through problems that came up as the robot was constructed. He played a large role in accomplishing a major milestone by meeting all expectations for the prototype. His largest task during the second semester was spent working independently on improving the path planner that he had started the year before.

Again, he played a large role in finishing everything up and working through numerous bugs that developed. Overall, he played a large part to the success of this project.

Appendix D

Time-line

Attached is a Gantt Chart detailing the project tasks as they occurred during the course of the project. This chart shows all the tasks that were performed during the project and each task's duration. Also included on the Gantt Chart are milestones and important dates for the project.

Appendix E

Costs

The team had a budget of $1000 for the entire year. In order to get the robot working, many new items purchase and a few replacement items were purchased. The final budget is shown in Table E.1. The team came in under budget with $103.25 remaining.

Table E.1: Final Budget

Date	Item	Cost	Budget Left
10/26/2000	RC Receiver	$40.99	$959.01
12/4/2000	Encoder	$39.00	$920.01
12/4/2000	Shipping	$7.00	$913.01
12/6/2000	Digital Compass	$50.00	$863.01
2/6/2000	Shipping	$9.14	$853.87
12/6/2000	Servo	$24.00	$829.87
12/6/2000	Servo	$28.00	$801.87
12/6/2000	Speed Controller	$60.00	$741.87
12/6/2000	Shipping	$5.00	$736.87
1/22/2001	AA Batteries	$7.11	$729.76
1/30/2001	Wireless Ethernet Card	$385.00	$344.76
2/23/2001	Video Camera	$40.95	$303.81
2/23/2001	PC Card	$74.95	$228.86
2/23/2001	Rebate	-$10.00	$238.86
2/26/2001	2 Encoders	$78.00	$160.86
2/26/2001	Shipping	$8.00	$152.86
3/20/2001	Speed Controller	$49.61	$103.25

Appendix F

Specifications

Following are the specifications for the components used on MoRoN.

- Traxxas Stampede R/C model[1]
- Traxxas 2015 B.E.C Receiver Digital Proportional Two Channel Radio System[1]
- 6 Cell Nicad Battery[1]
- Dell Latitude XP Laptop[2]
  - Intel 486 DX/4 Processor
  - 40 MB RAM
  - Linux RedHat 6.2 (Kernel 2.2.14-5)
- Three U.S. Digital E2-50-250 digital encoders (50 ticks per revolution)[3]
- Futaba S9402 Steering Servo (111 oz/in of torque)[1]
- Duratrax Spike MOSFET Speed Control[1]
- MIT Handyboard[4]
  - 52-pin Motorola 6811 microprocessor with system clock at 2 MHz
  - 32K of battery-backed CMOS static RAM
  - Internal 9.6v nicad battery
  - Board size of 4.25 x 3.15 inches
  - Gleason Research expansion board

- Vector 2X Electronic Compass made by Precision Navigation (1° resolution)[5]
- Orinoco Wireless Ethernet Kit (PCMCIA card and RG-1000 Wireless Hub)[6]
- Software
  - Interactive-C[4]
  - Handyboard downloader
  - Software written for project. All of the code is printed in a separate document.[11]
- Plexiglas platform - 190 in^2, 13 in by 14.5 in
- Metal Plates - 3.5 in by 6 in
- Generic double pole, single throw switch for manual and autonomous robot control

Where the parts can be obtained:

[1] Most hobby stores would carry this product.

[2] Any type of laptop could perform the basics of this project. It would be suggested not to use a computer much slower than the one used here.

[3] U.S. Digital products can only be purchased directly from U.S. Digital.
http://www.usdigital.com

[4] Gleason Research sells assembled MIT Handyboards as well as the expansion board.
http://www.gleasonresearch.com

[5] The compass can be purchased directly from Precision Navigation.
http://www.precisionnav.com/

[6] This can be purchased at various different places. It was purchased from PC Connection.
http://www.pcconnection.com