
Autonomous field equipment which can successfully replace human operators on tractors and other equipment may revolutionize agriculture. Autonomous machines designed so that one person supervises multiple machines would make it possible to use more machines without increasing the number of human operators. This makes it feasible to use smaller machines, which have benefits for cultivation, safety and the cost of purchasing and maintaining the equipment. The purpose of this research was to develop a prototype autonomous vehicle. It was constructed using a zero turning radius vehicle which navigates with a differential global positioning system (DGPS) receiver and an electronic compass. These sensors were coordinated using microcontrollers as input processors and another microcontroller using the inputs to control the machine. This microcontroller drove electronic actuators installed on the steering and speed control mechanisms of the machine. A control algorithm was implemented by combining the crosstrack and angular deviations from the reference path and applying a control input to the actuators based on that combined or "aggregate" error. The control input was calculated using a the traditional Proportional-Integral-Derivative controller. The system was developed and tested on a flat grassy lawn bordered by trees and buildings which provided an ideal simulation of a landscape management application. A real time kinematic GPS (RTK-GPS) receiver was used for development and testing to obtain high resolution results. The machine was tested by navigating from one end of a 45 meter long path to the other end. The performance was measured by recording the position reported by the RTK-GPS on a handheld computer and calculating the crosstrack error between the recorded position and the straight line path.
The autonomous navigation system was able to navigate the machine from point to point but the performance was characterized by oscillations which were of higher amplitude and higher frequency than is acceptable for agricultural machines. Limitations in the microcontrollers’ ability to manipulate the position data were a major factor. However, the potential of small zero turning radius platforms for agricultural robotics is significant and the simplicity of controlling such vehicles is a great advantage over traditional tractors.
Agricultural Robotics Using Absolute Position Sensors on a Zero Turning Radius Platform

by

Nathaniel B. Powell

A Thesis Submitted to the Graduate Faculty of
North Carolina State University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Biological and Agricultural Engineering

Raleigh
March 29, 2006

Approved by:

Dr. Gary Roberson
Dr. Larry Stikeleather

Dr. Michael D. Boyette (Chair)  Dr. Doug Barlage
Dedication

Soli Deo Gloria
Biography

Nathaniel B. Powell was born on November 18, 1980 to Glenn and Michelle Powell in Richmond, Kentucky. After living for 6 years in Richmond, the family moved to Pittsboro, North Carolina. His parents and grandparents encouraged him to investigate a wide variety of interests and to develop talents and skills wherever possible. Their influence has given him a lifelong love of reading, carpentry and masonry, auto and farm mechanics, gardening, machining and antique machines.

Nathaniel graduated from Northwood High School, Pittsboro, North Carolina, in 1999. He then enrolled in the Computer Engineering curriculum at North Carolina State University, Raleigh, North Carolina. During this course of study he added a second major in Electrical Engineering in order to study classical control systems and power transmission engineering. He was given the opportunity to participate in a specialized capstone design program, the Engineering Entrepreneurship Program, where he and his partners developed a novel educational computer using open source software and alternative input devices. He was awarded Bachelor of Science degrees in Electrical Engineering and Computer Engineering in 2003.

To pursue his Master’s degree Nathaniel elected to enroll in the Biological and Agricultural Engineering curriculum at North Carolina State University because of his personal interest in biological systems and the opportunity to apply electrical engineering to those systems as the biological engineering field expands. He began assisting with the development of the department’s autonomous vehicle research and working under the direction of Dr. Mike Boyette.

He is a member of Providence Baptist Church, Raleigh, NC, and is engaged to be married to Julie Ann Stansbury in May of 2006.
Acknowledgements

This work came from the vision that Dr. Boyette had with Devin Carrol and Joe Madren, and certainly would not have been possible without the work that Joe Madren and Stuart Spencer put into their researches on this system. They also provided valuable insights and some great conversations about how things should work.

Glenn and Howard Powell, my father and grandfather, have both graced me with stimulating conversations about the whys and wherefores of the project, and have helped me iron out my thinking on many occasions.

Most importantly, my thanks to Julie, whose kind words and encouraging smiles and grace have helped me through the late nights and sleepy days and various frustrations, and reminded me to keep my eyes on the thing that truly matters.
# Table of Contents

List of Figures ................................................................. viii
List of Tables ................................................................. ix

1 Introduction ................................................................. 1
   1.1 Problem Statement .................................................. 2
   1.2 Proposed Solution .................................................. 3
   1.3 Safety Emphasis .................................................... 5
   1.4 Terminology ......................................................... 7
   1.5 Figures ............................................................... 8

2 Related Work ............................................................... 11
   2.1 Absolute vs. Relative Position .................................... 11
   2.2 The Global Positioning System ................................... 12
       2.2.1 Geodetic Coordinate Systems ............................... 14
   2.3 Theory of Navigation ............................................... 18
       2.3.1 Types of Navigation Error ................................. 19
       2.3.2 Comparison of Navigation Methods ....................... 21
       2.3.3 Plan of Coverage ........................................... 22
   2.4 A Review of Published Research .................................. 24
   2.5 Development of the NC State ZTR Robot ....................... 28
       2.5.1 Choosing the Vehicle ...................................... 28
       2.5.2 Choosing the Computer .................................... 31
       2.5.3 Choosing the Sensors ...................................... 33
       2.5.4 Navigation With a Laser Positioning System ............. 34
       2.5.5 Navigation With Machine Vision ......................... 37
B Navigation Code for the NC State ZTR Robot .......................... 107
   B.1 Main Navigation Program Dynamic C Implementation File .......... 107
   B.2 Actuator Control Dynamic C Library File .......................... 125
   B.3 GPS Input Processor Dynamic C Implementation File ............. 131
   B.4 Compass Input Dynamic C Library File ........................... 138
   B.5 Compass Input Processor Dynamic C Implementation File ......... 140
List of Figures

1.1 The NC State ZTR Robot, May 2004 ........................... 9
1.2 The NC State ZTR Robot, July 2005 ........................... 10

2.1 Visual Representation of Crosstrack and Angular Error .......... 41
2.2 Front View of the ZTR Robot Indicating Laser Positioning Subsystem . 42
2.3 Front View of the ZTR Robot Indicating Machine Vision Subsystem . 43
2.4 Side View of the ZTR Robot Indicating Machine Vision Subsystem . 44

3.1 Side View of the ZTR Robot Indicating Sensor Subsystems ........ 64
3.2 Rear view of the ZTR Robot Indicating Sensor Systems ........... 65
3.3 Visual Representation of Angle Conversion ...................... 66

4.1 The Navigation Points Used for Testing ........................... 76
4.2 Plot of Position Data From Test 26 .............................. 76
4.3 Crosstrack Error Sequence for Test 25 ............................ 77
4.4 Crosstrack Error Sequence for Test 26 ............................ 78
4.5 Crosstrack Error Sequence for Test 28 ............................ 79
4.6 Crosstrack Error Sequence for Test 29 ............................ 80
4.7 Crosstrack Error Sequence for Test 30 ............................ 81
4.8 Crosstrack Error Sequence for Test 31 ............................ 82
List of Tables

2.1 Constants Used in Calculating State Plane Coordinates in North Carolina 18

3.1 Explanation of the NMEA RMC Sentence .......................... 48

4.1 Coordinates of Navigation Points Used for Testing ................ 70
4.2 Summary Statistics for All Straight-Line Tests ...................... 71
4.3 Summary Statistics for Straight-Line Test Error Data ................ 72

A.1 Raw data and calculated SPC coordinates for Test 25 ............... 90
A.2 Raw data and calculated SPC coordinates for Test 26 ............... 93
A.3 Raw data and calculated SPC coordinates for Test 28 ............... 96
A.4 Raw data and calculated SPC coordinates for Test 29 ............... 98
A.5 Raw data and calculated SPC coordinates for Test 30 ............... 101
A.6 Raw data and calculated SPC coordinates for Test 31 ............... 104
Agriculture has changed rapidly in recent decades, and this trend is continuing. Larger but more competitive global markets have changed demands and prices, so the economics of agriculture have evolved to increase productivity and decrease labor costs. Agricultural engineering has focused development on increasing the productivity of farm labor by enabling an individual to work a larger area of land in a given amount of time, and on increasing the productivity of the land by increasing yields per acre. Since 1948 the number of people employed in agriculture has decreased by more than half, and the average number of hours worked per laborer has also decreased [1]. This has been made possible by many significant improvements in farming technologies and techniques.

As engineering has progressed the traditional methods of the farm have given way to new technologies and techniques. Equipment technology improvements have been a major factor in increasing yields per acre and decreasing the labor required per acre. Cultivation practices have changed as knowledge about agricultural systems and our ability to control those systems has grown. Practices have changed to reduce mechanical tillage, best use inputs such as pesticides, fertilizers and irrigation, and time maintenance and harvest operations to maximize yields. Simultaneously, equipment technologies have changed because of improvements in machining and manufacture and the development of new technologies which were previously unavailable, such as the Global Positioning System. Equipment technologies have also changed to suit the changes in cultivation practices and cultivation practices have changed as the capabilities of equipment have changed.

The development of agricultural equipment is often based on a concept for improv-
ing agricultural practices which require new equipment, and those changes in practices are refined by the capabilities of the equipment used to implement them. In this new century one of the agricultural practices which presents an opportunity for change is the amount of agricultural labor employed to operate power equipment such as tractors, mowers and specific purpose field equipment. Increasing the size of machines is one way to reduce labor needs. Equipment that operates autonomously will increase work a person can do beyond the limits of increasing the size of machines because the person will be able to supervise multiple machines simultaneously. This will further reduce labor needs and improve yields.

1.1 Problem Statement

The agriculture industry today includes a variety of situations which use power equipment, such as tractors, lawnmowers, forklifts, combine harvesters and sprayers. These machines are the backbone of modern agriculture. They enhance productivity by allowing one person to work faster and more effectively and increase production by enabling agricultural methods which maximize crop yields. Over the past 50 years efforts to increase the productivity of the men and women working in agriculture have led equipment manufacturers to increase the size of tractors and other machines used for row crop production so that a field can be worked in the fewest number of swaths and shortest time, maximizing the productivity of the machine operator.

This trend toward increasing machine size is reaching the limits of usefulness for a variety of reasons: a large machine cannot be operated as precisely as a small machine, implementing site-specific management practices is more complex and expensive with large equipment; heavy equipment causes more soil compaction than small equipment, which has been shown to decrease yields; and large equipment is significantly more expensive and complex than small equipment. These factors are offset by the fact that one large machine can be more energy efficient than several small machines, there are fewer systems to maintain, and large tractors are more versatile than small ones. Even
so, if the additional labor cost involved in using more machines can be eliminated, great benefits may be gained by using small to medium sized equipment instead of large equipment for many applications. To this end much work is being done to develop robotic equipment which is able to operate without constant operator input. The next step in this process will be development of autonomous equipment, which is able to operate without any routine operator input, and as the technology develops, removing the operator from the vehicle to a remote supervisory position.

When this has been accomplished the productivity of equipment operators will not be limited by the size of the machines they control but by the number of machines they can supervise. An operator driving a machine which can cover 1 hectare per hour has the same productivity as an operator supervising 3 machines which can each cover $\frac{1}{3}$ hectare per hour. Other researchers have suggested that autonomous machines should be developed with this philosophy. [2][3] Much preliminary work has been done in this area of machinery development, and some working autonomous agricultural machines have been created, but very little work has been done in developing small, low cost autonomous agricultural machines.

### 1.2 Proposed Solution

Our idea is that if an economical small or medium-sized machine configured to operate autonomously can be created, it will be useful in row crop operations like spraying and fertilizing, in agritourism for cutting crop mazes, and in landscaping operations like mowing. Landscaping operations are an area of agriculture which has an increasing need for labor because of the increasing acreage of high maintenance turf throughout the United States. In order to demonstrate the validity of our idea in a practical way we set out to create a low cost autonomous machine. In order to overcome some of the difficulties of navigation using conventional tractors and other machines we sought to work with a simpler platform which is still viable for agriculture. Ultimately a zero turning radius (ZTR) vehicle was chosen as the best balance between mechanical
simplicity and low input cost. Because ZTR lawnmowers are produced commercially in several different styles and configurations for the commercial landscaping industry we selected an off the shelf model to use for our prototype. Some small modifications were made to adapt the machine to our needs, but no changes were made which impair any of the original functions or mechanisms of the mower. The resulting prototype has been called the NC State ZTR Robot. Figures 1.1 and 1.2 show the prototype at two different phases in the development which will be discussed.

A major aspect of the design in this project was the consideration of costs. It is reasonable to design a machine which can work $\frac{1}{5}$ to $\frac{1}{3}$ the area a large machine can work in the same amount of time. This means that 3 to 5 of these small machines will be needed to replace one large machine, which will be reasonable if the combined cost of the small machines is less than the cost of the large machine. [2] Using autonomous machines with a single human supervisor per 3-5 machines means that the labor costs would be the same as having a human operator for a large machine. A large row-crop tractor such as the John Deere 8130 has a suggested retail price of $118,454.00 for the basic model (Deere & Company, 8130, Moline, Illinois). Based on this we have estimated that an autonomous machine able to do $\frac{1}{5}$ the work of the 8130 should retail at approximately $10,000.00. This can be further broken down to the major components of the system.

- $5,000 for the vehicle platform
- $1,000 for the computer systems and interfaces
- $4,000 for sensor systems

Since the development of this project uses commercial off the shelf systems, the total cost of the systems is assumed to be approximately the retail price of the final product. Sections 2.5 and 3.1 will show the selection process for choosing these systems.
1.3 Safety Emphasis

Safety is a major concern in developing autonomous machines. Poor safety records or catastrophic events in any area of automata could have serious societal and economic impacts by hindering development and acceptance of all automata. This is especially true in agriculture, where many implements are quite dangerous regardless of whether they are directly operated by humans or through using computers. It is valuable to consider that safe autonomous machines will be safer than human operators over long work periods because machines are not subject to fatigue and therefore while a human operator’s alertness will decrease, the machine always performs at the same level.

Safety concerns for autonomous equipment can be separated into two types of reliability. For a machine to be safe it must reliably navigate its assigned course and reliably identify and avoid obstacles in the work area. These two aspects of safety can be addressed as separate behaviors when developing autonomous machines even though both behaviors control the machine through the same systems. This behavior-based development process comes from the research documented by Brooks [4]. Using this process a safe autonomous machine can be developed in phases which develop two major behaviors corresponding to the two aspects of safety. First the navigation behavior is developed, and then the obstacle avoidance behavior is developed. This research describes development of the navigation behavior. During the process of developing safe navigation behavior it is essential that careful steps be taken to ensure that the machine can be stopped easily and safely in any unplanned circumstances during development and testing.

To ensure the safety of the project at this point in its development, where all machine operation is monitored by the experimenters at close range, a safety stop system was installed which can be operated either on the machine or remotely via a wireless system when the experimenters supervising the testing feel it necessary to terminate the test. When the navigation development stage of the project is completed and we are ready to try simple deployments with more routine supervision it
will be necessary to develop a system of additional sensors and programming to incorporate obstacle detection and avoidance and implement safety into the machine’s control programming.

Machine vision will be a major element of a safety system for an autonomous agricultural vehicle. Machine vision for this purpose could use CCD cameras and other imaging sensors such as millimeter wave radar. Because the focus of this project was on sensors for navigation machine vision for obstacle detection will not be discussed further. Machine vision can also be used for navigation, and this use of machine vision will be discussed in Section 2.5.5.

Navigation safety is a matter of ensuring that the machine is always where it is programmed to be, traveling in the correct direction and at a safe speed. One way to make the machine navigate more safely is through a simple assumption: agriculture is assumed to be a controlled environment. The machine is not expected to be able to do exploratory work, but will always operate in known, pre-mapped locations. This assumption reduces some of the demands on safe controls, since it is acceptable for the machine to shut down if it is not in a recognized location. This means that when safety problems are addressed, they can be limited to reasonable differences between the sensed environment and the expected (mapped) environment.

This system is also being developed with a second assumption: to be safe, the machine’s operation must always be supervised by a person with the ability to immediately stop it in the event of any problem. For this security a radio based safety stop switch was built for the system, and other safeguards will be developed as the project continues.

The project is being developed with an eye to the safety requirements for a functioning autonomous machine, but the focus at present is in developing high quality navigation which will be inherently safe. The needs for obstacle detection and avoidance have not been addressed at this time. The problem will not be completely solved until these needs have been met, and in the long term this project is intended to be a test bed for such development.
1.4 Terminology

Throughout this paper where a numerical value is described as an "error" it refers to the 2σ ("2-sigma") error, that is, that the value is within two standard deviations plus or minus the mean, corresponding to 95 percent of total possible values. If a numerical value is described as an "individual error" it refers to an individual measurement.

In many equations throughout the text the coordinates of various points are used. The labels for these coordinate variables are represented using subscripts. For example, a point called $A$ with coordinates $(x, y)$ has its $x$ coordinate labeled $A_x$, and its $y$ coordinate labeled $A_y$. Similarly, a point called $Start$ with coordinates $(northing, easting)$ has components $Start_{northing}$ and $Start_{easting}$ or $Start_{nor}$ and $Start_{east}$ to save space.
1.5 Figures
Figure 1.1: The NC State ZTR Robot, May 2004
Figure 1.2: The NC State ZTR Robot, July 2005
Researchers, engineers and agriculturalists have undertaken many different efforts to create autonomous agricultural equipment in the more than 125 years since the traction engine was developed for farming. In the last decade the frequency and vigor of these efforts has increased, producing several different prototypes of autonomous machines, with different abilities and efficacies. The maturing fields of mobile robotics and non-linear control systems theory have influenced the process of developing agricultural automata. Changing technologies have spurred on research and development as computers have become smaller, more powerful, more adaptable and less expensive, and machinery benefits from new drive systems and computerized control. Sensing systems of all types have evolved at a remarkable rate, increasing in precision and speed and decreasing in cost. Finally, some machinery needs have changed as agriculture has adapted to new techniques and embraced new technologies.

2.1 Absolute vs. Relative Position

The position of the machine as it is working must be known in order to do work. An autonomous position control system must have the means for measuring the position of interest. The position of interest may be different for different machine types: for a mobile robot, the position of the robot itself; for a manipulator arm the position of the end effector; or for a more specialized machine some application specific position. For an agricultural mobile robot it is necessary to know the position of the vehicle and the implement it is using. For a drawbar pulled implement this is a very challenging problem because the relationship between vehicle motion and implement motion varies
with respect to time. Other vehicle-implement attachments such as the popular three point hitch have simpler relationships. For the machine discussed in this paper the implement, a mower deck, is positioned directly beneath the center of the machine, so the implement position is the same as the machine position. This was used to simplify the navigation processes.

Position measurement is also specific to the type of work being done. In general, these applications can be divided into absolute position measurement and relative position measurement. Absolute position can be defined as position measured with respect to a coordinate framework which includes the entire possible work area, and relative position is measured with respect to a coordinate framework local to the immediate surroundings of the machine. In other words, an absolute position sensor will always report a position which is unique within a work area. A relative position sensor may report a position which can occur in many different parts of the work area. Examples of absolute position systems used in agricultural automata include geodetic coordinate systems, buried cable guidance systems and reflected laser triangulation systems. Examples of relative position systems include all means of measuring the machine’s position with respect to plants or furrows, both contact and non-contact methods. Wilson [5] offers a slightly different definition which captures the same idea by describing two types of guidance: 

(1) guidance with respect to a directrix generated by the previous pass or operation; (2) guidance with respect to a directrix generated by fixed points in the field;” referring to relative and absolute position, respectively.

### 2.2 The Global Positioning System

This body of work will deal only with absolute position sensing using the global positioning system. The global positioning system (GPS) is composed of three major parts: a constellation of satellites, the control segment, and receivers. There are 24 satellites orbiting the earth transmitting unique and complex signals. These are maintained by the control segment, a set of five ground stations on the earth’s surface which monitor
the satellites and transmit correction information to the satellites. Each GPS user has a receiver which can receive and decode those signals, measuring the time difference between transmission and reception by comparing the received codes to the known transmitted codes and using that time difference to calculate the distance from those satellites. Because of problems with atmospheric interference and multi-path transmissions ordinary GPS sensors are only accurate to within a few meters. There are different techniques for compensating for these errors which can generally be grouped into differential corrections (DGPS), carrier-phase differential corrections (CP-DGPS), and Real-Time Kinematic corrections (RTK-GPS or RTK). The correction method used depends on the receiver, and DGPS is the least precise and least expensive, and RTK-GPS is the most precise and most expensive. Because the final cost of this system is a major design constraint, we attempted to specify the needs of the system and use the cheapest receiver that meets those needs.

For agricultural operations the machine or implement position of interest is the position with respect to plants or to previous furrowing or harvesting operations, a relative position as defined above. When a person is performing these tasks he measures the position visually, identifying the reference and measuring the distance between the reference and the actual position. This method of navigation has been implemented in various ways on robotic machines, including an investigation using the NC State ZTR Robot with a machine vision system demonstrating the simplicity and effectiveness of the method. However, relative position measurements only work when there is a reference and the measuring system can identify that reference. It is also desirable for an agricultural robot to be able to navigate using absolute position measurements.

Relative position measurement has an inherently higher precision because the position reference is also the object on which work is to be done. This means that a relative position navigation system will always strive to position the machine in the proper place with respect to the crop row or other object which the machine is intended to work on. In order to achieve the same work performance using absolute position measurements the position measurements must be very precise and the system must
have a very precise plan of coverage. An absolute position system may navigate precisely to the coordinates specified, but if the crop row is not along the specified path because of errors in the plan of coverage, the machine will not be correctly positioned with respect to the crop row. Because errors in an absolute positioning system may occur both in the sensed position and the desired position it is not expected that absolute position navigation will be superior in all situations, but in situations where relative position navigation is not possible a high quality absolute position system should be able to provide adequate navigation. This includes situations where the relative position sensor is malfunctioning or unable to sense any result because of some interference such as dust or intense vibration and situations where the relative position reference is not present or inadequate for proper navigation, such as a spot in a crop row where no crops are present, or when the sensed reference has no distinguishing characteristics, such as a machine vision image of a dense patch of grass or grain.

The errors being discussed here refer to the crosstrack error (c.f. section 2.3.1) and for a DGPS are on the order of 10-30 cm. For some agricultural situations this is negligible, such as cutting a crop maze, turning a vehicle in the headland of a field or driving along a road or path to a field. If the system is designed so that the error is always an overlap error then it may be adequate for most turf-grass situations. However, in other situations, most notably cultivating row crops, a 10 cm error is the maximum allowable. For this research a 30 cm crosstrack error was the goal.

### 2.2.1 Geodetic Coordinate Systems

Geographical locations can be represented in several different coordinate systems. Most maps and GPS units use the Latitude-Longitude system. For computerized navigation the most convenient coordinates would be a rectilinear system based on a unit of length measurement. Two common examples of this are the Universal Transverse Mercator (UTM) system and the State Plane Coordinate System 1983 (SPC83 or SPC).

The earth is an oblate spheroid with an irregular surface and no two-dimensional
surface projection can fit that surface perfectly. Consequently no two dimensional coordinate system can perfectly describe a point on the earth’s surface. This means that the process of converting actual position into two-dimensional coordinates will introduce some error in the position data. In areas close to sea level with no significant slopes this can reasonably be neglected. A more serious problem is that no rectilinear system can be fitted to an irregular oblate spheroid surface. This means that if we use a system such as UTM or SPC83 there will be a certain inherent error in any measurement made with that system. However, these systems were designed so that the inherent error never exceeds a maximum limit, and the maximum limits are quite small, 0.04% and 0.01% of the measurement respectively.

**Latitude-Longitude**

The Latitude-Longitude system was created for cartography and as our knowledge of the earth’s shape has improved the system has been modified to work on the oblate spheroid shape. Latitude and longitude lines describe great circles and the intersections of latitude and longitude lines are not perpendicular. This means that calculating angles, line lengths, and crosstrack errors cannot be done using ordinary plane geometry. In fact, the calculations are significantly more complicated. Computers can handle this, but for a simple robot which will not have to travel great distances a system which uses Cartesian coordinates is much superior for algorithm development, verification, accuracy and execution time.

**Universal Transverse Mercator (UTM)**

The Universal Transverse Mercator system was created for military use as a rectilinear coordinate system covering the entire earth, except for areas near the north and south poles. In the UTM system the earth is divided into 60 zones which run from 84°N to 80°S and are 6°of longitude wide. Each zone is subdivided into a north subzone and south subzone for informal application, or 20 subzones in official applications. The
coordinates in each zone are based on a transverse Mercator projection of that zone onto a plane. The coordinates are measured in meters of northing and easting from a specified origin for that zone. This coordinate system is designed to be accurate to 1 unit in 2,500. The transformation equations used to compute UTM coordinates from Latitude-Longitude coordinates can be found in Hooijberg[6]. These equations can be implemented on a computer readily, but the constants for each zone are different. The generality of the implementation is dependent on providing the constants for any zone in which the machine might be used and a means for checking the input data to discern which zone constants are needed. An implementation which works worldwide would have to have the constants for every zone stored in its memory. Crossing the boundary of two zones will also cause severe difficulties because the easting coordinate will jump from a low number to a high number when traveling west or a high number to a low number when traveling east. This is a problem with implementing this coordinate system in North Carolina, since the state is divided between zones 17 and 18. Other implementation challenges will be discussed with the implementation of State Plane Coordinates.

**State Plane Coordinate System (SPC)**

The State Plane Coordinate system of 1983 is a coordinate system designed for use in the United States. It is less general than the UTM system because it uses three different types of transform to project the earth’s surface onto a plane. The type of projection used for a given area depends on the shape of that area. Each projection requires a different set of translation equations. The projection and the translation constants vary from state to state, and even within states when it is necessary for a state to be divided into subzones. For this reason the coordinates are not compatible between zones, so crossing zone boundaries is a problem for SPC just as it is for UTM.

Despite these challenges, the SPC system has some advantages over UTM. Like UTM the State Plane Coordinates can be calculated in meters of northing and easting.
It is designed to be four times as accurate as UTM, with any measurement correct to 1 unit in 10,000 which is valuable when working with precision machines working in an area hundreds or even thousands of meters square. In several states, including North Carolina, there is only one SPC zone for the state. Overall, the State Plane system requires more programming to generalize, and will not work outside of the United States, but it is more accurate, and in North Carolina it provides a simpler solution than UTM.

Converting Latitude-Longitude coordinates to State Plane coordinates depends on the projection used. Zones which are longer north-south than east-west use a transverse Mercator projection similar to the one used for UTM. Zones which are longer east-west than north-south, like North Carolina, use the Lambert Conic Conformal Projection. The third possible projection, the oblique Mercator projection, is only used in the zone covering the Alaska panhandle. The equations for transforming Latitude-Longitude into State Plane coordinates in a Lambert Conformal Projection[7] are as follows:

\[ Q = \frac{1}{2} \left[ \ln \frac{1 + \sin \phi}{1 - \sin \phi} - e \ln \frac{1 + e \sin \phi}{1 - e \sin \phi} \right] \]  
(2.1)

\[ R = \frac{K}{\epsilon Q \sin \phi_0} \]  
(2.2)

\[ \gamma = (\lambda_0 - \lambda) \sin \phi_0 \]  
(2.3)

\[ N = R_b + N_b - R \cos \gamma \]  
(2.4)

\[ E = E_0 + R \sin \gamma \]  
(2.5)

There are several constants used in these equations which are specific to the SPC zone. Some of these constants are established by the geoid used, which for SPC83 is the Geodetic Reference System of 1980 (GRS80). Other constants were chosen as part of the SPC zone definition, and some of the constants are computed from these first constants. Table 2.1 gives the values of all the constants for the equations above.

The coordinate conversion equations were incorporated into the GPS input code.
Table 2.1: Constants Used in Calculating State Plane Coordinates in North Carolina[7].

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$</td>
<td>latitude</td>
<td>Latitude coordinate to be converted (radians)</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>longitude</td>
<td>Longitude coordinate to be converted (radians)</td>
</tr>
<tr>
<td>$e$</td>
<td>0.0818191910428</td>
<td>Eccentricity of the GRS80 ellipsoid (meters)</td>
</tr>
<tr>
<td>$K$</td>
<td>13178320.6222</td>
<td>Mapping radius at the equator (meters)</td>
</tr>
<tr>
<td>$\phi_0$</td>
<td>0.615259254692</td>
<td>Central parallel of the projection (radians)</td>
</tr>
<tr>
<td>$\lambda_0$</td>
<td>1.37881010908</td>
<td>Central meridian of the projection (radians)</td>
</tr>
<tr>
<td>$R_b$</td>
<td>9199785.5932</td>
<td>Mapping radius at the latitude of grid origin (meters)</td>
</tr>
<tr>
<td>$N_b$</td>
<td>0.0000</td>
<td>Northing value of the grid origin (meters)</td>
</tr>
<tr>
<td>$E_0$</td>
<td>609601.2199</td>
<td>Easting value of the grid origin (meters)</td>
</tr>
</tbody>
</table>

in such a way that the Latitude and Longitude coordinates are converted into SPC coordinates as soon as the data is parsed out of the NMEA sentence. This means that the program does not have to store the Latitude and Longitude but only the SPC Northing and Easting. This improves the memory efficiency of the process. However, the SPC conversions brought to light a problem which had previously gone unnoticed. This will be discussed in section 3.2.1. With the coordinate system established so that the computer understands the machine’s position, it is worthwhile to discuss the algorithms needed for navigation.

2.3 Theory of Navigation

Navigation is the means the machine uses to make its path of travel the same as the desired path. In agriculture it is very important that the machine follows the desired path as precisely as possible. Therefore, the first tenet of this theory of navigation is that the most important thing is that the machine have minimal crosstrack error on the desired path. Fortunately, for most agricultural situations desired paths can be
broken into straight line segments of various lengths. This is desirable since arbitrary
curve following behavior would be quite unnecessarily complex for agriculture. So the
second tenet of the theory of navigation is that the machine will travel along paths
composed of straight line segments joined at arbitrary angles. This tenet will only
work for zero turning radius vehicles because they can rotate to any arbitrary angle
at the end of a line segment. Other vehicles will require more qualifications to the
path design. Fortunately, these two tenets form an adequate basis for developing ZTR
navigation behavior. There are several possible ways to implement this behavior.

2.3.1 Types of Navigation Error

It will be useful to start the discussion of navigation with a close treatment of the
types of navigation error which occur in straight line navigation. The first problem is
position error. Position error in general is any error in the machine’s position in the
work space. This error may be due to sensor error, where the sensor is reporting the
wrong position; it may be due to incorrect path data, or it may be due to imprecise
or flawed navigation. In this discussion it is assumed that the sensor and the path are
correct, and so any position error would be due to bad navigation. The position error
can be measured by measuring the perpendicular distance from the control point of
the machine to the path line. This is called crosstrack or offset error and is illustrated
in Figure 2.1a. Another type of error is orientation error. An orientation error occurs
when the machine is in the proper position but its direction of travel is not the same
as the desired direction of travel, or the implement being operated with the machine
is not oriented correctly. These errors can again be due to sensors, incorrect input, or
navigation. For this project the only orientation of interest is the machine’s direction of
travel. The difference between path line heading and the machine’s direction of travel
will be referred to as angular error, illustrated in Figure 2.1b. As Larsen points out,
orientation errors for agricultural machines with drawbar towed implements are more
complex.[8]
For most agricultural situations position errors are a bigger problem than angular errors. This project has focused on minimizing crosstrack error. However, in navigation the best straight line travel can be achieved by minimizing both crosstrack and angular error. For that reason angular error is considered here, and it is very useful in the navigation theory. These errors are easily calculated if the work area is assumed to be a cartesian plane. Angular error is the simple difference between the heading of the desired path and the machine’s present direction of travel, however, this will range from $-360^\circ$ to $360^\circ$. It desirable to have this range limited from $-180^\circ$ to $180^\circ$, this can be done by adding $360^\circ$ to any value less than $-180^\circ$ and adding $-360^\circ$ to any value more than $180^\circ$.

The angular error is the difference between the direction of the path, represented as $\theta$, and the machine’s direction of travel, represented as $\alpha$.

$$\text{Error}_{\text{angular}} = \theta - \alpha$$ (2.6)

The crosstrack error calculation in a cartesian work area is the same as the calculation of the perpendicular distance from a line to a point on the plane. This calculation from a line $AB$ to a point $C$ on an $x, y$ plane is:

$$\text{Error}_{\text{xtrack}} = \frac{(B_y - A_y)(B_x - C_x) - (B_x - A_x)(B_y - C_y)}{\sqrt{(B_y - A_y)^2 + (B_x - A_x)^2}}$$ (2.7)

Transferring this to the autonomous vehicle system the line $AB$ is defined using state plane coordinates, and the point $C$ is the position of the machine. The equation then becomes:

$$\text{Error}_{\text{xtrack}} = \frac{(B_{nor} - A_{nor})(B_{east} - C_{east}) - (B_{east} - A_{east})(B_{nor} - C_{nor})}{\sqrt{(B_{nor} - A_{nor})^2 + (B_{east} - A_{east})^2}}$$ (2.8)

For a given path segment $AB$ when $C$ is moving most of the values in the equation are constant and this simplifies the amount of calculation required.
2.3.2 Comparison of Navigation Methods

The first possibility for line following is a scheme where the line is composed of a start point and an end point, and the heading (absolute direction) of the line is calculated. In this scheme the machine is positioned at the start point with its direction of travel the same as the heading of the line. The machine starts going forward, periodically comparing its direction of travel to the heading of the line. The difference between direction of travel and line heading will generally be referred to as angular error. As the machine travels, its heading is changed by steering slightly to the left or right to reduce the angular error. In order to prevent over-reaction the system is set to only respond to errors greater than a threshold magnitude, preliminary research on the NC State robot used a threshold of 10 degrees. This is a very simple navigation scheme, and there are several problems which make it a poor choice.

The most obvious problem with this scheme is that crosstrack error has no effect on the navigation. Since crosstrack error is not measured there is no feedback and the navigation scheme cannot be directly designed to minimize the error. This scheme will generally enable the machine to travel from the start point to the end point, but it will not follow a straight line between them. Using angular error by itself to try to minimize position error will be highly oscillatory and may even become unstable. This is due to the way position error lags angular error. This effect may be reduced depending on the architecture of the machine and its forward speed. The problem is greatest on front-wheel steer/rear-wheel drive vehicles like tractors and other standard agricultural machines. Articulated vehicles may have less difficulty, and zero turning radius vehicles the least problem with this. The problem is also speed dependent, and that dependency varies with the architecture. The best system for this navigation method would be a ZTR machine which can turn rapidly with respect to its forward speed.

A variation on this line following scheme takes the line and calculates a set of points which lie on that line at a short fixed distance apart. The machine navigates in the
same manner described above to each of these points in succession. This limits the crosstrack error by forcing the machine to be on the correct path at the navigation points. This method can reduce the average crosstrack error, but will not eliminate the problems with the system, and may add other difficulties as the navigation program becomes increasingly complex.

The second navigation scheme considered here uses a path line composed of a start point, an end point and a calculated line heading. As the machine travels along the line it calculates both the crosstrack error and the angular error. These errors are combined with linear scaling coefficients into an aggregate error as shown in equation 2.9, and the navigation control system is designed to minimize this aggregate error using classical control techniques. This method is only slightly more complicated than the first method, but is significantly more effective. The reasoning behind it was that the best way for a machine to travel along a straight line is to be both positioned on the line and headed in the same direction as the line. The linear combination of the error components gives the designer flexibility in designing the control system because the coefficients of the error terms and the limits of the error terms set the maximum value for the aggregate error.

\[
Error_{aggregate} = K_1 Error_{xtrack} + K_2 Error_{angular} \tag{2.9}
\]

### 2.3.3 Plan of Coverage

Once the navigation program has been developed to the point that the machine can acceptably travel along straight lines it is necessary to consider how the whole path, composed of straight lines connected at arbitrary angles, will be designed, created and stored in the machine’s memory. The design and creation of paths has implications for the design of the navigation program. The storage aspect is a simpler question and may be implemented in a few different ways without significant changes to the system design and programming.
Designing paths for agricultural tasks can be simple or complex depending on the task. Examples of three different types of path based on the task are: landscape mowing; spraying row crops; and cutting a crop maze. For cosmetic mowing it is desirable to get complete coverage of the area. A small amount of pass to pass overlap is desirable, but missed areas are not acceptable. There are often obstacles to avoid and complex curves to navigate. The crosstrack error must be small to avoid damage to non-turf landscaping, other items in the area and the machine itself. This path could be reused several times in a season, and perhaps even from year to year. This path will likely be made up of both short and long lines, often joined at sharp angles. It is reasonable to create the path by recording the path taken by a human driving the machine through the task. Spraying row crops requires navigation down a row, which is rarely a straight line. It is desirable to minimize both overlap and missed areas between passes. The crosstrack error must be small to avoid damage to crops. This path is likely to be reused during the season, but will be different from year to year. For a general non-circular field the path can be composed of long straight lines joined at shallow angles, plus headland turns. It may be reasonable to create this path by recording a human operator’s work, or it may be desirable to create the path using a combination of GIS input and real-time machine vision corrections. Cutting a crop maze requires precise navigation along a path composed of lines of arbitrary length joined at arbitrary angles. Some more complex maneuvers may also be required, such as cutting a circular area at the end of a path. Small crosstrack error can be tolerated but should be minimized to avoid excess crop removal. The path is not likely to be reused. It must be created using GIS input.

These different possible agricultural tasks show that there are various task related objectives which may dictate the type of autonomous machine needed. In agriculture many types of machines have been developed for different tasks, and there is no single machine which performs all tasks equally well. However, different machines use different drive systems and require specific adaptations of the automation strategy. For this reason this project focuses on some specific elements of automation technology and a
method of combining those to produce an efficient and effective autonomous machine at low cost, which may have significant agricultural ability in its own right, as well as serving as a test platform for further autonomous systems development.

2.4 A Review of Published Research

The earliest means for automating agricultural machines used various mechanical sensors for determining the position of a furrow or a row of crops [5][9]. These are generally described as mechanical sensing arms connecting to the steering linkage, which can relieve the operator of the need to steer as the machine is going down the row but require intervention at the end of the row to turn and line up for the next pass. This sort of mechanization is able to free up the operator to concentrate on controlling the implement, and is realized today in global positioning system based auto-steering systems which are available commercially from several vendors including Trimble and John Deere.

Wilson [5] notes the first effort to achieve autonomous operation for agricultural equipment patented in 1941 by F.W. Andrew. This system was a means of allowing a tractor to plow a circular field by winding or unwinding a cable on a drum at the center of the field. However, true autonomous agricultural machines could not effectively be considered until the development of small and inexpensive digital computing, and robust high precision electronic sensors. By comparing a review of this research area written in 1991 [9] to one written in 2000 [5] it is easy to see that the research of that decade was oriented around changing sensor technology.

Two major sensor technologies have been heavily used in recent experiments with agricultural automata: the global positioning system (GPS) and machine vision, corresponding to absolute and relative position measurement. Some more recent experiments have used both technologies together, but most have focused on one or the other [10][11]. This project will be using the GPS system only. Other research has validated the performance of the NC State ZTR Robot using machine vision [12]. GPS has
been identified as a useful technology for agriculture since its creation. In 1994 work was published on experiments done by Larsen, et al [8], testing the utility of the then newly completed GPS system for navigation in precision agriculture and developing a machine which used the GPS system to navigate along a line. This work demonstrated 1 cm position accuracy at update rates of 2 Hz.

Pliarski, et al [10], reported in 1999 that GPS based navigation was ”more accurate and more robust” than vision based navigation. Their system used an RTK-GPS with 10 cm CEP accuracy and a 5 Hz update rate, and a dead-reckoning system based on wheel encoders and gyroscopic inertial navigation sensors on a windrowing machine. These sensors were combined by referencing the dead reckoning calculations against the GPS input. The machine vision system used was a separate guidance system, so that the machine could be guided either by GPS or by machine vision. They reported results of an average error along a straight line of 4-6 cm. Several other researchers have followed similar methodology, using RTK-GPS in combination with a dead reckoning system. Zhang, et al [13], implemented a system with RTK-GPS and an inertial measurement unit combined by a rule based fusion model on a standard medium sized tractor and reported 15 cm peak error. Stentz, et al [11], used an RTK-GPS with 2 cm accuracy combined with a fiber optic gyroscope through the Extended Kalman Filter to achieve autonomous navigation within 10 cm on a standard medium sized tractor. Stentz also implemented a machine vision obstacle detection system successfully on this system. Noguchi, et al [14], used an inertial measurement unit to compensate for the error of RTK-GPS measurements due to roll and pitch of the tractor on uneven ground, reporting maximum errors of 8 cm using a small standard tractor. Most recently, Bak, et al [15], have reported on their development of an autonomous machine which uses CP-DGPS with specified accuracy of 2 cm, combined with a fiber optic gyroscope, wheel and steering encoders and an electronic compass sensor. This system was developed on a custom built portal tractor with four wheel drive and four wheel steering. The reported results are 8-11 cm errors in field tests.

Other researchers have tried a ”single sensor” solution using GPS without any
dead reckoning system. These projects have shown successful navigation, but have also identified difficulties with this method. Bell, reporting on work done at Stanford University between 1995 and 2000 [16], describes a Carrier-Phase Differential GPS system on a standard tractor using four GPS antennae mounted on the tractor cab separated sufficiently that the roll and pitch of the tractor could be detected as well as the yaw and position. This was considered necessary for high precision navigation since the GPS antenna(s) must be mounted high on the vehicle to maximize reception of satellite signals, but the height exacerbates the error due to roll and pitch if the control point is assumed to be directly beneath the antenna. With knowledge of the pitch and roll and the mounting position of the antenna relative to the control point the actual position of the control point can be calculated from the position of the antenna. Otherwise the vehicle must be assumed to be operating on relatively flat ground. Bell reports navigation with errors within 4 cm traveling in straight lines on level ground and 12 cm errors on rolling terrain.

Stombaugh [17] also describes work done with a single RTK-GPS on a standard tractor. The specifications for the RTK-GPS are 20 cm accuracy with 5 Hz updates. In this case the roll and pitch were neglected, although Stombaugh notes in his conclusions that these effects caused significant variation in input and should be compensated. The lag between position updates and effective corrections because of the vehicle kinematics of the standard tractor led Stombaugh to recommend mounting the GPS antenna ahead of the control point on the machine. This is intended to compensate for the lag of the heading change on a standard tractor by having the sensor physically lead the control point. With this system navigation was demonstrated with errors within 16 cm. Stombaugh also mentions that interference and instability were a problem in the microcontroller used for controlling the tractor.

Most recently, Thuilot [18] has described a standard tractor equipped with a single CP-DGPS with 2 cm accuracy at 10 Hz. The major contribution of this work is seen in their chained form model of their vehicle and the Kalman state reconstructor derived from that model, which is used to predict the future position and orientation of the
machine to compensate for the heading response lag also observed by Stombaugh. The result was navigation with errors of 6-10 cm. It can be seen from these examples that the physical configuration of the machine is a dominant force in the development of autonomous navigation. Much of the work done by Stombaugh and Thuilot was necessary to compensate for the heading change response lag inherent in a rear wheel drive/four wheel drive vehicle with front wheel steering when the control point is on the rear axle of the vehicle.

For this reason it is valuable to consider the nature of the vehicle and the needs of the tasks the vehicle will perform. A conventional tractor is a standard platform which is very familiar and able to be used for many different tasks. However, an autonomous machine may not require the versatility of the conventional tractor because it will only be called on to perform a subset of those possible tractor tasks. Autonomous vehicles are particularly well suited to tasks that require long duration repetitive activity, and also to tasks that require very precise navigation. Examples of these tasks are mowing, spraying crops and cutting crop mazes, as described above. Autonomous vehicles are not well suited to tasks that are performed irregularly or require considerable creativity in navigation, such as freeing a stuck vehicle.

Blackmore, et al [3][19], have described some requirements for autonomous agricultural vehicles based on the needs of safety and general agricultural use and suggested three different classes of vehicle which might suit the needs of autonomous agricultural machines. However, neither of these papers looks at vehicles for autonomous agriculture from a task oriented point of view. Many agricultural machines are highly task specific, such as harvesters, high-bodied sprayers and loaders. This specialization may be accepted in autonomous vehicles if it leads to decrease in costs and increase in efficiency. For this reason it is desirable to investigate non-conventional vehicles for their applicability to autonomous agriculture. In the next section, the criteria for selecting a vehicle for this project will be discussed, along with the general development of the project.
2.5 Development of the NC State ZTR Robot

Research on this project at NC State began in 2001, and the physical development of the prototype began in 2002 with the selection of a vehicle to use as the basis for the robot and the selection of a computer to do the navigation control. Some small modifications were made to the vehicle to make room for the necessary components, and possible position sensor technologies were identified for use on the robot. An absolute position method using a laser positioning system was selected as the most promising avenue of investigation. In 2003 the work of integrating these technologies through software began. The robot navigation software was developed and some testing was done. A photo of the prototype during this stage of the development can be seen in Figure 1.1. As will be discussed in section 2.5.4, the overall performance of the system was disappointing, primarily due to the inadequacy of the sensor used. In late 2003 and early 2004 a second investigation was done using the same platform and computer system combined with a relative position method using a CCD (coupled capacitive discharge) camera. This work has been reported by Spencer [12]. All this work has contributed substantially to the understanding of agricultural robotics at NC State, and has had a significant influence on the development of this body of work. A short description of the work of these first two phases shows the progress of thought and system development.

2.5.1 Choosing the Vehicle

The selection of a vehicle platform was a process of balancing various factors. As Spencer [12] has noted, the size of the machine, the cost and the drive system were the major factors. The desire to use a machine with small size and low cost has been explained previously. The specifications developed for the project suggest a machine size of 500 to 1,000 pounds, and a 20-30 hp motor. The cost is limited to about $5,000.00. The selection of the drive system has more complex implications.
Types of Drive Systems

Agricultural machines can have several different types of steering and drivetrain systems. Some of these systems are significantly more amenable to electro-mechanical control than others. First, there is an issue of fitting actuators to control the drive. The conventional tractor with a mechanical clutch and gearbox is the hardest to adapt to automatic control. Machines with hydrostatic or mechanical continuously variable or infinitely variable transmissions are more readily adapted for automatic control, although computer controlled transmissions may have proprietary systems which are difficult to incorporate without consultation with the manufacturer. Second, the steering system will have a profound impact on the control of the vehicle. Front wheel steered vehicles such as conventional tractors present difficulties in automation at the mechanical, navigation and path planning levels because these vehicles have steering decoupled from the drive system, and so require additional actuators and require significant space to make turns of 90° or more. Conventional tractors are cost effective, reliable and well understood, but more difficult to control. Zero turning radius (ZTR) vehicles are the ideal choice for automation because their turning flexibility makes navigation and path planning much simpler.

Different ZTR options have been used in agriculture. The most familiar ZTR vehicles are track-driven tractors, which use two large tracks instead of the conventional four wheels. This type of vehicle is experiencing a new surge in popularity, since tracks reduce soil compaction by increasing ground contact area. Steering for this type of vehicle is done by varying the speeds of the tracks independently, so the machine can make a turn of practically any desired radius. Skid steer machines operate by similar principles, and are also quite common, but are not adapted for general agricultural use.

Another type of ZTR vehicle which has been proposed [15][19] for autonomous use is the four wheel drive and four wheel steer vehicle, also called the portal tractor. This platform has the stability and power of a conventional tractor with considerably
higher mobility and the ability to make turns of any desired radius. However, there are no vehicles of this type currently in production, the drive and steering mechanisms are mechanically more complex than other vehicles, and the expense of manufacturing a prototype for this project would be prohibitive. The final type of ZTR vehicle available for agriculture is the power wheel steering [20] type of vehicle, which has two independent drive wheels and two freewheeling casters, and speed and direction are controlled by varying the rotational speed of the drive wheels. This configuration is common on commercial ZTR lawnmowers.

A vehicle with power wheel steering commonly has a gasoline engine powering one or two hydraulic pumps which power independent hydraulic motors on the drive wheels. The speed of the wheel is controlled by a spool valve between the pump and the motor which is actuated by a control lever. These are normally 3 position, 4 way valves to allow the motor to be driven forward or backward. This system is highly adaptable to electro-mechanical control. There are two simple possibilities for adding electromechanical actuators in this system. The simplest method is to install actuators which operate the original control levers. Alternatively, electrically operated spool valves can replace the mechanically operated valves in the hydraulic systems. This is the type of vehicle that was selected for this project. With these specifications all that was necessary was finding a vehicle commercially available which matched project parameters. The machine chosen was the John Deere Quik-Trak 647 (Deere & Company, 647, Moline, Illinois).

The Quik-Trak 647 is a small commercial mower with a 35 hp gasoline engine, weighing 750 pounds. It uses power wheel steering operated by two levers. The operator stands on a small platform on the back. This design is ideal for adapting to autonomous operation. The operator interfaces are easily left intact because they take up very little space and the manual control mechanism was preserved by installing electromechanical actuators on the control levers which disengage when no power is supplied to them. To accommodate the computer and other equipment which might be needed for this system a tray was constructed for the front of the machine above the engine. A mast
was installed over the rear axle approximately over the machine control point, so that
the laser transceiver unit and later the GPS antenna would be positioned properly up
high and over the control point. These modifications did not remove any function from
the original machine.

2.5.2 Choosing the Computer

Microcontroller computing is the ideal means for implementing autonomous operations
on such a machine. A microcontroller is a small, self-contained computer configured
with purpose specific hardware. Since microcontrollers are available with a myriad of
configurations it was merely a matter of identifying the needs of the project to select
a computer system. Again the focus was on the best functional configuration for the
lowest cost, but with an added desire for expansion and adaptability because of the
open-ended nature of the project.

There are a variety of reasons that a microcontroller is a superior choice to a
low-cost personal computer. Microcontrollers are a good choice for robotics because
they can be obtained in many different configurations and are easily connected to
other hardware using built-in analog inputs and outputs, relay outputs or serial ports.
Microcontrollers are generally programmed in a high level computer language such as C,
C++ or BASIC. Generally the only program running on a microcontroller is the control
program written for the specific purpose; most microcontroller applications do not use
the operating system and application paradigm that personal computers use. The
particular microcontroller selected for this project is the Z-World SmartStar Modular
Computer system (Z-World, SR9000, Davis, California). This is an expandable system
which we configured by adding expansion cards containing digital to analog outputs
and analog to digital inputs. This particular unit was chosen at the beginning of the
project to avoid the expense of having to buy a new controller if there were unexpected
interface needs.

The SmartStar unit ran the control program for the machine, which included several
different operations. The user interface was a text menu which was displayed on the screen of a laptop connected to the system. This allowed the user to select from the options:

- Get Position
- Record Path
- Print Path
- Navigate Path
- Change PID Gains

Each of these options executed a subroutine in the control program. "Get Position" took the sensor input and prints directly to the screen. "Record Path" was used to store a set of points to use as a path for navigation. "Print Path" displayed the points that made up the current path for the operator. "Navigate Path" drove the machine along the path from beginning to end. "Change PID Gains" allowed the user to set the parameters for the proportional-integral-derivative controller which was used to control the actuator outputs. All of these operations were performed by the program running on the SmartStar.

The choice of a laptop for user interface was a matter of convenience and the choice of research focus. Developing an economical, concise and intuitive user interface for an autonomous machine is an important problem that will have enormous impact on the general acceptance, utility and marketability of autonomous machines. However, the need for a user interface only occurs if there is a functional autonomous machine. In this project user interface development has only been done to the extent necessary for experimentation. In a later phase of the project research will be focused on creating a high quality user interface.

The SmartStar was connected to the drive actuators using analog outputs. Since the actuators are controlled by varying their input voltages between 2 and 4 volts, the control program simply set the desired voltage on the corresponding analog channel. The sensors were connected to the controller on RS-232 serial ports, and the laptop used as the user interface terminal was connected on the RS-232 programming port.
A second microcontroller was used in the system as an input filter and control for the electronic compass unit. The need for this and the program developed for it will be discussed in the next chapter.

2.5.3 Choosing the Sensors

The sensors used for controlling the machine’s position and orientation were the source of the biggest challenges in the project. There are a limited number of technologies for position and orientation sensing for autonomous machines operating outdoors. Within the different sensing technologies there are very low cost sensors which are generally not useful for autonomous vehicle control, moderate cost devices which may be used and high cost devices which have the highest precision currently available. Since the success of the project is not measured merely by its navigation ability but also by its overall cost it was desirable to get the most effect out of low cost devices as possible. A variety of sensors were used for the various stages of the project to study the performance. Two major methods, laser positioning and machine vision, will be discussed here, and a third method, differential global positioning (DGPS) will be discussed in the following chapters. All three of these methods were identified at the beginning of the project as possible solutions separately and in combination.

The early development of this project used the laser positioning system (LPS) because this system had not previously been seen in agriculture. Machine vision and GPS systems have been used for autonomous vehicles developed by other researchers and have demonstrable success for this application. The objectives of this phase of the project were to develop an autonomous machine; demonstrate line following behavior; demonstrate simple pattern following behavior; and to determine the viability of the laser positioning system in agriculture.
2.5.4 Navigation With a Laser Positioning System

The system chosen was the SICK Nav2000 Laser Positioning System (SICK AG, Nav2000, Waldkirch, Germany). In this system a combined laser transmitter and receiver unit was mounted on top of the machine’s sensor mast. This unit has a turret which rotates at 360 RPM and transmits a non-visible laser. The laser is reflected off of reflectors mounted on posts arranged in an asymmetric pattern at the perimeter of the work area. The reflected laser beams are received by the unit and the position and orientation of the machine are measured using triangulation. A control unit located elsewhere on the machine was used to do the position calculations and interface with the machine’s computer system. Figure 2.2 shows the transmitter-receiver unit and the control unit mounted on the machine and two reflector posts in the background.

SICK, Inc. markets this system for autonomous forklifts in warehouses.

The reflectors were made of retroreflective tape and the laser must hit the reflector at a horizontal angle of less than $75^\circ$ and a vertical angle of less than $15^\circ$. Each reflector must be at least 2 inches wide and 6 inches tall. These must be mounted at the same height as the laser unit, which is difficult when working on uneven terrain. The proposed solution to this difficulty was to make the reflectors larger than necessary by making them 24 inches tall. This solution was not ideal because the reflector material is expensive enough to make the entire system unfeasible. A further difficulty was that the sensor cannot detect any reflectors beyond a distance of 30 meters. On an agricultural scale this is inadequate.

The laser positioning system uses its own coordinate system which is established by mapping the work area. It is an absolute position sensor because it reports a position which is unique within the work area, but it does not have the large scale position sense that GPS has. To map an area the machine is placed in a location where all the reflectors for the work area are visible to the laser sensor. The mapping function is executed and after several minutes of sensing the machine builds a map of the area based on all the reflectors it can sense. This map is a cartesian plane with its axes set
by the orientation of the laser scanner during the mapping process. The coordinates on the map are x and y distances in millimeters and a direction of travel in degrees, with zero as the direction the scanner was pointed during the mapping process. Since this uses a cartesian representation the navigation algorithms can be simple, based on ordinary plane geometry.

The progress of software development for use with the LPS was hampered by a peculiar problem with the RS-232 communication between the LPS and the SmartStar. The SmartStar system is capable of implementing RS-232 using either a 5-wire connection or a 3-wire connection, and it was desirable to use the 3-wire connection to the LPS, which has no need of the RTS and CTS control signals. However, when these devices were connected they would not communicate unless a jumper sequence was executed. This was extremely worrying for the future of the project until it was finally diagnosed as a faulty RS-232 chip on the SmartStar. The SmartStar was replaced with a new unit.

The software developed for the robot took the form of the second navigation method described in section 2.3.2 above. This method was successful in navigating straight lines with deviations of roughly 0.5 to 0.7 m, but did not produce successful pattern navigation. The problem was that when executing a spin or very tight turn the laser scanner’s measurements were disrupted and it gave bad data. When the laser system cannot detect enough reflectors it stops sending data, and if this persists without reacquiring the reflectors its software will reset. This occurred frequently when the machine executed spins or tight turns. To solve the bad data problem an electronic compass was added for a rotation sensor.

The electronic compass is a unit manufactured by Honeywell, the HMR3200 (Honeywell International Inc., HMR3200, Morristown, New Jersey). This is a three-axis, two-dimensional magnetic sensor with a data processing unit that transforms the magnetic information into degrees of magnetic heading. This data was also transmitted to the computer over an RS-232 serial connection. The compass unit was mounted in a small box fixed atop the box containing the SmartStar and its supporting hardware.
The system was configured to combine this sensor with the LPS data, transforming the LPS direction information to the more universal compass style angular measurement through the navigation software. The navigation used the LPS data for traveling along straight lines and using the compass for turns. This improved the system's reliability, but could not compensate for the LPS resetting unexpectedly. No solutions to this problem were found because the problem was fundamental to the operation of the LPS unit.

The ZTR platform was easily adapted to autonomous control, but the laser positioning system used was not sufficiently robust for agricultural environments. Even so, some successful navigation was demonstrated: the machine could travel along straight lines with little deviation. The results were never measured, only evaluated qualitatively by visual observation. The laser positioning system was unsatisfactory because its data was too frequently erroneous or disrupted. Factors which contributed to failures of the LPS included

- Spin or tight turn maneuvers
- Vibration
- Changes in ambient light
- Debris on reflectors

The laser positioning system was also unsatisfactory because it requires reflectors permanently installed at the perimeter of its workspace and it must always be within 30 meters of 4 of those reflectors. Finally, the system update rate seemed to be too slow at times. SICK does not provide the update rate since the device must be polled for data, but the highest rate achieved by the navigation program was around 5 Hz. For these reasons the Nav2000 was deemed inadequate as a single sensor for autonomous agriculture, and did not seem to offer significant benefit to a combination of sensors because of its complexity, expense, and lack of robustness.
2.5.5 Navigation With Machine Vision

Having demonstrated some autonomous navigation with the LPS the next step was to improve the navigation by adding a new sensor. A relative position sensor was used because it was desirable to demonstrate the ability of the machine to follow a crop row. Machine vision as a sensing technology has become more effective and less expensive in recent years. A machine vision sensor is an ideal sensor for following a crop row. This is a synopsis of work performed and documented by Spencer. [12]

The machine vision sensor was the DVT Legend 542C (DVT, 542C, Duluth, Georgia), which is a self-contained camera and image processor. The camera captures images and the image processor measures features in those images to generate output. For the case of a row following robot, which is considered here, the features of interest in the image are the plants that make up the row. Plants have very high NIR reflectance compared to soil so images of plants and soil captured in the NIR band will have very high contrast, with high brightness in pixels that subtend plants, and low brightness in pixels that subtend soil. This makes the machine vision algorithms straightforward. We can also assume that the plants will be in a row, and that when the machine is on the correct path the camera will be directly over the plants. This last assumption may be changed to suit different navigation requirements; it merely establishes the specifics of the calculations.

The image processing needs are established from these assumptions. The camera was programmed with a manufacturer supplied software tool called FrameWork, which compiled the necessary functions to the processing hardware. FrameWork was used to generate processing which would take the input image and scan in a vertical line across the image from each side, looking for transitions from lower brightness to higher brightness. Two lines were generated from this, by doing a least-squares fit of the transition pixels from the left, and the same for the transition pixels from the right. The authenticity of these lines is verified by comparing them to each other and considering the reported location with respect to the mower; the sensor does not send back data if
either line is outside of reasonable limits. The offset and angle values of the two lines are averaged to estimate the respective offset and angle of the row.

The DVT camera was mounted on the front of the electronic equipment tray at a height of 1.1 meters and an angle of 60 with respect to the vertical. The camera mounted on the robot can be seen in Figures 2.3 and 2.4. The mount was not vibration isolated from the vehicle frame. The zoom, which is manually controlled, was set so that the camera captured an image about 1 meter² immediately in front of the vehicle. This sensing area is relatively small, and in areas of poor plant density the system would have to consult other sensors to navigate successfully. Power was supplied with two twelve volt cells in series, and the camera was connected to the Smart Star with an Ethernet cable.

Working with the machine vision system required only slight modifications to the code, primarily in acquiring the data via the Ethernet port rather than the serial port as had been done previously. The navigation code was modified only slightly from the code developed for the LPS. Navigation using the combination of crosstrack and angular error and a PID controller was highly successful. Some adjustment to the PID parameters used for the LPS system was necessary, and the classic Zeigler-Nichols method was employed for this.

The performance of this system was very good, with reported results comparable to the driving abilities of a skilled human operator in controlled environment tests. It is expected that machine vision will be a primary means of navigation on future agricultural automata, but a machine cannot operate autonomously using only relative position sensors. For this reason a third stage of the project was established, to develop high quality absolute position based navigation.

2.6 Summary

The development of the project to this point was done over three years, and can be divided into three segments based on the sensors used on the machine. The early
development of the machine used the laser positioning system. This was not particu-
larly successful. The second body of work was done using the machine vision system. 
This work drew upon much of the knowledge gained from the earlier development and 
benefited from that work, but was largely unrelated to it because the type of control 
done and the software written for that was very different than the first efforts. This 
produced the very successful results described by Spencer. The work described in the 
next chapters was done using a differential global position system (DGPS) receiver 
and included elements from both of the earlier efforts, but is more closely related to 
the early work using the laser positioning system because it relies on absolute position 
measurement.
2.7 Figures
Figure 2.1: Visual Representation of Crosstrack and Angular Error: (a) Crosstrack Error; (b) Angular Error
Figure 2.2: Front View of the ZTR Robot Indicating Laser Positioning Subsystem
Figure 2.3: Front View of the ZTR Robot Indicating Machine Vision Subsystem
Figure 2.4: Side View of the ZTR Robot Indicating Machine Vision Subsystem
CHAPTER 3
Implementation

The project began by acknowledging that the SICK laser would not satisfy the requirements for precision agriculture. The machine vision system demonstrated the capability of the robot with a good sensor, but absolute position guidance was still lacking. The previous research and the progress of technology and economics all point to using the global positioning system as the main absolute position sensor for an agricultural robot. A GPS unit was selected to replace the laser positioning system and software was developed to use the new technology. The software includes some routines from the old software, refined and adjusted for the new system. The software development process went through these stages: sensor communication; actuator control; and navigation development and fine tuning. A variety of difficulties were confronted during this process and various solutions were applied. During the fine tuning part of the process the system was tuned holistically by striving to optimize the straight line following ability of the machine.

3.1 Changing the Sensors

The absolute position sensor used for this part of the project was the Trimble AgGPS 132 (Trimble Navigation Limited, AgGPS 132, Sunnydale, California), which is a moderate-cost differential global position receiver. The antenna for this unit was mounted on the mast originally used for the laser, as shown in Figure 3.1, and the receiver and display unit was mounted on the equipment tray, as shown in Figure 3.2. Given what was learned from using the laser positioning system the following requirements were identified for absolute position sensors. These were the criteria for selecting
the DGPS receiver.

- Sampling Rate of 5-10 Hz
- $2\sigma$ error 20-30 cm
- Robustness to high slew rates and substantial vibration
- Cost not more than $5000
- Communication via RS-232 or Ethernet

We felt from our prior work and research that these requirements were the bare minimum to achieve successful autonomous navigation. This list outlines a particular segment of the present GPS market, DGPS receivers using subscription based satellite delivered corrections. For our purposes the functionality of such receivers is essentially standardized. Manufacturers offer many different features and options such as waypoint and mapping systems, but our software only needs the position data, so one system is much the same as another. The AgGPS 132 was chosen because it was already available in the N.C. State BAE department and there was significant department experience with the unit. The navigation software will work with any type of GPS which can transmit the NMEA 0183 RMC sentence over RS-232, however the navigation performance will be affected by the quality of the position data.

We selected the OmniStar subscription service for our differential corrections. OmniStar is a common subscription service for agriculture and offers three different levels of service. The AgGPS 132 can only use the lowest level of service, OmniStar VB (Omnistar Incorporated, Houston, Texas). When using this differential correction the specifications for the AgGPS 132 give a $2\sigma$ error of less than 1 meter. This does not meet specification 2 above, but prior experience with the receiver suggested that it was capable of better performance than Trimble’s specifications state. Since it fits the other specifications well and is a popular DGPS receiver for precision agriculture it was desirable to learn if it is possible to develop autonomous navigation using this unit.

All the literature on using GPS for navigation uses either CP-DGPS or RTK-GPS, no results have been reported using the less expensive differential GPS. Thus it is anticipated that a higher level of precision may be required for optimal performance.
However, Trimble does market the AgGPS 132 as a receiver which can be used with assisted navigation systems.

GPS receivers can calculate ground speed and heading or direction of travel when in motion. These calculations are done based on the difference between two positions, so the results are only as good as the position data. The directional information in particular is not reliable for tight or fast turns. For this reason the electronic compass unit used with the laser positioning system was retained for use with the GPS.

3.2 New Software

All the software for the system was developed in Z-world’s proprietary programming language, Dynamic C (Z-World, Dynamic C 9.21, Davis, California). This programming language is an extension of the ANSI C specification embedded in a development environment and packaged with numerous libraries and examples for common embedded systems functions using Z-world’s microcontrollers.

3.2.1 GPS Data Input

The first challenge was transferring the GPS position data into the control computer for navigation. The standardized communication protocol for GPS units is the National Marine Electronics Association (NMEA) 0183 sentence system. This standard was originally developed for LORAN aided navigation on the ocean and modified to work with GPS. It specifies several different sentences which can be used to communicate position and various other data including time, speed, heading, or number of satellites used to compute the position. For this project the position information was the most valuable, but it seemed that some other information might also be useful, such as the heading, speed, and magnetic declination. Therefore the RMC sentence was selected to communicate the GPS data since it includes all this information. The format of the RMC sentence is:
An example of an RMC sentence is:

$GPRMC,184804.00,A,3723.4765,N,12202.2397,W,000.0,0.0,051196,15.6,E*7C

Table 3.1: Explanation of the NMEA RMC Sentence

<table>
<thead>
<tr>
<th>Field</th>
<th>Variable</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$GPRMC</td>
<td>Sentence identifier</td>
</tr>
<tr>
<td>2</td>
<td>hhmmss.ss</td>
<td>Time</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>Status: 0-No GPS; 1-GPS; 2-DGPS</td>
</tr>
<tr>
<td>4</td>
<td>dddmm.mm</td>
<td>Latitude in degrees and decimal minutes</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>Latitude direction: N or S</td>
</tr>
<tr>
<td>6</td>
<td>dddmm.mm</td>
<td>Longitude in degrees and decimal minutes</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
<td>Longitude direction: E or W</td>
</tr>
<tr>
<td>8</td>
<td>x.x</td>
<td>Speed over ground</td>
</tr>
<tr>
<td>9</td>
<td>x.x</td>
<td>Heading or Course over ground</td>
</tr>
<tr>
<td>10</td>
<td>ddmmyy</td>
<td>Date</td>
</tr>
<tr>
<td>11</td>
<td>x.x</td>
<td>Magnetic declination</td>
</tr>
<tr>
<td>12</td>
<td>a</td>
<td>Direction of magnetic declination: E or W</td>
</tr>
<tr>
<td>13</td>
<td>*hh</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

The RMC sentence is transmitted as a string of ASCII characters over the RS-232 serial connection. RS-232 is a very flexible communication standard. It allows a variety of different configurations of the baud and the composition of the data packets. For this project high speed is desirable, so we chose the highest baud the SmartStar is capable of handling, 32,000 bits per second. The standard 8 data bits, no start bit and 1 stop bit (8N1) data packet was used. The AgGPS 132 continuously outputs position updates at a selected update rate, offering options of 10 Hz, 5 Hz, 2 Hz or 1 Hz. The 1 Hz update rate is standard, and an upgrade code must be purchased to access higher update rates. For this project the 10 Hz update rate was used.

Inside the SmartStar the ASCII characters are not of much use because it was
desirable to manipulate the values mathematically, so numerical values were needed. Since there are several different numerical values related to a single position, i.e. Northing, Easting, Heading, and Speed, a data structure containing these variables was created as a part of the software. This data structure is called a "GPSPosition", and it contains the following variables (variable names in parentheses):

- Northing (northing)
- Easting (easting)
- Heading reported by the electronic compass (compassHeading)

When the SmartStar received an RMC sentence from the GPS the sentence was converted from the string of characters into numerical values and stored in the appropriate variables so that the SmartStar could use the information. The string of characters is comma delimited, so parsing the individual values out of the string was done by counting the commas, and the position of a value in the sentence corresponds to the type of data the value represents. To extract the string of characters from a field in the sentence the program simply traversed along the sentence until encountered a comma, and then it copied the subsequent characters up to the next comma in the sentence into a temporary string. Once the temporary string of number characters was extracted from the sentence it must be converted into a floating point number so that the SmartStar can manipulate it mathematically. This was done by aid of a Dynamic C math library function called "atof", which uses a string of ASCII characters as input and returns a float as output. By this method all of the data of interest in the sentence was extracted and saved as numerical variables.

Since the GPS transmits position information in Latitude and Longitude the values were converted to State Plane coordinates using the SmartStar. This conversion was performed as a step in the sentence parsing process. The function to do the conversion was written to use latitude and longitude stored in temporary variables and directly store the northing and easting values in the data structure. This function simply implemented the equations listed in section 2.2.1 using the constants from Table 2.1. Since the sine, exponent and natural log functions are defined for floating point mathematics
in the math library included with Dynamic C the equations were implemented quite easily. However, when this was done the state plane coordinates were not calculated correctly. This revealed a significant problem with the Z-world microcontrollers.

This problem arises because calculating position to millimeter precision requires that the constants used in the calculations (see Table 2.1) be specified to 12 significant places. The computer representation of the value must be able to manipulate numbers with 12 or more significant places, which requires a 64 bit floating point data type. This data type is usually called a "double," referring to the fact that it is twice the size of an ordinary floating point variable. This was a problem with the Z-world microcontrollers because they do not have a native 64 bit data type. This was a significant challenge to the design process. Some different solutions were considered: a different microcontroller with a 64 bit data type could be used as an input processor to parse the data from the GPS and calculate the SPC coordinates, passing those on to the SmartStar as 32 bit floats; software floating point methods could be written for the SmartStar; or a different microcontroller could be used for the whole control system. These options were not particularly desirable. The input processor idea had merit and was ultimately used for this and for the compass input system to solve a problem with that system as well. The software floating point programming was started, but quickly exceeded the programming abilities available. However, investigation into software floating point solutions turned up a software library written for the Z-world controllers to do double precision floating point. This library was developed by an engineer with a similar need and offered for sale to the general public. This was a viable solution to the problem and saved much difficulty and expense in the development of GPS input.

After obtaining the floating point library and implementing it in the control code it was decided that adding a microcontroller dedicated to processing the GPS input would streamline the control code and minimize the effect of the SmartStar’s lack of floating point precision. A Jackrabbit (Z-world, BL1820, Davis, California) microcontroller was chosen because of its low cost, availability and convenience of programming. However, it has the same lack of floating point precision the SmartStar has, so it was still
necessary to use the double precision floating point library. The software for parsing the RMC sentence and converting the coordinates to SPC coordinates was converted to work on the Jackrabbit and removed from the SmartStar. The Jackrabbit has two serial ports called Port B and Port C. The GPS was connected to Port B and Port C was used to connect the Jackrabbit to the SmartStar. The Jackrabbit was also used to filter the position data using a 3 term running average to remove some signal noise due to vibration. The Jackrabbit transmitted the SPC position data, which can be represented as a 32 bit float, as a packet ten times per second, and the SmartStar retrieved the information from its serial port and converted the packet into 32 bit northing and easting values. The Jackrabbit transmitted the SPC position data, which can be represented as a 32 bit float, as a packet ten times per second, and the SmartStar retrieved the information from its serial port and converted the packet into 32 bit northing and easting values. The data packet format begins with a ‘$’ character, followed by the four bytes of the northing value, then a ‘*’ character and then the four bytes of the easting value. The characters were used to identify and parse the input values, so that there was no confusion between northing and easting, and also to help prevent data corruption. The serial ports were all configured to transmit standard 8 data bit, no parity bit and 1 stop bit (8N1) data packets, at 38,400 bits per second. The software developed for the Jackrabbit GPS input processor is included in the appendices of this document. This program was called mowergps.c, and the main navigation code was adapted to read the data directly from the Jackrabbit.

### 3.2.2 Compass Data Input

The Honeywell electronic compass was retained as a secondary sensor for use with the GPS data because heading information was used in the navigation method and the GPS does not generate reliable heading information when it is not moving or when the machine executes a spin or tight turn. As shown in Figures 3.1 and 3.2, the physical mounting of the unit was not changed in this implementation. However, some changes were made to the way that the data is transferred to the SmartStar to improve the performance of the sensor.

The compass is a continuous output serial device like the GPS. It output the ma-
chine’s current heading as a string of ASCII characters followed by a carriage return character and a linefeed character. The heading is measured to one decimal place, so the output can range from 0.0 to 359.9 and the output strings corresponding to those two values are "0.0<cr><lf>" and "359.9<cr><lf>.” An input parsing technique similar to that used with the GPS data was implemented with this system. The serial input was traversed, adding characters to a temporary string until the decimal point was reached, adding the decimal point and the next character to the string, and converting that temporary string to a floating point value using the ”atof” function which converts a string of ASCII characters into a floating point number. This function is supplied with Dynamic C in the standard math library. However, the compass update rate is 10 Hz, and if the serial input is collected at a rate less than 10 Hz there tends to be a data mixup. This caused various problems with erroneous data.

This problem was corrected by adding a Jackrabbit (Z-World, BL1820, Davis, California), an inexpensive Z-world microcontroller, between the compass and the Smart-Star as an input management processor. The Jackrabbit ran freestanding software that read the compass input at a rate higher than 10 Hz on one serial port and was connected to the SmartStar on a second serial port. Both serial ports were configured to operate at 19,200 baud, transmitting the standard data package of eight data bits, no parity bit and one stop bit (8N1). The software was configured to keep a running average of the last 3 heading values and be polled by the SmartStar. This means that when the compass data was needed the SmartStar sent a single ‘!’ character to the Jackrabbit. When the Jackrabbit received that character it multiplied the current running average value by 10, converted that value to ASCII characters, packaged it into a string prefixed with a ‘$’ and suffixed with a ‘*’ and transmitted the string of characters to the SmartStar. As an example, if the compass measured the heading 345.6° then it would send the string of characters ”345.6<cr><lf>” to the Jackrabbit, which would parse this, convert it to a number and combine the number with the running average to produce an average heading of, for example, 346.1°. When a ‘!’ character was received by the Jackrabbit from the SmartStar the Jackrabbit would transmit the string
On the SmartStar side, the compass was polled every time the GPS input was parsed, so that every time a position is recorded the heading is recorded at the same time and stored in the same GPSPosition structure. The SmartStar would execute a function called "get_heading" that outputs a '!' character on the appropriate serial port and waits for the string of characters preceded by a '$' and followed by a '*'), removing those control characters as it stores the values in a temporary string. When the temporary string was complete it would convert it to an integer value and then divide it by 10 and store it as a floating point value to get the original transmitted value. This conversion and transmission system may seem complicated, but it simplifies the process of parsing the input and rejecting bad data. This is why the data string uses a unique character at the beginning and the end.

The code developed to implement these processes appears in the appendices of this document. The program running on the Jackrabbit is mowercompass.c and the SmartStar code related to the compass input is in compass.lib.

### 3.2.3 Navigation Code

**Path Input**

For line navigation between points using GPS the minimum information needed for each point is the coordinate pair that defines the point and position of that point in the path, described by numbering the points sequentially. So a path might be made by storing an array of northing and easting coordinates in the order they are to be navigated. In this case the navigation code must calculate the direction from one point to the next, expressed in degrees from magnetic north, which involves trigonometric math and condition checking, and therefore uses several processor cycles. So, the navigation process was streamlined by calculating the directions between subsequent points in the path when the path was first input into the system, since the direction from one point to the next is a fixed value. This requires storing the direction value to be stored for each
path point, which adds an additional variable per point named "directionOfTravel." When the path input process was finished a function called "calc direction of travel" performed the direction calculations and stored the direction of travel from point 1 to point 2 in the "directionOfTravel" variable belonging to point 1 and so forth for all subsequent points. In the navigation software each point is a data structure called a navPoint which contains four variables (variable names in parentheses):

- Northing (northing)
- Easting (easting)
- Direction from this point to the next point (directionOfTravel)
- A flag indicating if the machine should stop at this point (stopFlag)

The path itself was an array of these points. This was a fixed array and its size was set at 500 points, although the SmartStar is capable of handling ten thousand points if need be. A fixed array was used because of its simplicity and reliability. An array of fixed length will always occupy a known amount of memory and access to a fixed array can be managed through the index of the array. The index also provides a built in ordering mechanism. The alternative would be a dynamically created linked list, but this mechanism does not have any protection from consuming all available memory if the path is too large. Linked list access methods are also more complicated than a simple array and require more code development. The machine was configured to hold one path. It is likely that in the interests of convenience and utility a commercial version of the machine will need to store several paths onboard or have a quick and convenient download system so that several paths can be used repeatedly without effort or time expended on repeated path input.

Calculating the direction of travel between two points on a cartesian plane can be done with ordinary trigonometric functions. However, these functions are based on the standard geometric angle definition which is shown in Figure 3.3b, which is significantly different from the compass angle definition shown in Figure 3.3a. In order to obtain the proper compass direction the result must be transformed by flipping and rotating $\pi/2$ radians. The equations used for obtaining the direction from point $A$ to point $B$
in radians are:

\[ N = B_{\text{northing}} - A_{\text{northing}} \]  \hspace{1cm} (3.1) \\
\[ E = B_{\text{easting}} - A_{\text{easting}} \]  \hspace{1cm} (3.2) \\
\[ \theta = \text{atan2}(N, E) \]  \hspace{1cm} (3.3)

In 3.3 atan2 is the double sided arctangent, which ranges from \(-\pi\) to \(\pi\) instead of the ordinary arctangent which ranges from \(-\pi/2\) to \(\pi/2\). When this is converted to degrees by the degrees-to-radians relationship \(180/\pi = 1\) it can be seen in Figure 3.3b that a transformation is needed to convert this into a system equivalent to Figure 3.3a.

The first step in such a transformation is to reverse the sign of the result. This can be done by changing the sign on the \(N\) value. This will yield the results seen in Figure 3.3c, where the values increase clockwise instead of anticlockwise as in Figure 3.3b. Then, to change the scale to a 0 to 360 scale add a constant to the result. Adding 90 degrees provides the correct solution in 3 quadrants, as seen in Figure 3.3d. However, the quadrant where \(N \geq 0\) and \(E < 0\) still ranges from \(-90\) to 0. So in this quadrant we must add an additional 360 degrees, for a total of 450 degrees. This yields the results shown in Figure 3.3e. The equations used to achieve this result are:

\[ \text{neg}N = A_{\text{northing}} - B_{\text{northing}} \]  \hspace{1cm} (3.4) \\
\[ E = B_{\text{easting}} - A_{\text{easting}} \]  \hspace{1cm} (3.5) \\
\[ \phi = 180/\pi \ast \text{atan2}(\text{neg}N, E) \]  \hspace{1cm} (3.6)

Direction = \begin{cases} 
\phi + 450 & \text{if } \text{neg}N \leq 0 \text{ and } E < 0 \\
\phi + 90 & \text{otherwise} 
\end{cases}  \hspace{1cm} (3.7)

These equations were used to implement the direction calculation. This was done in the function "calc_direction_of_travel" which would be executed after the path has been input into the program. The direction of travel from the starting position to the first
point in the path must be calculated at beginning of navigation. The direction of travel for all other points was calculated by this function. The Dynamic C code for the function can be found in the main navigation code appended to this document.

The stop flag was used to control the machine’s behavior during navigation. When progressing along a path, the machine could either stop at a navigation point or continue directly to the next point. The machine was designed to stop at any point where it must rotate to reach the next point. If the difference between the direction of travel from point A to point B and the direction of travel from point B to point C is greater than 12° then the machine would stop at point B to rotate toward point C. The machine must also stop at the last point on the path. This behavior was controlled by the stop flag associated with each point. If the stop flag for a point was set to 1 the machine would stop at that point and rotate. If the stop flag was set to 2, indicating that this is the last point in the path, the machine would stop and wait for a new command. If the stop flag was set to 0 then the machine will proceed to the next point without stopping. Because of the way the stop flag was defined the stop flags for a path were set when the path was created and the direction of travel to each point was calculated. Again, the first point in the path is an exception, its stop flag must be set at the beginning of navigation, when machine’s starting position is known.

As discussed in section 2.3.3 there are two types of path entry methods which were considered desirable for the autonomous machine: recording the path a human operator used to do the task; and creating a path from an input file generated by a GIS system. For the development of the navigation program only the simpler input method, recording, was implemented. The recording process was implemented by having the user select the record function, and then press a button when he is ready to record. Then the SmartStar would record the northing and easting coordinates from the GPS input every 2 seconds until the button is pressed again. At the end of the recording process the "calc_direction_of_travel" function would be executed and the stop flags set.
Navigating A Path

Once a path has been created the user can execute the path navigation function, which drives the machine from its current position to the subsequent points of the path. This is a complex function which must perform many actions, calling various helper functions. The function does some preliminary calculations and then enters a "while" loop which controls the path, handling rotation to start a path segment and calculations related to that segment and then enters a while loop which controls navigation along the segment. This is the path segment navigation loop, which does sensor update, control calculation, and steering. The combination of these loops controls the autonomous function of the machine.

The preliminary calculations needed for navigation were all executed before the machine starts moving. These functions only need to be executed once to navigate the path. First the path index variable was initialized. Then the machine position data was collected from the sensors and stored as a navPoint so that it can be used to navigate to the first point of the path. Then the direction of travel to the first point of the path and the stop flag for that point are set.

The path navigation loop sets up the machine to navigate the next path segment and executes the command to start moving before starting the path segment navigation loop. This loop contains the functions which must be executed each time the machine arrives at a point where it must rotate at the beginning of the next path segment in order to proceed along that path segment. These functions are the rotation and calculating the constants used in the crosstrack error calculation in that segment. Then the actuators are set to make the machine go straight forward and the path segment navigation loop is started.

The rotation function puts the mower through a zero or near zero radius turn to point it along the path segment between two points. This is done by measuring the angular error (the difference between the direction of the path segment and the current heading of the machine), and executing a turn with turning speed proportional to that
angular error. The actuators are set so that one wheel turns forward and the other
turns backward to achieve the very small radius turn. The actuator on the wheel that
turns forward is set to a fixed position, so that the wheel turning forward always turns
at the same speed. The actuator on the wheel that turns backward is adjusted each
time the angular error is measured so that its speed decreases as the angular error
decreases, and therefore the rotation speed of the machine decreases as the angular
error decreases. The functions that control this proportion are different for each wheel
because the actuator position for a given wheel speed is slightly different on each side.
The general form of the equation is:

\[ V_{\text{out}} = E_{\text{r}}\text{r}_{\text{angular}} \frac{(180 - 0)}{(V_{\text{max}} - V_{\text{neutral}})} + V_{\text{neutral}} \]  

- \( V_{\text{out}} \) is the analog output voltage to the actuator
- \( V_{\text{max}} \) is the voltage corresponding to maximum forward wheel speed
- \( V_{\text{neutral}} \) is the voltage corresponding to zero wheel speed

It was not necessary to make the angular error zero for successful navigation. The
program was configured to stop the rotation when the angular error was less than 10
degrees. It should be noted that the angular error was signed to indicate direction,
and this sign was the means by which the software determines whether it should turn
to the left or the right. A positive sign indicated that the machine must be turned
to the right to decrease the angular error, and a negative sign meant that it must be
turned to the left. The angular error calculation was also adjusted so that it fit in the
range -180 to 180. This was done simply by adding 360 to any value less than -180 or
subtracting 360 from any value greater than 180.

The crosstrack error calculation has been discussed in section 2.3.1. It was observed
that for a given path segment the equation contains several constant values. For the
navigation program this equation was simplified to:

\[
Error_{xtrack} = \frac{NtoDestination(B\text{east} - C\text{east}) - EtoDestination(B\text{nor} - C\text{nor})}{\text{segmentLength}} \tag{3.9}
\]

Where \(A\) and \(B\) are the endpoints of the line, and represent the previous and current points in the path array respectively. In the program these points are referred to as path[pathIndex-1] and path[pathIndex]. The three values \(NtoDestination\), \(EtoDestination\), and \(\text{segmentLength}\) are all constants. The program computes these constants after rotating, using the following equations:

\[
NtoDestination = (B\text{northing} - A\text{northing}) \tag{3.10}
\]

\[
EtoDestination = (B\text{easting} - A\text{easting}) \tag{3.11}
\]

\[
\text{segmentLength} = \sqrt{(B\text{northing} - A\text{northing})^2 + (B\text{easting} - A\text{easting})^2} \tag{3.12}
\]

Since these constants are specific to the path segment each time the machine arrives at a point and increments the path index these constants must be recomputed. The line segment from the machine’s start position to the first point of the path is a special case because there is no prior point to use as the start point for the line segment. The start position is used as point \(A\), so for that situation a slightly different form of the constant calculations is used. The crosstrack error calculation was also signed to indicate left and right. A positive sign indicated that the machine was on the right side of the line.

When the rotation is completed and the constants have been calculated the machine is set into motion by setting the actuators so that it begins traveling forward. The path segment navigation loop is started, which controls the machine as it travels along a path segment. This is the loop which calculates the angular and crosstrack errors, calculates a correction based on those errors and changes the actuator positions based on that correction. It also calculates the distance to the current target point and evaluates the conditions for changing to the next point.
The navigation was controlled by minimizing angular and crosstrack errors. When these errors were calculated as described above they were combined into an aggregate error according to the following equation.

\[
\text{Error}_{aggregate} = K_1 \text{Error}_{xtrack} + K_2 \text{Error}_{angular}
\]

The error values were scaled using coefficients \(K_1\) and \(K_2\) so that the aggregate error varies between -1 and 1. These coefficients were sized according to the maximum allowable errors:

\[
K_1 = \frac{0.5}{xtrack_{max}}
\]

\[
K_2 = \frac{0.5}{angular_{max}}
\]

This was designed so that in the situation that the machine is at its maximum crosstrack error and turning toward the line at its maximum angular error the aggregate error will be zero. Using a control law which was designed to reduce the aggregate error to zero, the steering effort would be proportional to the crosstrack error in that situation. The values for \(K_1\) and \(K_2\) were adjusted slightly to tweak the performance. The final values used are \(K_1 = 1.02\) and \(K_2 = 0.0684\). The value for \(K_2\) was reduced from the original calculated value because the mower responded too aggressively to angular errors, and the final result was half the original calculated value. This yielded smoother responses.

The control law used to reduce the aggregate error to zero is the traditional Proportional + Integral + Derivative controller. The PID code used was taken from sample code supplied with Dynamic C. The algorithm implemented is the standard discrete PID algorithm:

\[
\text{correction} = P \text{Error}_{aggregate} + I \Sigma \text{Error}_{aggregate} + D (\text{Error}_{aggregate} - \text{lastError})
\]

Where \(\Sigma \text{Error}_{aggregate}\) is a cumulative sum of the errors and \(\text{lastError}\) was the \(\text{Error}_{aggregate}\) value for the previous calculation. The parameters \(P\), \(I\) and \(D\) must be adjusted to
achieve the desired performance of the system. It would be desirable to use the same values for the parameters that were used for controlling the machine with the machine vision sensor, however the results were substantially different. For this system the best PID parameters are $P = 0.09$, $I = 0.004$, and $D = 0.002$. The parameters were found by systematic experimentation following the procedure described in [21].

By this method, the machine proceeds along a straight line between the starting point and the ending point of that path segment. When the end point is reached the machine checks the stop flag for that point, and if it is not set the machine increments the path index and calculates new constants for the crosstrack error calculation, and continues without stopping. If the stop flag is set the actuators are set to the neutral position and the path segment navigation loop is ended, and the machine rotates toward the next point to begin again, or stops altogether if the final point in the path has been reached. Arrival at the final point was considered successful if the machine was within 0.4 meters of the final point. This allowable error gave the system some leeway and prevented unnecessary human intervention, since the machine stops if its navigation is not successful. The value 0.4 meters was determined as a desirable goal for this stage of development based on the size of the machine, since the machine’s footprint is about 1 meter$^2$ so if its center is 0.4 meters from the final point then the point must be under the machine.

### 3.3 Performance Measurement

A line stretched between two points served as the reference when evaluating the performance of the machine in straight line travel. For tuning the navigation program and its various parameters the performance was measured visually by observing the position of the reference line through a grate between the rear wheels of the machine. This provided a quick and coarse method of measuring the machine position relative to the line, allowing measurements of ±1 inch. In order to report results with sufficient resolution an RTK-GPS was used instead of the DGPS for the data collection portion.
of the project. The RTK-GPS was used for navigation and data collection by connecting it to both the navigation computer and a separate handheld computer with a data collection program on it. This will be further described in the next chapter.

The performance should be measured both in straight line travel and in navigating paths of moderate complexity, such as a pattern for mowing a non-rectangular area, or a pattern for a crop maze.
3.4 Figures
Figure 3.1: Side View of the ZTR Robot Indicating Sensor Subsystems
Figure 3.2: Rear view of the ZTR Robot Indicating Sensor Systems
Figure 3.3: Visual Representation of Angle Conversion: (a) Compass angle definition; (b) Cartesian angle definition; (c) First manipulation; (d) Second manipulation; (e) Final result
CHAPTER 4

Results

The system was tested and demonstrated on a rectangular grassy lot about 2000 meter\(^2\) in size adjacent to Weaver Labs on N.C. State’s main campus. One side of this lot has several very tall trees and two other sides have buildings, so this provided a very realistic test environment for a real world application. Specific tests were designed for the machine and used in development and testing. These tests were primarily concerned with the basic straight-line following ability of the machine, but some tests were also used to evaluate the limits of the machines abilities. In this chapter the machine’s performance will be described both qualitatively and quantitatively. This chapter will also include remarks on various attributes of the autonomous system such as environmental robustness, usability and adaptability for other equipment and tasks. All the test results described in this chapter were obtained using a Real-time Kinematic GPS (RTK-GPS) receiver as the position sensor for both navigation and quantitative recording.

4.1 Qualitative Observations

As discussed previously, the primary behavior of interest for this project is the ability to travel between two points along a straight line. Therefore this was the focus of much testing. Tests designed to evaluate this include:

- Navigating to a point 45 meters away;
- Navigating to a starting point and then to a point 45 meters from that point;
- Navigating a series of 3 points 15 meters apart along a straight line (45
meters total);

For all these tests a reference line made from a 6mm rope approximately 45.5 meters long was stretched between the starting and ending points for comparison. Tests were conducted over the course of the development process to assess the improvement of performance as individual components were altered or refined. At the end of the development process tests were conducted on several different days at different times of day over the course of two weeks. This repetition was intended to provide observations which include a variety of GPS conditions.

For these final tests the machine traveled from point to point successfully regardless of the distance traveled. The performance on the three tests described above was nearly identical from test to test. Informal tests with points at closer and farther spacings also demonstrated successful navigation. There was a minimum useful distance between points because the system could not reliably reach a point less than 2 meters away. The machine was able navigate to a given point following a straight-line path.

The machine was able to successfully complete all of the tests regardless of the starting heading. The system correctly rotated the mower through any angle to navigate to that point. This is an important feature because it enables the machine to treat complex shapes as a series of straight lines connected at arbitrary angles. The zero turning radius design is capable of a rotation which displaces the center of the machine only a few centimeters. However, the autonomous rotation will displace the center of the machine somewhere between a few centimeters and a meter because the castor wheels on the front of the machine rotate to permit the machine to spin, and a significant impulse is required to effect that rotation when the machine is stationary. This displacement is largest for small angles, between $10^\circ$ and $45^\circ$. This may seem counterintuitive, but the impulse required to rotate with little displacement is great enough to cause over-rotation for a small angle. An additional displacement will occur when straightening out the casters to permit the machine to travel straight again, and this displacement will have both crosstrack and along-track components. These turning displacements are acceptable since they on the order of 5-10 cm and only occur at
the beginning of a path segment.

The autonomous performance on a straight-line path was not as precise as a human operator. The machine appeared to over-steer, oscillating down the path rather than following a smooth curve. These results may be seen in the GPS output from a test run shown in Figure 4.2. The reasons for this excessive oscillation are not known, but further observations have informed the conjectures presented in this chapter. The first observation was that the computer’s representation of its position was not as precise as the GPS representation of position. The loss of precision was due to a limitation within the microcontroller, either its native inability to store 64-bit floating point numbers or a flawed calculation in the added library used to work with 64-bit values. This loss of precision was observed by moving the machine a short distance while monitoring the GPS data and the calculated SPC coordinates. Small changes in the GPS data were not reflected in the SPC coordinates.

A second observation was that the machine responded sluggishly to small steering inputs, but too aggressively to moderate inputs. The input range was reduced in response to this observation, but the improvement due to this reduction was slight. There may also be other delays in the steering response which contribute to the oscillatory behavior. Finally, it was observed that the system never displays any tendency to veer off in a completely unexpected direction, that is to say that despite the oscillatory response it always turns back toward the path it should be following.

The machine was able to navigate from point to point along a straight-line path, but it oscillated on that path with greater amplitude and frequency than is acceptable. The tires of the machine did not cross the reference line that marks the center of the path more than once per test, but approached the line multiple times per test. Ideally the center of the machine would be over the reference line for the majority of the time. Some tests also showed a slight bias toward the left, but this could not be reliably determined because of the oscillations. Quantitative data will further explain the performance of the machine and aid in the discussion of these results.
4.2 Quantitative Observations

The machine's position data was recorded using the RTK-GPS and a handheld computer with Farm Works Site Mate (CTN Data Source, Site Mate, Hamilton, Indiana) software. The RTK-GPS was used because it is the precision standard for GPS. The output from the GPS was used as the input for the control system as well as for recording the position. The Site Mate software was configured to automatically record the position data by recording one position per second.

All the data collection was done in one test session, spanning about 2 hours. The GPS signal quality was consistently high throughout this session. All the tests during this session were set up identically in the format described as test type 3 in the previous section. A reference line was stretched between two endpoints 45 meters apart. Those two points and two other points 15 and 30 meters along the line were used as the navigation points for the path. After allowing the RTK-GPS to warm up for 15 minutes the data recording system was used to record the positions of the navigation points in a separate data file for use as a reference in the analysis. The navigation point positions were also input into the control software. The coordinates of these points are shown in Table 4.1 below. A visual representation of these points is shown in Figure 4.1, which shows that the line runs from southeast to northwest. The exact heading of the line is 298° magnetic.

Table 4.1: Coordinates of Navigation Points Used for Testing, stated in meters according to the State Plane Coordinate System of North Carolina.

<table>
<thead>
<tr>
<th>Northing</th>
<th>Easting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>225576.541</td>
</tr>
<tr>
<td>2</td>
<td>225583.264</td>
</tr>
<tr>
<td>3</td>
<td>225590.837</td>
</tr>
<tr>
<td>4</td>
<td>225597.561</td>
</tr>
</tbody>
</table>
The tests were conducted by manually positioning the machine at point 1, the southeastern endpoint of the line with the machine’s direction of travel approximately along the line. The control program and the data recording was started and the machine navigated to points 2, 3 and 4 in succession. Six tests conducted during this session were recorded, with an average of 54 points per test.

When all the data were collected the tests were first examined visually for discrepancies using the ESRI ARCViev (ESRI, ARCViev 9.1, Redlands, California) GIS visualization program. No problems were observed. The data was recorded in Latitude-Longitude coordinates, so the Corpscon (U.S. Army Corps of Engineers, Corpscon 6.0.1, Alexandria, Virginia) coordinate conversion program was used to convert all the data into State Plane Coordinates. Then the coordinates were imported into Microsoft’s Excel (Microsoft, Excel 2000, Redmond, Washington) spreadsheet program. The crosstrack error calculation discussed in 2.3.1 was used again to convert the position data into individual error data. The reference endpoint coordinates define the line, and the crosstrack error at each data point was computed. This individual error value is the perpendicular distance between the control point of the machine and the intended path. The sign convention for these values is the same as that used for navigation: a negative error indicates the machine is on the left side of the line. Once the error data was generated some particular statistics were calculated for each test and for the data from all the tests. These summary statistics are collected in Tables 4.2 and 4.3 below.

<table>
<thead>
<tr>
<th>All Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Points</td>
</tr>
<tr>
<td>Standard Deviation (m)</td>
</tr>
<tr>
<td>Absolute Maximum (m)</td>
</tr>
</tbody>
</table>
Table 4.3: Summary Statistics for Straight-Line Test Error Data

<table>
<thead>
<tr>
<th></th>
<th>Test 25</th>
<th>Test 26</th>
<th>Test 28</th>
<th>Test 29</th>
<th>Test 30</th>
<th>Test 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Points</td>
<td>55</td>
<td>55</td>
<td>48</td>
<td>57</td>
<td>52</td>
<td>58</td>
</tr>
<tr>
<td>Standard Deviation (m)</td>
<td>0.1740</td>
<td>0.2215</td>
<td>0.2412</td>
<td>0.2500</td>
<td>0.2344</td>
<td>0.2492</td>
</tr>
<tr>
<td>Absolute Maximum (m)</td>
<td>0.3539</td>
<td>0.4477</td>
<td>0.5304</td>
<td>0.5824</td>
<td>0.5072</td>
<td>0.4485</td>
</tr>
<tr>
<td>Peak to Peak Error (m)</td>
<td>0.5819</td>
<td>0.8182</td>
<td>0.9606</td>
<td>0.9619</td>
<td>0.8789</td>
<td>0.9344</td>
</tr>
</tbody>
</table>

Measures of the variance of the error are needed to describe the performance of the system. The standard deviation is used as the measure of variance in this analysis. Standard deviation ($\sigma$) is a common measure of variance. Assuming that the data is normally distributed, 95% of data values lie within a range of twice the standard deviation plus or minus the mean. The total range of error values was also calculated for each individual test, this gives another way of looking at the performance. The third value reported is the absolute maximum for each test and for all the tests. This is the farthest the control point of the machine got from the path during each test.

These three performance measures agree quite well for the data collected. Using the plus or minus $2\sigma$ rule the predicted range for 95% of the data is 0.9264 meters, calculated from the standard deviation for all the data. Test 29 had the largest range, 0.9619 meters, which was just slightly larger than the predicted range. This agreement indicates that the error values are spread fairly evenly throughout the range. If most of the values were smaller and clustered around the mean then the predicted range would be notably smaller than the actual range. The same observation can be made using the absolute maximum values. From this data the control point of the machine is expected to be within 0.5 meters of the path at any given time. The absolute maximum values agree with this expectation, being within 0.6 meters of the path. If the absolute maximum values were outliers the predicted absolute maximum value would be much lower. It was observed that the results for Test 25 are notably better than the other
tests, but it is not known why this occurred.

The individual error data was also plotted as a sequence from each test. These graphs are given in Figures 4.3 through 4.8. These are plots of the crosstrack error against the point number, so that the data is in sequential order. The ordinal axis is reversed so that error sign convention makes more sense graphically. All the graphs are similar in overall character, with substantial oscillations over most of the range. These give another visual representation of the behavior described in the previous section.

4.3 Summary

The machine navigated from point to point along a straight line keeping its control point within 0.4632 meters of the straight line path 95% of the time. This is not as high precision as is desired for most operations, where 0.1-0.3 meters to either side of the reference is the outer limit. Furthermore, the oscillations in the path were higher amplitude and higher frequency than is acceptable for an agricultural vehicle.

The observation that the microcontroller representation of the machine’s position was not as precise as the position reported by the GPS is very important. The conversion from Latitude-Longitude Coordinates to State Plane Coordinates on high precision position data requires the ability to calculate and store numbers with twelve significant digits. Since the Z-world microcontrollers do not have this capability they will introduce errors into the process. These errors tend to ”pixelate” the data, meaning that distinct Latitude-Longitude Coordinates correspond to identical State Plane Coordinates. This sort of error can cause oscillations of the sort seen in the test results because the reported position appears to ”jump” from the center of one ”pixel” to the center of another.

Other factors may be contributing to the oscillation and to the imprecise navigation. It may be valuable to consider these things in the course of further development:

• The GPS antenna was positioned on the machine’s centerline at or very slightly ahead of the control point. Other research [17] suggests that this
causes the measured position to lag the current position. To correct this the antenna may be positioned farther forward or an algorithm may be used to predict the machine’s position.

- The left and right controls have slightly different response characteristics, and those responses will change slightly over time due to component wear, etc. Since the control is dependent on balancing the left and right wheel speeds a sensor system which measures those wheel speeds could be added to improve the precision of the control. Currently the overall control strategy assumes that different response characteristics can be lumped with angular error.

- Control loop timing can cause erratic or oscillatory responses if it is too slow to properly track changes. This can be corrected by simplifying the control loop or distributing the control functions between several microcontrollers, or by using a faster microcontroller.

Because of the known errors introduced by the microcontroller it is not possible to determine if these factors are negatively influencing the machine’s performance until the known problem is corrected.

The implementation of autonomous navigation for this system was done without many of the challenges that developers face when working with conventional vehicles. The differential steering zero turning radius vehicle simplifies the navigation problem because it has a simple, easily accessed steering and speed control system. It also allows the designer to treat the machine’s path as a series of lines connected at arbitrary angles, which reduces the amount of data required to store the path and works well with a simple control algorithm. These results do not reduce the potential that zero turning radius vehicles have for autonomous agriculture.
4.4 Figures
Figure 4.1: The Navigation Points Used for Testing

Figure 4.2: Plot of Position Data From Test 26
Figure 4.3: Crosstrack Error Sequence for Test 25
Figure 4.4: Crosstrack Error Sequence for Test 26
Figure 4.5: Crosstrack Error Sequence for Test 28
Figure 4.6: Crosstrack Error Sequence for Test 29
Figure 4.7: Crosstrack Error Sequence for Test 30
Figure 4.8: Crosstrack Error Sequence for Test 31
Autonomous agricultural machines are the next major step in agriculture and a major focus of present research and development. These machines may increase productivity at a rate comparable to the introduction of the conventional tractor. This body of work has described the development of an autonomous agricultural machine capable of using GPS position data to navigate from point to point along a straight-line path. The development was built on a foundation of work done using other types of position sensors on a zero turning radius vehicle and microcontroller computing, and influenced by research published in recent years on similar efforts at other universities throughout the world.

The system used microcontroller computing and a simple control strategy which treated paths as a straight line between two points, and simultaneously minimized the deviation between the machine’s direction of travel and the heading of the path (angular error) and the distance between the machine’s control point and the path (crosstrack error). This strategy reduces the control algorithm to a few simple calculations, which are easily designed and implemented. It was desirable to treat the machine’s position information as cartesian coordinates, which required converting Latitude-Longitude coordinates to the State Plane Coordinate system, a high precision rectangular coordinate system.

This system was implemented on a zero turning radius (ZTR) commercial lawn-mower using electronic actuators, an expandable microcontroller with analog voltage outputs and a GPS receiver and an electronic compass as sensor inputs. These components were purchased as commercial off-the-shelf units and combined into a functional autonomous vehicle.
Development and testing were done on a flat grassy lawn bordered by trees and buildings which provided an ideal simulation of a landscape management application. A straight line path 45 meters in length was set up for testing the machine. The machine started at one end of this path and navigated to the other end. The performance was measured by recording the position reported by the RTK-GPS on a handheld computer and calculating the crosstrack error between the recorded position and the straight line path.

The results show that a system has been developed which can use precision GPS data to navigate from point to point in a work area. However, the machine’s performance is not yet adequate for most applications. The range of the deviation from a straight-line path is 0.9264 meters, and the machine oscillates substantially within that range. Those oscillations are of shorter period and higher amplitude than is desirable for an agricultural vehicle, which should follow a smooth path. Consequently, the machine will require further development in hardware and software, and may require changes to the control strategy.

The first and most important area for development comes from the observation that the microcontroller does not have the computing power to fully represent the position data or perform needed calculations on that data. This is a problem which must be corrected before high precision behavior can be expected from the control system. This can be done by a complete replacement with a different microcontroller, or by addition of another microcontroller to distribute the computing load. Adding a second controller with the architecture to work with high precision GPS data but without the interfacing capability of the SmartStar would be a superior solution for this project because it would distribute the computing requirements and reduce the overall expense, since a controller which can communicate via RS-232 and perform 64-bit calculations may be had at a lower cost than a controller which can do those things as well as high precision analog output and user interfacing. A greater advantage is that distributing the computing power allows elements of the control program to run in parallel on separate processors, which minimizes any control loop delay. This will
be extremely important when developing integration between the GPS and machine vision sensors.

It is desirable for the machine to follow a smooth path rather than an oscillatory one in order to minimize damage to the work area, to minimize damage to the machine, and to instill confidence in people who observe the machine working. Therefore every effort should be made to optimize the path for smoothness rather than error response. This corresponds to the way a human operator steers, making a slight gentle correction rather than a sudden and swift correction. The PID controller may be optimized for this behavior by penalizing the derivative term more than the integral term, if a stable controller can be generated with this configuration. It is particularly important to note that excessive oscillation will cause the drivetrain to wear out more quickly, reducing the lifespan of the machine.

These developments must be made to achieve the desired level of performance for an autonomous agricultural machine. Other areas of development which should be addressed include:

- Developing a robust user interface;
- Integrating GPS and machine vision based navigation systems;
- Designing and implementing hazard detection safety systems;
- Developing applications and consumer confidence;

These things will be essential to a truly useful and marketable autonomous agricultural vehicle, particularly one which will be used in contexts which are highly visible to the general public.

The present system is a very hardy system within the limits of the original vehicle. The added electronics and their connections are able to withstand conditions likely to be encountered in agricultural applications, such as dust, heat, high humidity, direct sunlight, and vibration. The system as a whole is very good and has much to offer as an autonomous vehicle in both commercial landscaping and agritourism. The ability to successfully navigate from one point to another is a step in the process of achieving high precision navigation.
Literature Cited


Appendices
# Appendix A

## Data

### A.1 Data From Test 25

Table A.1: Raw data and calculated SPC coordinates for Test 25

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35.78304907</td>
<td>-78.67956087</td>
<td>638569.621</td>
<td>225576.8</td>
</tr>
<tr>
<td>2</td>
<td>35.78304907</td>
<td>-78.67956087</td>
<td>638569.621</td>
<td>225576.8</td>
</tr>
<tr>
<td>3</td>
<td>35.78304907</td>
<td>-78.67956087</td>
<td>638569.621</td>
<td>225576.8</td>
</tr>
<tr>
<td>4</td>
<td>35.78304907</td>
<td>-78.67956087</td>
<td>638569.621</td>
<td>225576.8</td>
</tr>
<tr>
<td>5</td>
<td>35.78304936</td>
<td>-78.67956159</td>
<td>638569.556</td>
<td>225576.832</td>
</tr>
<tr>
<td>6</td>
<td>35.78305184</td>
<td>-78.67957005</td>
<td>638568.791</td>
<td>225577.105</td>
</tr>
<tr>
<td>7</td>
<td>35.78305461</td>
<td>-78.67957904</td>
<td>638567.977</td>
<td>225577.409</td>
</tr>
<tr>
<td>8</td>
<td>35.78305739</td>
<td>-78.67958785</td>
<td>638567.179</td>
<td>225577.715</td>
</tr>
<tr>
<td>9</td>
<td>35.78306016</td>
<td>-78.67959756</td>
<td>638566.301</td>
<td>225578.02</td>
</tr>
<tr>
<td>10</td>
<td>35.78306381</td>
<td>-78.67960656</td>
<td>638565.486</td>
<td>225578.422</td>
</tr>
<tr>
<td>11</td>
<td>35.7830676</td>
<td>-78.67961501</td>
<td>638564.72</td>
<td>225578.84</td>
</tr>
<tr>
<td>12</td>
<td>35.78307183</td>
<td>-78.67962383</td>
<td>638563.922</td>
<td>225579.307</td>
</tr>
<tr>
<td>13</td>
<td>35.78307708</td>
<td>-78.67963246</td>
<td>638563.14</td>
<td>225579.887</td>
</tr>
<tr>
<td>14</td>
<td>35.78308219</td>
<td>-78.67964163</td>
<td>638562.309</td>
<td>225580.451</td>
</tr>
<tr>
<td>15</td>
<td>35.78308686</td>
<td>-78.67965098</td>
<td>638561.462</td>
<td>225580.966</td>
</tr>
<tr>
<td>16</td>
<td>35.78309095</td>
<td>-78.67966178</td>
<td>638560.484</td>
<td>225581.417</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>35.783094</td>
<td>−78.67967</td>
<td>638559.5</td>
<td>225581.77</td>
</tr>
<tr>
<td>18</td>
<td>35.783096</td>
<td>−78.67968</td>
<td>638558.56</td>
<td>225582.025</td>
</tr>
<tr>
<td>19</td>
<td>35.783099</td>
<td>−78.67969</td>
<td>638557.47</td>
<td>225582.395</td>
</tr>
<tr>
<td>20</td>
<td>35.783103</td>
<td>−78.67970</td>
<td>638556.496</td>
<td>225582.844</td>
</tr>
<tr>
<td>21</td>
<td>35.783109</td>
<td>−78.67971</td>
<td>638555.599</td>
<td>225583.425</td>
</tr>
<tr>
<td>22</td>
<td>35.783115</td>
<td>−78.67972</td>
<td>638554.703</td>
<td>225584.085</td>
</tr>
<tr>
<td>23</td>
<td>35.783120</td>
<td>−78.67935</td>
<td>638553.775</td>
<td>225584.682</td>
</tr>
<tr>
<td>24</td>
<td>35.783125</td>
<td>−78.67947</td>
<td>638552.749</td>
<td>225585.18</td>
</tr>
<tr>
<td>25</td>
<td>35.783128</td>
<td>−78.67958</td>
<td>638551.74</td>
<td>225585.565</td>
</tr>
<tr>
<td>26</td>
<td>35.783132</td>
<td>−78.67969</td>
<td>638550.713</td>
<td>225585.983</td>
</tr>
<tr>
<td>27</td>
<td>35.783136</td>
<td>−78.67980</td>
<td>638549.72</td>
<td>225586.465</td>
</tr>
<tr>
<td>28</td>
<td>35.783141</td>
<td>−78.67979</td>
<td>638548.694</td>
<td>225586.98</td>
</tr>
<tr>
<td>29</td>
<td>35.783146</td>
<td>−78.67902</td>
<td>638547.733</td>
<td>225587.527</td>
</tr>
<tr>
<td>30</td>
<td>35.783150</td>
<td>−78.67914</td>
<td>638546.691</td>
<td>225587.945</td>
</tr>
<tr>
<td>31</td>
<td>35.783154</td>
<td>−78.67925</td>
<td>638545.616</td>
<td>225588.411</td>
</tr>
<tr>
<td>32</td>
<td>35.783158</td>
<td>−78.67938</td>
<td>638544.542</td>
<td>225588.909</td>
</tr>
<tr>
<td>33</td>
<td>35.783164</td>
<td>−78.67948</td>
<td>638543.547</td>
<td>225589.537</td>
</tr>
<tr>
<td>34</td>
<td>35.783169</td>
<td>−78.67959</td>
<td>638542.537</td>
<td>225590.101</td>
</tr>
<tr>
<td>35</td>
<td>35.783174</td>
<td>−78.67987</td>
<td>638541.446</td>
<td>225590.599</td>
</tr>
<tr>
<td>36</td>
<td>35.783178</td>
<td>−78.67983</td>
<td>638540.437</td>
<td>225591.065</td>
</tr>
<tr>
<td>37</td>
<td>35.783182</td>
<td>−78.67989</td>
<td>638539.525</td>
<td>225591.451</td>
</tr>
<tr>
<td>38</td>
<td>35.783186</td>
<td>−78.67990</td>
<td>638538.581</td>
<td>225591.917</td>
</tr>
<tr>
<td>39</td>
<td>35.783190</td>
<td>−78.67991</td>
<td>638537.457</td>
<td>225592.415</td>
</tr>
<tr>
<td>40</td>
<td>35.783196</td>
<td>−78.67993</td>
<td>638536.82</td>
<td>225593.057</td>
</tr>
<tr>
<td>41</td>
<td>35.783199</td>
<td>−78.67994</td>
<td>638534.657</td>
<td>225593.394</td>
</tr>
<tr>
<td>42</td>
<td>35.783203</td>
<td>−78.67958</td>
<td>638533.631</td>
<td>225593.86</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.1 – continued from previous page

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>35.78320899</td>
<td>−78.67996934</td>
<td>638532.638</td>
<td>225594.423</td>
</tr>
<tr>
<td>44</td>
<td>35.78321424</td>
<td>−78.67997996</td>
<td>638531.676</td>
<td>225595.003</td>
</tr>
<tr>
<td>45</td>
<td>35.78321964</td>
<td>−78.67999183</td>
<td>638530.601</td>
<td>225595.598</td>
</tr>
<tr>
<td>46</td>
<td>35.78322431</td>
<td>−78.68000424</td>
<td>638529.477</td>
<td>225596.113</td>
</tr>
<tr>
<td>47</td>
<td>35.78322752</td>
<td>−78.68001629</td>
<td>638528.387</td>
<td>225596.465</td>
</tr>
<tr>
<td>48</td>
<td>35.78323102</td>
<td>−78.6800287</td>
<td>638527.264</td>
<td>225596.85</td>
</tr>
<tr>
<td>49</td>
<td>35.78323525</td>
<td>−78.68003931</td>
<td>638526.303</td>
<td>225597.316</td>
</tr>
<tr>
<td>50</td>
<td>35.78323978</td>
<td>−78.68004884</td>
<td>638525.44</td>
<td>225597.816</td>
</tr>
<tr>
<td>51</td>
<td>35.78323978</td>
<td>−78.68004866</td>
<td>638525.456</td>
<td>225597.816</td>
</tr>
<tr>
<td>52</td>
<td>35.78323978</td>
<td>−78.68004866</td>
<td>638525.456</td>
<td>225597.816</td>
</tr>
<tr>
<td>53</td>
<td>35.78323978</td>
<td>−78.68004866</td>
<td>638525.456</td>
<td>225597.816</td>
</tr>
<tr>
<td>54</td>
<td>35.78323992</td>
<td>−78.68004866</td>
<td>638525.456</td>
<td>225597.832</td>
</tr>
<tr>
<td>55</td>
<td>35.78323992</td>
<td>−78.68004884</td>
<td>638525.44</td>
<td>225597.832</td>
</tr>
</tbody>
</table>

92
### A.2 Data From Test 26

Table A.2: Raw data and calculated SPC coordinates for Test 26

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35.78304863</td>
<td>−78.67956015</td>
<td>638569.687</td>
<td>225576.751</td>
</tr>
<tr>
<td>2</td>
<td>35.78304863</td>
<td>−78.67956015</td>
<td>638569.687</td>
<td>225576.751</td>
</tr>
<tr>
<td>3</td>
<td>35.78304994</td>
<td>−78.67956393</td>
<td>638569.345</td>
<td>225576.896</td>
</tr>
<tr>
<td>4</td>
<td>35.78305213</td>
<td>−78.67957094</td>
<td>638568.71</td>
<td>225577.137</td>
</tr>
<tr>
<td>5</td>
<td>35.78305476</td>
<td>−78.6795785</td>
<td>638568.026</td>
<td>225577.426</td>
</tr>
<tr>
<td>6</td>
<td>35.78305724</td>
<td>−78.67958065</td>
<td>638567.342</td>
<td>225577.699</td>
</tr>
<tr>
<td>7</td>
<td>35.78306322</td>
<td>−78.67960296</td>
<td>638565.811</td>
<td>225578.358</td>
</tr>
<tr>
<td>8</td>
<td>35.78306731</td>
<td>−78.67961123</td>
<td>638565.062</td>
<td>225578.809</td>
</tr>
<tr>
<td>9</td>
<td>35.78307183</td>
<td>−78.67961933</td>
<td>638564.328</td>
<td>225579.308</td>
</tr>
<tr>
<td>10</td>
<td>35.78307679</td>
<td>−78.67962742</td>
<td>638563.595</td>
<td>225579.856</td>
</tr>
<tr>
<td>11</td>
<td>35.78308161</td>
<td>−78.67963642</td>
<td>638562.78</td>
<td>225580.388</td>
</tr>
<tr>
<td>12</td>
<td>35.78308671</td>
<td>−78.67964559</td>
<td>638561.949</td>
<td>225580.951</td>
</tr>
<tr>
<td>13</td>
<td>35.78309095</td>
<td>−78.67965512</td>
<td>638561.086</td>
<td>225581.419</td>
</tr>
<tr>
<td>14</td>
<td>35.78309445</td>
<td>−78.67966501</td>
<td>638560.191</td>
<td>225581.804</td>
</tr>
<tr>
<td>15</td>
<td>35.78309707</td>
<td>−78.67967491</td>
<td>638559.295</td>
<td>225582.092</td>
</tr>
<tr>
<td>16</td>
<td>35.78309912</td>
<td>−78.67968462</td>
<td>638558.416</td>
<td>225582.317</td>
</tr>
<tr>
<td>17</td>
<td>35.78310145</td>
<td>−78.67969451</td>
<td>638557.521</td>
<td>225582.572</td>
</tr>
<tr>
<td>18</td>
<td>35.78310393</td>
<td>−78.6797044</td>
<td>638556.626</td>
<td>225582.845</td>
</tr>
<tr>
<td>19</td>
<td>35.78310758</td>
<td>−78.6797143</td>
<td>638555.73</td>
<td>225583.247</td>
</tr>
<tr>
<td>20</td>
<td>35.78311196</td>
<td>−78.67972383</td>
<td>638554.867</td>
<td>225583.73</td>
</tr>
<tr>
<td>21</td>
<td>35.78311736</td>
<td>−78.67973246</td>
<td>638554.085</td>
<td>225584.326</td>
</tr>
<tr>
<td>22</td>
<td>35.78312319</td>
<td>−78.67974182</td>
<td>638553.237</td>
<td>225584.971</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>35.78312815</td>
<td>−78.67975189</td>
<td>638552.325</td>
<td>225585.518</td>
</tr>
<tr>
<td>24</td>
<td>35.78313239</td>
<td>−78.67976304</td>
<td>638551.315</td>
<td>225585.985</td>
</tr>
<tr>
<td>25</td>
<td>35.78313472</td>
<td>−78.67977455</td>
<td>638550.274</td>
<td>225586.24</td>
</tr>
<tr>
<td>26</td>
<td>35.78313735</td>
<td>−78.67978642</td>
<td>638549.2</td>
<td>225586.529</td>
</tr>
<tr>
<td>27</td>
<td>35.78314056</td>
<td>−78.67979685</td>
<td>638548.256</td>
<td>225586.882</td>
</tr>
<tr>
<td>28</td>
<td>35.78314435</td>
<td>−78.67980729</td>
<td>638547.311</td>
<td>225587.299</td>
</tr>
<tr>
<td>29</td>
<td>35.78314858</td>
<td>−78.67981718</td>
<td>638546.415</td>
<td>225587.765</td>
</tr>
<tr>
<td>30</td>
<td>35.78315383</td>
<td>−78.67982725</td>
<td>638545.503</td>
<td>225588.345</td>
</tr>
<tr>
<td>31</td>
<td>35.78315836</td>
<td>−78.6798366</td>
<td>638544.656</td>
<td>225588.845</td>
</tr>
<tr>
<td>32</td>
<td>35.78316303</td>
<td>−78.6798456</td>
<td>638543.841</td>
<td>225589.36</td>
</tr>
<tr>
<td>33</td>
<td>35.78316755</td>
<td>−78.67985603</td>
<td>638542.896</td>
<td>225589.859</td>
</tr>
<tr>
<td>34</td>
<td>35.78317193</td>
<td>−78.67986574</td>
<td>638542.017</td>
<td>225590.342</td>
</tr>
<tr>
<td>35</td>
<td>35.78317601</td>
<td>−78.67987564</td>
<td>638541.12</td>
<td>225590.792</td>
</tr>
<tr>
<td>36</td>
<td>35.78317995</td>
<td>−78.67988625</td>
<td>638540.16</td>
<td>225591.226</td>
</tr>
<tr>
<td>37</td>
<td>35.78318345</td>
<td>−78.67989668</td>
<td>638539.216</td>
<td>225591.611</td>
</tr>
<tr>
<td>38</td>
<td>35.78318739</td>
<td>−78.67990675</td>
<td>638538.304</td>
<td>225592.045</td>
</tr>
<tr>
<td>39</td>
<td>35.7831906</td>
<td>−78.67991754</td>
<td>638537.327</td>
<td>225592.398</td>
</tr>
<tr>
<td>40</td>
<td>35.78319396</td>
<td>−78.67992798</td>
<td>638536.382</td>
<td>225592.768</td>
</tr>
<tr>
<td>41</td>
<td>35.78319746</td>
<td>−78.67993859</td>
<td>638535.422</td>
<td>225593.153</td>
</tr>
<tr>
<td>42</td>
<td>35.78320169</td>
<td>−78.67994884</td>
<td>638534.494</td>
<td>225593.619</td>
</tr>
<tr>
<td>43</td>
<td>35.78320593</td>
<td>−78.67995801</td>
<td>638533.663</td>
<td>225594.087</td>
</tr>
<tr>
<td>44</td>
<td>35.78321045</td>
<td>−78.67996755</td>
<td>638532.799</td>
<td>225594.586</td>
</tr>
<tr>
<td>45</td>
<td>35.78321526</td>
<td>−78.67997888</td>
<td>638531.773</td>
<td>225595.116</td>
</tr>
<tr>
<td>46</td>
<td>35.78322081</td>
<td>−78.67999704</td>
<td>638530.129</td>
<td>225595.727</td>
</tr>
<tr>
<td>47</td>
<td>35.78322373</td>
<td>−78.68000712</td>
<td>638529.217</td>
<td>225596.048</td>
</tr>
<tr>
<td>48</td>
<td>35.78322635</td>
<td>−78.68001719</td>
<td>638528.306</td>
<td>225596.335</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>35.78323</td>
<td>-78.6800269</td>
<td>638527.427</td>
<td>225596.737</td>
</tr>
<tr>
<td>50</td>
<td>35.7832335</td>
<td>-78.68003643</td>
<td>638526.564</td>
<td>225597.123</td>
</tr>
<tr>
<td>51</td>
<td>35.78323759</td>
<td>-78.68004597</td>
<td>638525.7</td>
<td>225597.574</td>
</tr>
<tr>
<td>52</td>
<td>35.78323846</td>
<td>-78.68004884</td>
<td>638525.44</td>
<td>225597.67</td>
</tr>
<tr>
<td>53</td>
<td>35.78323846</td>
<td>-78.68004866</td>
<td>638525.457</td>
<td>225597.67</td>
</tr>
<tr>
<td>54</td>
<td>35.78323846</td>
<td>-78.68004866</td>
<td>638525.457</td>
<td>225597.67</td>
</tr>
<tr>
<td>55</td>
<td>35.78323846</td>
<td>-78.68004866</td>
<td>638525.457</td>
<td>225597.67</td>
</tr>
</tbody>
</table>
### A.3 Data From Test 28

Table A.3: Raw data and calculated SPC coordinates for Test 28

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35.78304907</td>
<td>−78.67956069</td>
<td>638569.638</td>
<td>225576.8</td>
</tr>
<tr>
<td>2</td>
<td>35.78304907</td>
<td>−78.67956087</td>
<td>638569.621</td>
<td>225576.8</td>
</tr>
<tr>
<td>3</td>
<td>35.78305097</td>
<td>−78.67956789</td>
<td>638568.986</td>
<td>225577.009</td>
</tr>
<tr>
<td>4</td>
<td>35.78305345</td>
<td>−78.67957598</td>
<td>638568.254</td>
<td>225577.282</td>
</tr>
<tr>
<td>5</td>
<td>35.78305578</td>
<td>−78.67958497</td>
<td>638567.44</td>
<td>225577.537</td>
</tr>
<tr>
<td>6</td>
<td>35.78305826</td>
<td>−78.67959487</td>
<td>638566.545</td>
<td>225577.81</td>
</tr>
<tr>
<td>7</td>
<td>35.7830657</td>
<td>−78.67961429</td>
<td>638564.786</td>
<td>225578.63</td>
</tr>
<tr>
<td>8</td>
<td>35.78307081</td>
<td>−78.67962275</td>
<td>638564.02</td>
<td>225579.194</td>
</tr>
<tr>
<td>9</td>
<td>35.78307636</td>
<td>−78.6796312</td>
<td>638563.254</td>
<td>225579.807</td>
</tr>
<tr>
<td>10</td>
<td>35.78308234</td>
<td>−78.67964037</td>
<td>638562.423</td>
<td>225580.468</td>
</tr>
<tr>
<td>11</td>
<td>35.78308774</td>
<td>−78.67965027</td>
<td>638561.526</td>
<td>225581.064</td>
</tr>
<tr>
<td>12</td>
<td>35.78309255</td>
<td>−78.67966178</td>
<td>638560.483</td>
<td>225581.595</td>
</tr>
<tr>
<td>13</td>
<td>35.78309562</td>
<td>−78.67967347</td>
<td>638559.426</td>
<td>225581.932</td>
</tr>
<tr>
<td>14</td>
<td>35.78309853</td>
<td>−78.67968408</td>
<td>638558.465</td>
<td>225582.251</td>
</tr>
<tr>
<td>15</td>
<td>35.78310145</td>
<td>−78.67969577</td>
<td>638557.408</td>
<td>225582.572</td>
</tr>
<tr>
<td>16</td>
<td>35.78310495</td>
<td>−78.67970638</td>
<td>638556.447</td>
<td>225582.957</td>
</tr>
<tr>
<td>17</td>
<td>35.78310977</td>
<td>−78.67971646</td>
<td>638555.534</td>
<td>225583.489</td>
</tr>
<tr>
<td>18</td>
<td>35.78311561</td>
<td>−78.67972653</td>
<td>638554.622</td>
<td>225584.134</td>
</tr>
<tr>
<td>19</td>
<td>35.78312086</td>
<td>−78.67973678</td>
<td>638553.693</td>
<td>225584.714</td>
</tr>
<tr>
<td>20</td>
<td>35.78312509</td>
<td>−78.67974901</td>
<td>638552.586</td>
<td>225585.179</td>
</tr>
<tr>
<td>21</td>
<td>35.7831283</td>
<td>−78.6797607</td>
<td>638551.528</td>
<td>225585.532</td>
</tr>
<tr>
<td>22</td>
<td>35.78313239</td>
<td>−78.67977257</td>
<td>638550.454</td>
<td>225585.982</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>35.78313749</td>
<td>-78.67978354</td>
<td>638549.46</td>
<td>225586.545</td>
</tr>
<tr>
<td>24</td>
<td>35.78314377</td>
<td>-78.67979362</td>
<td>638548.547</td>
<td>225587.239</td>
</tr>
<tr>
<td>25</td>
<td>35.78315456</td>
<td>-78.67981646</td>
<td>638546.478</td>
<td>225588.429</td>
</tr>
<tr>
<td>26</td>
<td>35.78315836</td>
<td>-78.67982833</td>
<td>638545.403</td>
<td>225588.847</td>
</tr>
<tr>
<td>27</td>
<td>35.78316128</td>
<td>-78.67983948</td>
<td>638544.394</td>
<td>225589.168</td>
</tr>
<tr>
<td>28</td>
<td>35.78316434</td>
<td>-78.67985063</td>
<td>638543.385</td>
<td>225589.504</td>
</tr>
<tr>
<td>29</td>
<td>35.78316799</td>
<td>-78.67986197</td>
<td>638542.359</td>
<td>225589.906</td>
</tr>
<tr>
<td>30</td>
<td>35.78317222</td>
<td>-78.6798733</td>
<td>638541.333</td>
<td>225590.372</td>
</tr>
<tr>
<td>31</td>
<td>35.78317733</td>
<td>-78.67988373</td>
<td>638540.388</td>
<td>225590.936</td>
</tr>
<tr>
<td>32</td>
<td>35.78318273</td>
<td>-78.67989452</td>
<td>638539.411</td>
<td>225591.532</td>
</tr>
<tr>
<td>33</td>
<td>35.78318739</td>
<td>-78.67990531</td>
<td>638538.434</td>
<td>225592.046</td>
</tr>
<tr>
<td>34</td>
<td>35.78319119</td>
<td>-78.67991682</td>
<td>638537.392</td>
<td>225592.464</td>
</tr>
<tr>
<td>35</td>
<td>35.78319444</td>
<td>-78.67992834</td>
<td>638536.35</td>
<td>225592.817</td>
</tr>
<tr>
<td>36</td>
<td>35.78319746</td>
<td>-78.67994093</td>
<td>638535.21</td>
<td>225593.152</td>
</tr>
<tr>
<td>37</td>
<td>35.78320155</td>
<td>-78.67995226</td>
<td>638534.185</td>
<td>225593.603</td>
</tr>
<tr>
<td>38</td>
<td>35.78320636</td>
<td>-78.67996251</td>
<td>638533.256</td>
<td>225594.134</td>
</tr>
<tr>
<td>39</td>
<td>35.78321176</td>
<td>-78.67997276</td>
<td>638532.328</td>
<td>225594.73</td>
</tr>
<tr>
<td>40</td>
<td>35.78321701</td>
<td>-78.67998355</td>
<td>638531.35</td>
<td>225595.309</td>
</tr>
<tr>
<td>41</td>
<td>35.78322227</td>
<td>-78.67999471</td>
<td>638530.34</td>
<td>225595.889</td>
</tr>
<tr>
<td>42</td>
<td>35.78322664</td>
<td>-78.68000658</td>
<td>638529.265</td>
<td>225596.371</td>
</tr>
<tr>
<td>43</td>
<td>35.78323015</td>
<td>-78.68001899</td>
<td>638528.142</td>
<td>225596.756</td>
</tr>
<tr>
<td>44</td>
<td>35.78323263</td>
<td>-78.68003068</td>
<td>638527.084</td>
<td>225597.028</td>
</tr>
<tr>
<td>45</td>
<td>35.78323788</td>
<td>-78.68005154</td>
<td>638525.196</td>
<td>225597.605</td>
</tr>
<tr>
<td>46</td>
<td>35.78323788</td>
<td>-78.68005118</td>
<td>638525.229</td>
<td>225597.605</td>
</tr>
<tr>
<td>47</td>
<td>35.78323788</td>
<td>-78.68005118</td>
<td>638525.229</td>
<td>225597.605</td>
</tr>
<tr>
<td>48</td>
<td>35.78323773</td>
<td>-78.68005118</td>
<td>638525.229</td>
<td>225597.588</td>
</tr>
</tbody>
</table>
### Data From Test 29

Table A.4: Raw data and calculated SPC coordinates for Test 29

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35.78305</td>
<td>−78.6796</td>
<td>638569.9</td>
<td>225576.6</td>
</tr>
<tr>
<td>2</td>
<td>35.78305</td>
<td>−78.6796</td>
<td>638569.9</td>
<td>225576.6</td>
</tr>
<tr>
<td>3</td>
<td>35.78305</td>
<td>−78.6796</td>
<td>638569.3</td>
<td>225576.9</td>
</tr>
<tr>
<td>4</td>
<td>35.78305</td>
<td>−78.6796</td>
<td>638568.5</td>
<td>225577.2</td>
</tr>
<tr>
<td>5</td>
<td>35.78306</td>
<td>−78.6796</td>
<td>638567.8</td>
<td>225577.6</td>
</tr>
<tr>
<td>6</td>
<td>35.78306</td>
<td>−78.6796</td>
<td>638566.3</td>
<td>225578.3</td>
</tr>
<tr>
<td>7</td>
<td>35.78307</td>
<td>−78.6796</td>
<td>638565.5</td>
<td>225578.7</td>
</tr>
<tr>
<td>8</td>
<td>35.78307</td>
<td>−78.6796</td>
<td>638564.7</td>
<td>225579.1</td>
</tr>
<tr>
<td>9</td>
<td>35.78307</td>
<td>−78.6796</td>
<td>638563.9</td>
<td>225579.6</td>
</tr>
<tr>
<td>10</td>
<td>35.78308</td>
<td>−78.6796</td>
<td>638563.1</td>
<td>225580.1</td>
</tr>
<tr>
<td>11</td>
<td>35.78308</td>
<td>−78.6796</td>
<td>638562.3</td>
<td>225580.5</td>
</tr>
<tr>
<td>12</td>
<td>35.78309</td>
<td>−78.6797</td>
<td>638561.5</td>
<td>225581</td>
</tr>
<tr>
<td>13</td>
<td>35.78309</td>
<td>−78.6797</td>
<td>638560.6</td>
<td>225581.4</td>
</tr>
<tr>
<td>14</td>
<td>35.78309</td>
<td>−78.6797</td>
<td>638559.7</td>
<td>225581.8</td>
</tr>
<tr>
<td>15</td>
<td>35.7831</td>
<td>−78.6797</td>
<td>638558.8</td>
<td>225582.1</td>
</tr>
<tr>
<td>16</td>
<td>35.7831</td>
<td>−78.6797</td>
<td>638557.8</td>
<td>225582.5</td>
</tr>
<tr>
<td>17</td>
<td>35.7831</td>
<td>−78.6797</td>
<td>638556.9</td>
<td>225582.8</td>
</tr>
<tr>
<td>18</td>
<td>35.78311</td>
<td>−78.6797</td>
<td>638556</td>
<td>225583.2</td>
</tr>
<tr>
<td>19</td>
<td>35.78311</td>
<td>−78.6797</td>
<td>638555.2</td>
<td>225583.7</td>
</tr>
<tr>
<td>20</td>
<td>35.78312</td>
<td>−78.6797</td>
<td>638554.4</td>
<td>225584.3</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>35.78312</td>
<td>−78.6797</td>
<td>638553.6</td>
<td>225584.8</td>
</tr>
<tr>
<td>22</td>
<td>35.78313</td>
<td>−78.6797</td>
<td>638552.7</td>
<td>225585.4</td>
</tr>
<tr>
<td>23</td>
<td>35.78313</td>
<td>−78.6798</td>
<td>638551.7</td>
<td>225585.9</td>
</tr>
<tr>
<td>24</td>
<td>35.78314</td>
<td>−78.6798</td>
<td>638549.8</td>
<td>225586.6</td>
</tr>
<tr>
<td>25</td>
<td>35.78314</td>
<td>−78.6798</td>
<td>638548.7</td>
<td>225587</td>
</tr>
<tr>
<td>26</td>
<td>35.78315</td>
<td>−78.6798</td>
<td>638547.7</td>
<td>225587.4</td>
</tr>
<tr>
<td>27</td>
<td>35.78315</td>
<td>−78.6798</td>
<td>638546.8</td>
<td>225588</td>
</tr>
<tr>
<td>28</td>
<td>35.78316</td>
<td>−78.6798</td>
<td>638545.9</td>
<td>225588.5</td>
</tr>
<tr>
<td>29</td>
<td>35.78316</td>
<td>−78.6798</td>
<td>638544.9</td>
<td>225589</td>
</tr>
<tr>
<td>30</td>
<td>35.78316</td>
<td>−78.6798</td>
<td>638544</td>
<td>225589.5</td>
</tr>
<tr>
<td>31</td>
<td>35.78317</td>
<td>−78.6799</td>
<td>638542.9</td>
<td>225589.8</td>
</tr>
<tr>
<td>32</td>
<td>35.78317</td>
<td>−78.6799</td>
<td>638541.8</td>
<td>225590.2</td>
</tr>
<tr>
<td>33</td>
<td>35.78317</td>
<td>−78.6799</td>
<td>638540.8</td>
<td>225590.6</td>
</tr>
<tr>
<td>34</td>
<td>35.78318</td>
<td>−78.6799</td>
<td>638539.8</td>
<td>225591.2</td>
</tr>
<tr>
<td>35</td>
<td>35.78318</td>
<td>−78.6799</td>
<td>638538.8</td>
<td>225591.7</td>
</tr>
<tr>
<td>36</td>
<td>35.78319</td>
<td>−78.6799</td>
<td>638537.8</td>
<td>225592.2</td>
</tr>
<tr>
<td>37</td>
<td>35.78319</td>
<td>−78.6799</td>
<td>638536.8</td>
<td>225592.6</td>
</tr>
<tr>
<td>38</td>
<td>35.7832</td>
<td>−78.6799</td>
<td>638535.8</td>
<td>225593</td>
</tr>
<tr>
<td>39</td>
<td>35.7832</td>
<td>−78.6799</td>
<td>638534.8</td>
<td>225593.4</td>
</tr>
<tr>
<td>40</td>
<td>35.7832</td>
<td>−78.68</td>
<td>638533.9</td>
<td>225593.9</td>
</tr>
<tr>
<td>41</td>
<td>35.78321</td>
<td>−78.68</td>
<td>638532.9</td>
<td>225594.4</td>
</tr>
<tr>
<td>42</td>
<td>35.78321</td>
<td>−78.68</td>
<td>638532</td>
<td>225594.9</td>
</tr>
<tr>
<td>43</td>
<td>35.78322</td>
<td>−78.68</td>
<td>638530.7</td>
<td>225595.6</td>
</tr>
<tr>
<td>44</td>
<td>35.78322</td>
<td>−78.68</td>
<td>638528.9</td>
<td>225596.1</td>
</tr>
<tr>
<td>45</td>
<td>35.78323</td>
<td>−78.68</td>
<td>638527.8</td>
<td>225596.4</td>
</tr>
<tr>
<td>46</td>
<td>35.78323</td>
<td>−78.68</td>
<td>638526.7</td>
<td>225596.8</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>35.78324</td>
<td>-78.68</td>
<td>638525.7</td>
<td>225597.3</td>
</tr>
<tr>
<td>48</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>49</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>50</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>51</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>52</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>53</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>54</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>55</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>56</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
<tr>
<td>57</td>
<td>35.78324</td>
<td>-78.6801</td>
<td>638525</td>
<td>225597.8</td>
</tr>
</tbody>
</table>
## A.5 Data From Test 30

Table A.5: Raw data and calculated SPC coordinates for Test 30

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35.78304849</td>
<td>−78.67956069</td>
<td>638569.638</td>
<td>225576.736</td>
</tr>
<tr>
<td>2</td>
<td>35.78304849</td>
<td>−78.67956069</td>
<td>638569.638</td>
<td>225576.736</td>
</tr>
<tr>
<td>3</td>
<td>35.78304849</td>
<td>−78.67956069</td>
<td>638569.638</td>
<td>225576.736</td>
</tr>
<tr>
<td>4</td>
<td>35.78304965</td>
<td>−78.67956285</td>
<td>638569.442</td>
<td>225576.864</td>
</tr>
<tr>
<td>5</td>
<td>35.78305243</td>
<td>−78.67957112</td>
<td>638568.694</td>
<td>225577.17</td>
</tr>
<tr>
<td>6</td>
<td>35.78305564</td>
<td>−78.6795803</td>
<td>638567.863</td>
<td>225577.523</td>
</tr>
<tr>
<td>7</td>
<td>35.78305885</td>
<td>−78.67958893</td>
<td>638567.081</td>
<td>225577.877</td>
</tr>
<tr>
<td>8</td>
<td>35.78306293</td>
<td>−78.67959792</td>
<td>638566.267</td>
<td>225578.327</td>
</tr>
<tr>
<td>9</td>
<td>35.78306775</td>
<td>−78.67960872</td>
<td>638565.289</td>
<td>225578.859</td>
</tr>
<tr>
<td>10</td>
<td>35.78307592</td>
<td>−78.67962472</td>
<td>638563.84</td>
<td>225579.76</td>
</tr>
<tr>
<td>11</td>
<td>35.78308029</td>
<td>−78.67963408</td>
<td>638562.992</td>
<td>225580.242</td>
</tr>
<tr>
<td>12</td>
<td>35.78308453</td>
<td>−78.67964397</td>
<td>638562.096</td>
<td>225580.71</td>
</tr>
<tr>
<td>13</td>
<td>35.78308774</td>
<td>−78.67965386</td>
<td>638561.201</td>
<td>225581.063</td>
</tr>
<tr>
<td>14</td>
<td>35.78309109</td>
<td>−78.67966376</td>
<td>638560.305</td>
<td>225581.432</td>
</tr>
<tr>
<td>15</td>
<td>35.78309357</td>
<td>−78.67967383</td>
<td>638559.394</td>
<td>225581.704</td>
</tr>
<tr>
<td>16</td>
<td>35.78309635</td>
<td>−78.67968336</td>
<td>638558.531</td>
<td>225582.01</td>
</tr>
<tr>
<td>17</td>
<td>35.78309999</td>
<td>−78.67969433</td>
<td>638557.538</td>
<td>225582.41</td>
</tr>
<tr>
<td>18</td>
<td>35.78310422</td>
<td>−78.6797044</td>
<td>638556.626</td>
<td>225582.877</td>
</tr>
<tr>
<td>19</td>
<td>35.78310919</td>
<td>−78.6797134</td>
<td>638555.811</td>
<td>225583.426</td>
</tr>
<tr>
<td>20</td>
<td>35.78311458</td>
<td>−78.67972239</td>
<td>638554.996</td>
<td>225584.021</td>
</tr>
<tr>
<td>21</td>
<td>35.78311969</td>
<td>−78.67973174</td>
<td>638554.149</td>
<td>225584.585</td>
</tr>
<tr>
<td>22</td>
<td>35.78312436</td>
<td>−78.67974236</td>
<td>638553.187</td>
<td>225585.1</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>35.78312742</td>
<td>-78.67975387</td>
<td>638552.146</td>
<td>225585.436</td>
</tr>
<tr>
<td>24</td>
<td>35.78313049</td>
<td>-78.67976466</td>
<td>638551.169</td>
<td>225585.774</td>
</tr>
<tr>
<td>25</td>
<td>35.78313384</td>
<td>-78.67977563</td>
<td>638550.176</td>
<td>225586.142</td>
</tr>
<tr>
<td>26</td>
<td>35.78313778</td>
<td>-78.6797866</td>
<td>638549.183</td>
<td>225586.576</td>
</tr>
<tr>
<td>27</td>
<td>35.78314231</td>
<td>-78.67979757</td>
<td>638548.19</td>
<td>225587.076</td>
</tr>
<tr>
<td>28</td>
<td>35.783148</td>
<td>-78.67980783</td>
<td>638547.26</td>
<td>225587.704</td>
</tr>
<tr>
<td>29</td>
<td>35.78315383</td>
<td>-78.67981862</td>
<td>638546.283</td>
<td>225588.348</td>
</tr>
<tr>
<td>30</td>
<td>35.78315909</td>
<td>-78.67982941</td>
<td>638545.306</td>
<td>225588.928</td>
</tr>
<tr>
<td>31</td>
<td>35.78316346</td>
<td>-78.67984092</td>
<td>638544.264</td>
<td>225589.409</td>
</tr>
<tr>
<td>32</td>
<td>35.7831674</td>
<td>-78.67985279</td>
<td>638543.189</td>
<td>225589.843</td>
</tr>
<tr>
<td>33</td>
<td>35.7831712</td>
<td>-78.67986484</td>
<td>638542.098</td>
<td>225590.261</td>
</tr>
<tr>
<td>34</td>
<td>35.78317528</td>
<td>-78.67987635</td>
<td>638541.056</td>
<td>225590.71</td>
</tr>
<tr>
<td>35</td>
<td>35.7831801</td>
<td>-78.67988733</td>
<td>638540.062</td>
<td>225591.242</td>
</tr>
<tr>
<td>36</td>
<td>35.78318491</td>
<td>-78.67989848</td>
<td>638539.052</td>
<td>225591.772</td>
</tr>
<tr>
<td>37</td>
<td>35.78318958</td>
<td>-78.67991035</td>
<td>638537.978</td>
<td>225592.287</td>
</tr>
<tr>
<td>38</td>
<td>35.78319338</td>
<td>-78.67992204</td>
<td>638536.919</td>
<td>225592.705</td>
</tr>
<tr>
<td>39</td>
<td>35.78319688</td>
<td>-78.67993445</td>
<td>638535.796</td>
<td>225593.09</td>
</tr>
<tr>
<td>40</td>
<td>35.78320038</td>
<td>-78.67994632</td>
<td>638534.722</td>
<td>225593.475</td>
</tr>
<tr>
<td>41</td>
<td>35.78320476</td>
<td>-78.67995711</td>
<td>638533.745</td>
<td>225593.958</td>
</tr>
<tr>
<td>42</td>
<td>35.78320972</td>
<td>-78.67996737</td>
<td>638532.816</td>
<td>225594.505</td>
</tr>
<tr>
<td>43</td>
<td>35.78321453</td>
<td>-78.6799778</td>
<td>638531.871</td>
<td>225595.035</td>
</tr>
<tr>
<td>44</td>
<td>35.78321891</td>
<td>-78.67998967</td>
<td>638530.796</td>
<td>225595.518</td>
</tr>
<tr>
<td>45</td>
<td>35.78322489</td>
<td>-78.68001233</td>
<td>638528.746</td>
<td>225596.175</td>
</tr>
<tr>
<td>46</td>
<td>35.78322796</td>
<td>-78.68002402</td>
<td>638527.688</td>
<td>225596.512</td>
</tr>
<tr>
<td>47</td>
<td>35.78323146</td>
<td>-78.68003464</td>
<td>638526.727</td>
<td>225596.897</td>
</tr>
<tr>
<td>48</td>
<td>35.78323569</td>
<td>-78.68004543</td>
<td>638525.75</td>
<td>225597.363</td>
</tr>
</tbody>
</table>
Table A.5 – continued from previous page

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>35.78323846</td>
<td>−78.68005118</td>
<td>638525.229</td>
<td>225597.669</td>
</tr>
<tr>
<td>50</td>
<td>35.78323846</td>
<td>−78.680051</td>
<td>638525.245</td>
<td>225597.669</td>
</tr>
<tr>
<td>51</td>
<td>35.78323846</td>
<td>−78.680051</td>
<td>638525.245</td>
<td>225597.669</td>
</tr>
<tr>
<td>52</td>
<td>35.78323832</td>
<td>−78.680051</td>
<td>638525.245</td>
<td>225597.654</td>
</tr>
</tbody>
</table>
A.6 Data From Test 31

Table A.6: Raw data and calculated SPC coordinates for Test 31

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35.78304819</td>
<td>−78.67955835</td>
<td>638569.85</td>
<td>225576.703</td>
</tr>
<tr>
<td>2</td>
<td>35.78304834</td>
<td>−78.67955835</td>
<td>638569.85</td>
<td>225576.72</td>
</tr>
<tr>
<td>3</td>
<td>35.78305009</td>
<td>−78.67956465</td>
<td>638569.279</td>
<td>225576.912</td>
</tr>
<tr>
<td>4</td>
<td>35.78305257</td>
<td>−78.67957256</td>
<td>638568.563</td>
<td>225577.185</td>
</tr>
<tr>
<td>5</td>
<td>35.78305491</td>
<td>−78.6795803</td>
<td>638567.863</td>
<td>225577.442</td>
</tr>
<tr>
<td>6</td>
<td>35.78305709</td>
<td>−78.67958821</td>
<td>638567.147</td>
<td>225577.682</td>
</tr>
<tr>
<td>7</td>
<td>35.78305928</td>
<td>−78.67959667</td>
<td>638566.381</td>
<td>225577.922</td>
</tr>
<tr>
<td>8</td>
<td>35.78306235</td>
<td>−78.67960548</td>
<td>638565.584</td>
<td>225578.26</td>
</tr>
<tr>
<td>9</td>
<td>35.783066</td>
<td>−78.67961465</td>
<td>638564.754</td>
<td>225578.663</td>
</tr>
<tr>
<td>10</td>
<td>35.78307037</td>
<td>−78.67962347</td>
<td>638563.955</td>
<td>225579.145</td>
</tr>
<tr>
<td>11</td>
<td>35.78307519</td>
<td>−78.67963174</td>
<td>638563.205</td>
<td>225579.677</td>
</tr>
<tr>
<td>12</td>
<td>35.78307986</td>
<td>−78.67963911</td>
<td>638562.537</td>
<td>225580.193</td>
</tr>
<tr>
<td>13</td>
<td>35.78308467</td>
<td>−78.67964739</td>
<td>638561.787</td>
<td>225580.724</td>
</tr>
<tr>
<td>14</td>
<td>35.78308905</td>
<td>−78.67965602</td>
<td>638561.005</td>
<td>225581.208</td>
</tr>
<tr>
<td>15</td>
<td>35.78309226</td>
<td>−78.67966591</td>
<td>638560.11</td>
<td>225581.561</td>
</tr>
<tr>
<td>16</td>
<td>35.78309445</td>
<td>−78.67967617</td>
<td>638559.182</td>
<td>225581.801</td>
</tr>
<tr>
<td>17</td>
<td>35.78309664</td>
<td>−78.67968696</td>
<td>638558.206</td>
<td>225582.041</td>
</tr>
<tr>
<td>18</td>
<td>35.78309956</td>
<td>−78.67969865</td>
<td>638557.148</td>
<td>225582.362</td>
</tr>
<tr>
<td>19</td>
<td>35.78310422</td>
<td>−78.67971124</td>
<td>638556.008</td>
<td>225582.875</td>
</tr>
<tr>
<td>20</td>
<td>35.78311356</td>
<td>−78.67972761</td>
<td>638554.525</td>
<td>225583.906</td>
</tr>
<tr>
<td>21</td>
<td>35.78311984</td>
<td>−78.67973588</td>
<td>638553.775</td>
<td>225584.601</td>
</tr>
<tr>
<td>22</td>
<td>35.78312626</td>
<td>−78.67974577</td>
<td>638552.879</td>
<td>225585.31</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.6 – continued from previous page

<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>35.78313151</td>
<td>−78.67975674</td>
<td>638551.885</td>
<td>225585.889</td>
</tr>
<tr>
<td>24</td>
<td>35.7831356</td>
<td>−78.67976844</td>
<td>638550.826</td>
<td>225586.34</td>
</tr>
<tr>
<td>25</td>
<td>35.78313822</td>
<td>−78.67977959</td>
<td>638549.817</td>
<td>225586.627</td>
</tr>
<tr>
<td>26</td>
<td>35.7831407</td>
<td>−78.67979074</td>
<td>638548.808</td>
<td>225586.899</td>
</tr>
<tr>
<td>27</td>
<td>35.78314391</td>
<td>−78.67980189</td>
<td>638547.799</td>
<td>225587.252</td>
</tr>
<tr>
<td>28</td>
<td>35.78314756</td>
<td>−78.67981268</td>
<td>638546.822</td>
<td>225587.654</td>
</tr>
<tr>
<td>29</td>
<td>35.78315179</td>
<td>−78.67982329</td>
<td>638545.861</td>
<td>225588.12</td>
</tr>
<tr>
<td>30</td>
<td>35.78315675</td>
<td>−78.67983319</td>
<td>638544.965</td>
<td>225588.667</td>
</tr>
<tr>
<td>31</td>
<td>35.78316157</td>
<td>−78.67984236</td>
<td>638544.134</td>
<td>225589.199</td>
</tr>
<tr>
<td>32</td>
<td>35.78316653</td>
<td>−78.67985315</td>
<td>638543.157</td>
<td>225589.746</td>
</tr>
<tr>
<td>33</td>
<td>35.78317061</td>
<td>−78.67986358</td>
<td>638542.212</td>
<td>225590.196</td>
</tr>
<tr>
<td>34</td>
<td>35.78317485</td>
<td>−78.67987456</td>
<td>638541.218</td>
<td>225590.663</td>
</tr>
<tr>
<td>35</td>
<td>35.78317879</td>
<td>−78.67988499</td>
<td>638540.274</td>
<td>225591.097</td>
</tr>
<tr>
<td>36</td>
<td>35.78318302</td>
<td>−78.67989506</td>
<td>638539.362</td>
<td>225591.564</td>
</tr>
<tr>
<td>37</td>
<td>35.78318769</td>
<td>−78.67990603</td>
<td>638538.369</td>
<td>225592.079</td>
</tr>
<tr>
<td>38</td>
<td>35.78319177</td>
<td>−78.6799117</td>
<td>638537.376</td>
<td>225592.528</td>
</tr>
<tr>
<td>39</td>
<td>35.78319557</td>
<td>−78.67992834</td>
<td>638536.349</td>
<td>225592.946</td>
</tr>
<tr>
<td>40</td>
<td>35.78319863</td>
<td>−78.67993967</td>
<td>638535.324</td>
<td>225593.283</td>
</tr>
<tr>
<td>41</td>
<td>35.78320111</td>
<td>−78.67994974</td>
<td>638534.413</td>
<td>225593.555</td>
</tr>
<tr>
<td>42</td>
<td>35.78320417</td>
<td>−78.67996017</td>
<td>638533.469</td>
<td>225593.891</td>
</tr>
<tr>
<td>43</td>
<td>35.78320797</td>
<td>−78.67997042</td>
<td>638532.541</td>
<td>225594.31</td>
</tr>
<tr>
<td>44</td>
<td>35.78321235</td>
<td>−78.67998032</td>
<td>638531.644</td>
<td>225594.793</td>
</tr>
<tr>
<td>45</td>
<td>35.78321731</td>
<td>−78.67999003</td>
<td>638530.764</td>
<td>225595.34</td>
</tr>
<tr>
<td>46</td>
<td>35.78322241</td>
<td>−78.6800001</td>
<td>638529.852</td>
<td>225595.903</td>
</tr>
<tr>
<td>47</td>
<td>35.78322821</td>
<td>−78.68001053</td>
<td>638528.907</td>
<td>225596.531</td>
</tr>
<tr>
<td>48</td>
<td>35.78323234</td>
<td>−78.68002204</td>
<td>638527.865</td>
<td>225596.999</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Number</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>35.78323628</td>
<td>-78.68003374</td>
<td>638526.806</td>
<td>225597.432</td>
</tr>
<tr>
<td>50</td>
<td>35.78323861</td>
<td>-78.68004561</td>
<td>638525.732</td>
<td>225597.687</td>
</tr>
<tr>
<td>51</td>
<td>35.78323963</td>
<td>-78.68005244</td>
<td>638525.114</td>
<td>225597.798</td>
</tr>
<tr>
<td>52</td>
<td>35.78323949</td>
<td>-78.68005226</td>
<td>638525.131</td>
<td>225597.783</td>
</tr>
<tr>
<td>53</td>
<td>35.78323949</td>
<td>-78.68005226</td>
<td>638525.131</td>
<td>225597.783</td>
</tr>
<tr>
<td>54</td>
<td>35.78323949</td>
<td>-78.68005226</td>
<td>638525.131</td>
<td>225597.783</td>
</tr>
<tr>
<td>55</td>
<td>35.78323934</td>
<td>-78.68005226</td>
<td>638525.131</td>
<td>225597.766</td>
</tr>
<tr>
<td>56</td>
<td>35.78323934</td>
<td>-78.68005226</td>
<td>638525.131</td>
<td>225597.766</td>
</tr>
<tr>
<td>57</td>
<td>35.78323934</td>
<td>-78.68005208</td>
<td>638525.147</td>
<td>225597.766</td>
</tr>
<tr>
<td>58</td>
<td>35.78323934</td>
<td>-78.68005208</td>
<td>638525.147</td>
<td>225597.766</td>
</tr>
</tbody>
</table>
Appendix B
Navigation Code for the NC State ZTR Robot

B.1 Main Navigation Program Dynamic C Implementation File

/*
GPSPID092805.c

This program is written for the SmartStar controlling the NCSU ZTR mower robot. It uses the BAE_GPS.lib and compass.lib and a Jackrabbit running mowercompass.c which is used to filter and relay the input from the electronic compass. The GPS system is a Trimble AgGPS 132. It should be set to output RMC sentences at 38.4K baud.

For control it uses the mowerdrive.lib which controls the DAC card outputting voltages to voltage controlled actuators connected to the mower’s control levers. This system is designed to work in "record–playback" mode, where the operator drives the desired path once in record mode and the machine stores the path which can then be played back autonomously.

Much of this has been developed over time as part of the previous phases of the mower robot, however, practically all of the functions in this file have been substantially modified for use with the GPS, and the navigation control has been altered to better mimic human driving.

9/28/05
N.B. Powell

Written for the SR9150 using Dynamic C 9.21 */
// A path is an entire collection of NavPoints that makes up a desired autonomous navigation of an area or course.
// Synonymous to course, trajectory, maze, swathing pattern
// A path segment is the line that connects two NavPoints in a path. This is always treated as a straight line, and is the basis of the autonomous navigation behaviors

#ifndef CINBUFSIZE
#define CINBUFSIZE 31
#endif
#ifndef COUTBUFSIZE
#define COUTBUFSIZE 31
#endif
#define RIGHT 513  // Channel 1
#define LEFT 516   // Channel 4
#define ROTATION_TOLERANCE 10  // max angular difference // between heading and desired direction to stop rotation
#define SPINGOOD 12  // max acceptable angular // difference between heading and desired direction
#define TOL 0.5  // Distance from actual point for arrival (in meters)

/* PID Control */
// define weights such that when both are at a maximum allowed error, they cancel
#define K1 1.05  // dist_error weight (default k1 = 1)
#define K2 0.05  // angle_error weight (default k2 = 0.1)
#define SETPOINT 0
#define LMAX 50  // changed from 35
#define LMIN -50

// GPSPosition Structure
// all variables assigned
typedef struct {
    float easting;  // state plane easting in meters  // (x coordinate, always positive)
    float northing;  // state plane northing in meters  // (y coordinate, always positive)
    float compassHeading;  // degrees from true north  // converted from magnetic north by adding magneticVariation
} GPSPosition;

// NavPoint Structure
//northing and easting assigned,
//direction of travel and stop flag calculated

typedef struct NavPoint {
  float northing;
  float easting;
  int directionOfTravel;
  char stopFlag; //1 if the robot should stop at the point
                  //2 if this is the last point
                //0 otherwise
} NavPoint;

//PID Structure
//SetPoint, Proportion, Integral, and Derivative assigned
//LastError, PrevError, and SumError are calculated

typedef struct PID {
  float SetPoint;       //Desired Value
  float Proportion;    //Proportional Const
  float Integral;      //Integral Const
  float Derivative;    //Derivative Const
  float LastError;     //Error\[−1\]
  float PrevError;     //Error\[−2\]
  float SumError;      //Sums of Errors
} PID;

#use "COMPASS.LIB"
#use "MOWERDRIVE.LIB"

/////////Global Variable Section/////////
float proportionalGain;
float integralGain;
float derivitiveGain;
float lastDistToEnd;
float distToEnd;
NavPoint path[500];
int pathIndex;
GPSPosition currentPos;

//variables for the crosstrack error calculation
float NtoDest, EtoDest, pathSegLength;

//at destination flag
//−1 – no destination set
//0 - not at the destination
//1 - at the destination, stop
//2 - overshot the destination, stop
int atDestination;

//Set actuator to be used
//1 - use the left actuator
//2 - use the right actuator
int actuatorFlag, lastActuatorFlag;

void PIDInit (PID *pp){
    memset ( pp,0,sizeof(PID) );
}

float PIDCalc(PID *pp, float NextPoint){
    float dError, Error;
    pp->SumError += (Error = pp->SetPoint - NextPoint);
    if(pp->SumError > LMAX){ // resets integrator
        pp->SumError = 0.5 * LMAX;
        printf("INTEGRATOR\HIGH\RESET\n\n\n\n\n\n");
    }
    if(pp->SumError < LMIN){ // resets integrator
        pp->SumError = 0.5 * LMIN;
        printf("INTEGRATOR\LOW\RESET\n\n\n\n\n\n");
    }
    dError = pp->LastError - pp->PrevError;
    pp->PrevError = pp->LastError;
    pp->LastError = Error;
    return ( pp->Proportion * Error
            + pp->Integral * pp->SumError
            + pp->Derivative * dError );
}

//start_gps_serial is the function which sets up
//communication port C connected to the JackRabbit which
//processes the GPS data.
//10/19/05
//N.B. Powell
void start_gps_serial(){
    serCopen(38400);    //38400 Baud
    serCdatabits(PARAM_SBIT);    //8 data, 1 start, 1 stop
    serCparity(PARAM_NOPARITY);    //No parity bits

    serMode(1);
}

//get_gps_input is a function which reads the data being
//transmitted by the JR processing the GPS data. The JR
//transmits a string of bytes of the format: $ffff*ffff
//where $fff is a 4 byte float, and the first ffff is
//northing and and the second is easting. This function
//waits for this string and stores the pair of values in
//the pos structure passed as the argument of the function
//returns 0 for success, -1 for failure
//10/19/05
//N.B. Powell
int get_gps_input(GPSPosition *pos){
    int input, bytesin, result;
    float nDataIn, eDataIn;
    long j;

    do{input = serCgetc();}
        while(input != '$');
    j = MS_TIMER;
    while(MS_TIMER < j + 5);    //Time delay

    bytesin = serCread(&nDataIn,4,5000);
    if(bytesin != 4)
        printf("Not 1\n");

    do{input = serCgetc();}
        while(input != '*');

    bytesin = serCread(&eDataIn,4,5000);
    if(bytesin != 4)
        printf("Not 2\n");

    if(nDataIn < 0 || nDataIn > 100000)
        return -1;
    if(eDataIn < 0 || eDataIn > 100000)
        return -1;
    pos->northing = nDataIn;
    pos->easting = eDataIn;
    serCrdFlush();
    result = get_heading(pos);
return result;

//calc_direction_of_travel finds the bearing from one //NavPoint to another this involves some logic to convert //between trig angle conventions and compass angle //conventions.
//Input: From: the NavPoint to start from.
//To: the NavPoint destination for this path segment //Returns the direction of travel //08/09/05 //N.B. Powell
int calc_direction_of_travel(NavPoint *from, NavPoint *to) {
    auto float negN, E;  //N and E are component differences
    auto float bearing, angle;  //bearing is compass style //angle is trig style

    negN = from->northing - to->northing;
    E = to->easting - from->easting;
    angle = 180 / 3.14159265 * atan2(negN, E);
    if(E < 0 && negN <= 0) {  //this if deals with converting
        bearing = 450 + angle;  //and the angle references
        //converting a measure of //−180 to 180 anticlockwise to //0 to 360 clockwise.
    } else bearing = 90 + angle;
    return (int)bearing;
}

//populate path stores values in the path array as the user //drives the machine around. this creates the path the //machine will follow autonomously. //Returns 1 for success, 0 for failure //Calls BAE_GPS.lib, compass.lib functions //08/08/05 //N.B. Powell
int populate_path() {
    long k;
    int angle;

    //This loop outputs position updates to the screen and //waits for a user keypress to start recording the path
    while(1) {
        if(get_gps_input(&currentPos) != 0)
            continue;
        if(kbhit()){
            getchar();
        }
    }
}
break;
}

pathIndex = 0;
k = MS_TIMER;
//This loop records the path and outputs the path data
//to the screen. A user keypress ends recording and
//saves the path. Records 1 point per 2 seconds
while(pathIndex < 500){
    if(kbhit()){
        getchar();
        path[pathIndex-1].stopFlag = 2;
        break;
    }
    if(get_gps_input(&currentPos) != 0)
        continue;
    if((MS_TIMER - k) > 3000){
        path[pathIndex].northing = currentPos.northing;
        path[pathIndex].easting = currentPos.easting;
        pathIndex++;
        k = MS_TIMER;
    }
}

//Calculate the direction of travel from point 1 to
//point 2 and store it in point 2’s directionOfTravel
//field and so forth for all points. Set the direction
//of travel to the first point to zero to avoid boundary
//condition problems. The first point’s direction of
//travel and stopFlag must be set in the autonomous code
//prior to beginning navigation.
path[0].directionOfTravel = 0;
for(pathIndex = 1; pathIndex < 500; pathIndex++){
    path[pathIndex].directionOfTravel =
calc_direction_of_travel(
        &path[pathIndex-1], &path[pathIndex]);
    if(path[pathIndex].stopFlag == 2){
        break;
    }

    angle =
        path[pathIndex].directionOfTravel -
        path[pathIndex-1].directionOfTravel;
    if((angle>SPINGOOD||angle<-SPINGOOD||pathIndex==499){
        path[pathIndex-1].stopFlag = 1;
    }
    else path[pathIndex-1].stopFlag = 0;

113
// update_current gets the current position from the GPS and
// calculates the distance to the next point in the course.
// Also sets the atDestination variable
// outputs this information to stdio.
// Returns 1 for success, 0 for failure
// Calls BAE_GPS.lib, compass.lib and external functions
// 09/10/2005
// N.B. Powell

int update_current(){
    while(get_gps_input(&currentPos) != 0);
    lastDistToEnd = distToEnd;
    distToEnd =
        sqrt(
            pow(currentPos.easting-path[pathIndex].easting,2)+
            pow(currentPos.northing-path[pathIndex].northing,2));
    // printf(" Dist to go: %f\n", distToEnd);
    if(distToEnd - lastDistToEnd > 0.3){
        atDestination = 2;
    }
    if(distToEnd < TOL){
        atDestination = 1;
    }
    return 1;
}

// calc_angle_error finds the angle that is the difference
// between the current heading and the heading of the line
// between origin and destination. This is set between
// −180 and 180, with negative meaning turn left and
// positive meaning turn right
// Returns this value
// 08/10/2005
// N.B. Powell

float calc_angle_error(){
    auto float result;
    result = path[pathIndex].directionOfTravel -
            currentPos.compassHeading;
    if(result > 180){
        result -= 360;
    }
    else if(result < -180){
        result += 360;
    }
    return result;
}
// get_dist_to_line calculates the distance from the current
// position to the line which is being followed, the simple
// x,y distance to a line formula.
// NtoDest is end.northing - start.northing,
// EtoDest is end.easting - start.easting,
// where start is path[pathIndex-1]
// and end is path[pathIndex]
// pathSegLength is the length of the path segment
// these globals must be computed each time pathIndex
// is changed.
// Returns dist, the distance to the line
// 01/14/04 (modified 8/10/05)
// Stuart Spencer (N.B. Powell)

float get_dist_to_line(void){
    float dist;
    dist =
        (NtoDest*(path[pathIndex].easting-currentPos.easting)
     -EtoDest*(path[pathIndex].northing-currentPos.northing))
        /pathSegLength;
    return dist;
}

// rotate controls the approximately stationary rotation of
// the machine which it uses to turn so that it is pointed
// toward its destination. since this function is so
// specialized and temporary it only uses the compass and
// directly sets the output voltages which control the
// actuators.
// Returns 1 for success, 0 for failure.
// 8/11/05
// N.B. Powell
int rotate(void){
    float error;
    float fwdVoltsOut, revVoltsOut;
    get_heading(&currentPos);
    error = calc_angle_error();
    printf("Angle Error: %.2f\n", error);
    if(error > 0){ //rotate to the right
        printf("Spin right\n");
while(error > ROTATION_TOLERANCE) {  
//this sets up rotation proportional to angle error  
//based on maximum output 4.0 volts  
//by setting the forward actuator to a fixed value  
//and changing the reverse actuator proportionally  
  revVoltsOut = error*(0.01) + 3.46;  
  if(revVoltsOut > 4.3)  
    revVoltsOut = 4.3;  
  anaOutVolts(LEFT, 3.0);  
  anaOutVolts(RIGHT, revVoltsOut);  
  if(anaInVolts(ChanAddr(2,8)) > 4){  
    stop_mower();  
    return 0;  
  }  
  if(kbhit()){  
    getchar();  
    stop_mower();  
    printf("User Halt\n");  
    return 0;  
  }  
  get_heading(&currentPos);  
  error = calc_angle_error();  
  printf("Angle Error: \%f\n", error);  
}

else{  
  printf("Spin left\n");  
  while(error < -ROTATION_TOLERANCE){  
    revVoltsOut = error*(-0.009111) + 3.5;  
    if(revVoltsOut > 4.26)  
      revVoltsOut = 4.26;  
    anaOutVolts(RIGHT, 3.1);  
    anaOutVolts(LEFT, revVoltsOut);  
    if(anaInVolts(ChanAddr(2,8)) > 4){  
      stop_mower();  
      return 0;  
    }  
    if(kbhit()){  
      getchar();  
      stop_mower();  
      printf("User Halt\n");  
      return 0;  
    }  
  }  
  printf("Spin left\n");}
get_heading(&currentPos);
error = calc_angle_error();
printf("Angle_Error: \%f\n", error);
}
go_forward();
return 1;
}
//navigate_path does the overall autonomous navigation
//control, from preliminary calculations to the path
//navigation loop to the path segment navigation loop
//starts from the mower's current (stationary) location
//and navigates to path[0] then to path[1] and so forth
//until it reaches a NavPoint with any field value <0
//or reaches the maximum possible index value, and stops.
//Calls mowerdrive.lib, external functions
//08/09/05
//N.B. Powell
void navigate_path(){
    auto NavPoint start;
    auto int result;

    //initialize PID variables
    PID sPID;  //PID Control Structure
    float voltageOutChange;  //voltage adjustment(PID Output)
    float xTrackError;  //distance to line(PID Feedback)
    float angleError;  //angle between heading and path
    float totalError;  //aggregate error term

    PIDInit(&sPID);  //Initialize Structure
    sPID.Proportion = proportionalGain;  //Set Coefficients
    sPID.Integral = integralGain;
    sPID.Derivative = derivitiveGain;
    sPID.SetPoint = SETPOINT;  //Set PID Setpoint

    //Preliminary calculations:
    //set pathIndex to zero
    //get current position information
    //calculate the heading from current position to path[0]
    //and set path[0].stopFlag
    pathIndex = 0;
    distToEnd = 10000;
    lastDistToEnd = 10000;
    while(get_gps_input(&currentPos) != 0);
    while(get_gps_input(&currentPos) != 0);
    start.northing = currentPos.northing;
start.easting = currentPos.easting;
path[0].directionOfTravel =
calc_direction_of_travel(&start, &path[0]);
if(path[0].stopFlag != 2){
  angleError =
    path[0].directionOfTravel−path[1].directionOfTravel;
  if(angleError > SPINGOOD || angleError < −SPINGOOD){
    path[0].stopFlag = 1;
  } else path[0].stopFlag = 0;
}

//Path navigation loop
while(pathIndex < 500){
  printf("PI:%d\n", pathIndex);
  printf("N:%f\n", path[pathIndex].northing);
  printf("E:%f\n", path[pathIndex].easting);
  printf("D:%d\n", path[pathIndex].directionOfTravel);
  printf("SF:%d\n", path[pathIndex].stopFlag);
  if(pathIndex > 0){
    NtoDest =
      path[pathIndex].northing−path[pathIndex−1].northing;
    EtoDest =
      path[pathIndex].easting − path[pathIndex−1].easting;
    pathSegLength =
      sqrt(
        pow(path[pathIndex].easting
          −path[pathIndex−1].easting,2) +
        pow(path[pathIndex].northing
          −path[pathIndex−1].northing,2));
  } else{
    NtoDest=path[pathIndex].northing−currentPos.northing;
    EtoDest=path[pathIndex].easting−currentPos.easting;
    pathSegLength=
      sqrt(pow(path[pathIndex].easting−currentPos.easting,2) +
      pow(path[pathIndex].northing−currentPos.northing,2));
  }
  update_current();
  angleError = calc_angle_error();
  if(angleError > SPINGOOD || angleError < −SPINGOOD){
    printf("Must Rotate\n");
    if(!rotate()){
      printf("Bad Rotation\n");
    }
stop_mower();
return;
}
else go_forward();
update_current();
atDestination = 0; //reset the atDestination flag

//Path Segment Navigation Loop
while(1){
  if(kbhit()){
    getchar();
    printf("User Halt\n");
    stop_mower();
    result = 0;
    return;
  }
  update_current();
  if(atDestination == 1){
    if(path[pathIndex].stopFlag == 0){
      pathIndex++;
      NtoDest =
      path[pathIndex].northing
      -path[pathIndex -1].northing;
      EtoDest =
      path[pathIndex].easting - path[pathIndex -1].easting;
      pathSegLength =
      sqrt{
      pow(path[pathIndex].easting
      -path[pathIndex -1].easting ,2)
      + pow(path[pathIndex].northing
      - path[pathIndex -1].northing ,2));
      distToEnd = 10000;
      atDestination = 0;
      printf("
**Next Point**\n");
    }
    else{
      stop_mower();
      printf("Arrived\n");
      result = 1;
      break;
    }
  }
  if(atDestination == 2){
    stop_mower();
    printf("Missed the target\n");
    result = 0;
  }
}
break;
}  
xTrackError = get_dist_to_line();
angleError = calc_angle_error();
totalError = K1*xTrackError + K2*angleError;

lastActuatorFlag = actuatorFlag;

if (totalError < 0){
    actuatorFlag = 2;  // Adjusting right actuator
    totalError = 0 - totalError;
}  
else  actuatorFlag = 1;  // Adjusting left actuator

printf("Distance: \%f, Angle: \%f, Aggregate: \%f\n", xTrackError, angleError, totalError);

// Offset with PID on aggregate error
voltageOutChange = PIDCalc(&sPID, totalError);
printf("PID result: \%f\n", voltageOutChange);
if (voltageOutChange<0){
    voltageOutChange = -voltageOutChange;
}

PID_adjust(actuatorFlag, lastActuatorFlag, voltageOutChange);

}  // end of path segment navigation loop

if (result == 0)
    break;

// if this is the last point, exit
if (path[pathIndex].stopFlag == 2){
    printf("End of Path\n");
    return;
}

pathIndex++;

NtoDest=
    path[pathIndex].northing - path[pathIndex-1].northing;

EtoDest=
    path[pathIndex].easting - path[pathIndex-1].easting;

pathSegLength =
    sqrt(
        pow(path[pathIndex].easting - path[pathIndex-1].easting, 2) +
        pow(path[pathIndex].northing - path[pathIndex-1].northing, 2));


```c
// end of path navigation loop
}

// print_path prints out the recorded path
// 8/11/05
// N.B. Powell
void print_path(){
    for(pathIndex = 0; pathIndex < 500; pathIndex++){
        printf("N: %f
", path[pathIndex].northing);
        printf("E: %f
", path[pathIndex].easting);
        printf("D: %d
", path[pathIndex].directionOfTravel);
        printf("SF: %d
", path[pathIndex].stopFlag);
        if((path[pathIndex].stopFlag < 0 || path[pathIndex].stopFlag > 1))
            break;
    }
}

// input_path_data is an alternative input mechanism that
// gets the navPoints from user input on the terminal
// 10/21/05
// N.B. Powell
void input_path_data(){
    auto char tempbuf[10];
    auto float inputf;
    auto int inputd, angle;

    inputd = 1;
    for(pathIndex = 0; pathIndex < 500; ){
        printf("Point #: %d
", pathIndex);
        printf("Northing: %f
", path[pathIndex].northing);
        printf("Easting : %f
", path[pathIndex].easting);
        printf("Direction: %f
", path[pathIndex].directionOfTravel);
        printf("Stop Flag: %d
", path[pathIndex].stopFlag);
        printf("New Northing\(-1\) for no change:\n");
        inputf = atof(gets(tempbuf));
        if(inputf > 0){
            path[pathIndex].northing = inputf;
        }
        printf("New Easting\(-1\) for no change:\n");
        inputf = atof(gets(tempbuf));
        if(inputf > 0){
            path[pathIndex].easting = inputf;
        }
        printf("Stop flag = 2 for last point or 0 otherwise\nNew Stop Flag: \n");
    }
}
```

121
path[pathIndex].stopFlag = atoi(gets(tempbuf));
printf("Northing: %f\n", path[pathIndex].northing);
printf("Easting: %f\n", path[pathIndex].easting);
printf("StopFlag: %d\n", path[pathIndex].stopFlag);
inputd = atoi(gets(tempbuf));
if(inputd == 1){
    if(path[pathIndex].stopFlag == 2)
        break;
    pathIndex++;
}

//calculate the direction of travel for each path segment
//and set the stop flags
path[0].directionOfTravel = 0;
if(path[0].stopFlag == 2)
    return;
for(pathIndex = 1; pathIndex < 500; pathIndex++){
    path[pathIndex].directionOfTravel =
        calc_direction_of_travel
            (&path[pathIndex-1],&path[pathIndex]);
    if(path[pathIndex].stopFlag == 2){
        break;
    }
    angle = path[pathIndex].directionOfTravel
        - path[pathIndex-1].directionOfTravel;
    if(angle > SPINGOOD || angle < -SPINGOOD || pathIndex == 499){
        path[pathIndex-1].stopFlag = 1;
    } else
        path[pathIndex-1].stopFlag = 0;
}

//input_pid_params is a quick way to change the PID gains
//8/24/05
//N.B. Powell
void input_pid_params(){
    auto char tempbuf[8];

    printf("P: %f\n", proportionalGain);
    printf("NewP: " );
    proportionalGain = atof(gets(tempbuf));
    printf("I: %f\n", integralGain);
    printf("NewI: " );
    integralGain = atof(gets(tempbuf));
    printf("D: %f\n", derivitiveGain);

printf("New D: ");
derivativeGain = atof(gets(tempbuf));
printf("P: %f\n", proportionalGain);
printf("I: %f\n", integralGain);
printf("D: %f\n", derivativeGain);
return;
}

void test_combined_input(){
    while(1){
        get_gps_input(&currentPos);
        printf("Northing: %f\n", currentPos.northing);
        printf("Easting: %f\n", currentPos.easting);
        printf("Heading: %f\n", currentPos.compassHeading);
        if(kbhit()){
            getchar();
            return;
        }
    }
}

int main(){
    auto int input;
    auto char tempbuf[3];
    auto int msgcode;
    
brdInit();
    start_compass_serial();
    start_gps_serial();
    start_dac();
    set_speed_flag(2); // see mowerdrive.lib
    stop_mower();
    
    // Initialize ADC for WKS
    if (msgcode = anaInEERd(ChanAddr(2, 8))){
        printf("Error %d: eeprom unreadable or empty slot: channel 8\n", msgcode);
        exit(0);
    }
    
    // initialize PID coefficients:
    proportionalGain = .09;
    integralGain = 0.004;
    derivativeGain = 0.002;
    
    // initialize the other flags

atDestination = -1;

while(1)
{
    printf("Use numbers 1-0 to select a command:\n\n");  
    printf("(1) Get Position\n");           
    printf("(2) Record Path\n");            
    printf("(3) Print Path\n");             
    printf("(4) Navigate Path\n");          
    printf("(5) Change PID gains\n");       
    printf("(6) Manually Input Path\n");     
    printf("(0) Quit\n\n");                
    printf(" Command to send ----> ");      

    input = atoi(gets(tempbuf));

    switch(input){
        case 1: test_combined_input(); break;
        case 2: populate_path();  break;
        case 3: print_path();      break;
        case 4: navigate_path();   break;
        case 5: input_pid_params(); break;
        case 6: input_path_data(); break;
        case 0: return 0;          break;
        default: printf("Try another key.\n\n");
    }
}
B.2 Actuator Control Dynamic C Library File

/**** BeginHeader */
#ifndef _MOWERDRIVE_LIB
#define _MOWERDRIVE_LIB
/**** EndHeader */

/**********************************************************************************
 Control Program To Steer the Mower
 07–07–2003
 Joe Madren
 Nate Powell

 Using Dynamic C v. 7.33P
 **********************************************************************************/

/**** BeginHeader is_moving */
// is_moving
// 0 – not moving
// 1 – moving
extern int is_moving;
/**** EndHeader */

int is_moving;

/**** BeginHeader turn_left */
// turn_left
// 0 – going straight
// 1 – turning left
// 2 – turning right
extern int turn_left;
/**** EndHeader */

int turn_left;

/**** BeginHeader spin */
// spin
// 0 – not spinning
// 1 – spinning left
// 2 – spinning right
extern int spin;
/**** EndHeader */

int spin;

/**** BeginHeader speed_flag */
// speed_flag
extern int speed_flag;

int speed_flag;

void start_dac(void);
```c
void stop_mower(void) {
  printf("Stop mower\n");
  anaOutVolts(RIGHT, R_NEUTRAL);
  anaOutVolts(LEFT, L_NEUTRAL);
  is_moving = 0;
  turn_left = 0;
}

void go_forward(void) {
  printf("Go forward\n");
  switch(speed_flag) {
    case 0: {
      stop_mower();
      break;
    }
    case 1: {
      printf("Slow\n");
      anaOutVolts(RIGHT, R_FORWARD_SLOW);
      anaOutVolts(LEFT, L_FORWARD_SLOW);
      break;
    }
    case 2: {
      printf("Fast\n");
      anaOutVolts(RIGHT, R_FORWARD);
      break;
    }
  }
}
```
anaOutVolts(LEFT, L_FORWARD);
    break;
}
default: printf("ERROR: Speed Flag set wrong\n"); return;
}
is_moving = 1;
turn_left = 0;

/*** BeginHeader spin_mower_left */
//spin_mower_left turns the mower to the left
//without changing x,y position
void spin_mower_left(void);
/*** EndHeader **/

void spin_mower_left(){
    printf("Spin left\n");
    spin = 1;
    anaOutVolts(RIGHT, R_SPINFORWARD);
    anaOutVolts(LEFT, L_SPINBACK);
}

/*** BeginHeader spin_mower_right */
//spin_mower_right turns the mower to the right
//without changing position
void spin_mower_right(void);
/*** EndHeader **/

void spin_mower_right(){
    printf("Spin right\n");
    spin = 2;
    anaOutVolts(LEFT, L_SPINFORWARD);
    anaOutVolts(RIGHT, R_SPINBACK);
}

/*** BeginHeader is_stopped */
//is_stopped returns 1 if the mower is stopped
//or zero if it is moving
int is_stopped(void);
/*** EndHeader **/

int is_stopped(){
    if(is_moving)
        return 0;
    else return 1;
}
```c
int set_speed_flag(int input)
{
    speed_flag = input;
    return speed_flag;
}
```

```c
int get_speed_flag()
{
    return speed_flag;
}
```

```c
void PID_adjust(int actuator_flag, int last_actuator_flag, float offset)
{
    printf("PID: %f\n", offset);
    if (offset > 0.3) //changed from 0.7
        offset = 0.3;
    if (offset < 0)
        offset = 0;

    //Reset both sides if actuator change occurs
    if (actuator_flag != last_actuator_flag)
    {
        printf("Switched sides\n");
        switch(last_actuator_flag)
        {
            case 1:{ //Reset left
                anaOutVolts(LEFT, L_FORWARD);
                break;
            }
            case 2:{ //Reset right
```

129
anaOutVolts(RIGHT, R_FORWARD);
break;
}
}

switch(actuator_flag){  //Apply computed correction
  case 1:{  //Adjust left actuator
    printf("Actuator: Left\n");
    anaOutVolts(LEFT, L_FORWARD-offset);
    break;
  }
  case 2:{  //Adjust right actuator
    printf("Actuator: Right\n");
    anaOutVolts(RIGHT, R_FORWARD-offset);
    break;
  }
}

/**** BeginHeader */
#endif
/**** EndHeader */
B.3 GPS Input Processor Dynamic C Implementation File

/*
 * mowergps.c
 * A program for the BL1820 Jackrabbit to read data from a GPS unit transmitting a NMEA RMC sentence, convert the data into SPC coordinates, filter it using a running average, and output it via RS-232 on request. For use with the NC State robot mower.

10/11/05
N.B. Powell
*/

#define BINBUFSIZE 31
#define BOUTBUFSIZE 31
#define CINBUFSIZE 31
#define COUTBUFSIZE 31
#define MAXSENCE 100  //Max # of characters from serial

#define TELPERS 0x92,0xD1,0xB6,0x3D,0x1A,0xF2,0xB4,0x3F}
#define K 0x64,0x03,0xE9,0x13,0xBA,0x22,0x69,0x41
#define SINB0 0x85,0x84,0x4F,0xC1,0x2D,0x78,0xE2,0x3F
#define L0 0x83,0x02,0x5E,0x30,0x9B,0x0F,0xF6,0x3F
#define RB 0xAF,0x96,0xBF,0x32,0x15,0x8C,0x61,0x41
#define E0 0x99,0xBB,0x96,0x70,0x82,0x9A,0x22,0x41
#define RADF 0x7A,0xC6,0x52,0xA2,0x46,0xDF,0x91,0x3F
#define LOGE 131

const double ELP =
{ 0x92, 0xD1, 0xB6, 0x3D, 0x1A, 0xF2, 0xB4, 0x3F};
//0.081819191042831
const double K =
{ 0x64, 0x03, 0xE9, 0x13, 0xBA, 0x22, 0x69, 0x41};
//13178320.622194
const double SINB0 =
{ 0x85, 0x84, 0x4F, 0xC1, 0x2D, 0x78, 0xE2, 0x3F};
//0.5771702552412
const double L0 =
{ 0x83, 0x02, 0x5E, 0x30, 0x9B, 0x0F, 0xF6, 0x3F};
//1.378810109076
const double RB =
{ 0xAF, 0x96, 0xBF, 0x32, 0x15, 0x8C, 0x61, 0x41};
//9199785.5932115
const double E0 =
{ 0x99, 0xBB, 0x96, 0x70, 0x82, 0x9A, 0x22, 0x41};
//609601.2199
const double RADF =
{ 0x7A, 0xC6, 0x52, 0xA2, 0x46, 0xDF, 0x91, 0x3F};
//0.0174532925199 = Pi/180
const double LOGE =

131
const double BASE_E =
{ 0xA3, 0x83, 0x2B, 0x65, 0x47, 0x15, 0xF7, 0x3F };  // 0.693147180560
const double dPONE =
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F };  // 1.442695040889
const double dPTWO =
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40 };  // 1
const double sixEFive =
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x4F, 0x22, 0x41 };  // 600000
const double twoEFive =
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6A, 0x08, 0x41 };  // 200000

// This structure holds geographical position as reported by a GPS receiver.
// Use the gps_get_position function below to set the fields.
typedef struct
{
  float easting;  // state plane easting in meters
  // (x coordinate, always positive)
  float northing;  // state plane northing in meters
  // (y coordinate, always positive)
} spcPosition;

spcPosition mowerPos;

// start_serial_ports opens the two RS-232 ports, B and C.
// B is connected to the GPS, C is connected to the SmartStar's serial port D.
// 10/11/05
// N.B. Powell
void start_serial_ports();

// gps_parse_coordinate is helper function for splitting xxxx.xxxx into decimal degrees
// returns 0 if succeeded
int gps_parse_coordinate(char *coord, float *decimalDegrees);

// Converts a Lat/Lon position into State Plane coordinates
// using the constants defined above for the particular state plane zone. This function is configured for the Lambert projection for North Carolina.
//PARAMETER1: pos - the structure to store the coordinates
//PARAMETER2: lat - the latitude of the point
//PARAMETER3: lon - the longitude of the point

int convert_to_state_plane
    (spcPosition *pos, float lat, float lon);

//get_gps_input reads the input from the gps and converts
//it into SPC coordinates
//Passes pos: the position in SPC coordinates
void get_gps_input(spcPosition *pos);

//output_to_smrtstar is a function to do serial output to
//the Smart Star on port C. It outputs the global variable
//mowerPos when it receives a signal character
//from the Smart Star.
void output_to_smrtstar();

int main(void){
    spcPosition posMatrix[3];
    long j;

    start_serial_ports();

    get_gps_input(&posMatrix[0]);
    get_gps_input(&posMatrix[1]);

    while(1){
        get_gps_input(&posMatrix[2]);
        mowerPos.northing =
            (posMatrix[0].northing+posMatrix[1].northing
             +posMatrix[2].northing)/3;
        mowerPos.easting =
            (posMatrix[0].easting+posMatrix[1].easting
             +posMatrix[2].easting)/3;
        posMatrix[0].northing=posMatrix[1].northing;
        posMatrix[0].easting=posMatrix[1].easting;
        posMatrix[1].northing=posMatrix[2].northing;
        posMatrix[1].easting=posMatrix[2].easting;
        printf("N:\%fn", mowerPos.northing);
        printf("E:\%fn", mowerPos.easting);
        output_to_smrtstar();
        j = MS_TIMER;
        while(MS_TIMER < j + 100); //Time delay
    }
    return 0;
}
void start_serial_ports()
{
    serBopen(38400);  //38400 Baud to GPS
    serBdatabits(PARAM_8BIT); //8 data bits, 1 start, 1 stop
    serBparity(PARAM_NOPARITY); //No parity bits
    serCopen(38400);  //38400 Baud to SmartStar
    serCdatabits(PARAM_8BIT); //8N1 as usual
    serCparity(PARAM_NOPARITY);
    return;
}

int gps_parse_coordinate
(char *coord, float *decimalDegrees)
{
    auto char *decimalPoint;
    auto char temp;
    auto char *dummy;
    auto float minutes;
    auto float degrees;

    decimalPoint = strchr(coord, '.');
    if (decimalPoint == NULL)
        return -1;
    temp = *(decimalPoint - 2);
    *(decimalPoint - 2) = 0;  //temporary terminator
    degrees = atoi(coord);
    *(decimalPoint - 2) = temp;  //reinstate character
    minutes = strtod(decimalPoint - 2, &dummy);
    *decimalDegrees = degrees + minutes / 60;
    return 0;
}

int convert_to_state_plane
(spcPosition *pos, float lat, float lon){
    double B, L, sinB, Q, R, gamma, N, E, Qsub1, Qsub2;

    // B=dpFloat2Double(lat)*PI/180;
    B=dpMul(dpFloat2Double(lat), RADF);
    // L=dpFloat2Double(lon)*PI/180;
    L=dpMul(dpFloat2Double(lon), RADF);
    sinB=dpSine(B);
    // Qsub1=(1+sinB)/(1-sinB)
    Qsub1=dpDiv(dpAdd(dpONE, sinB), dpSub(dpONE, sinB));
    // Qsub2=(1-ELPS*sinB)/(1-ELPS*sinB)
    Qsub2=dpDiv(dpAdd(dpONE, dpMul(ELPS, sinB)),
               dpSub(dpONE, dpMul(ELPS, sinB)));
    // Q=(LOG_E*dbLog2(Qsub1)-ELPS*LOG_E*dbLog2(Qsub2))/2;
Q = \text{dpDiv}(\text{dpSub}(\text{dpMul}(\text{LOG}_E, \text{dpLog2}(Q_{sub1})), \\
\text{pMul}(\text{dpMul}(\text{ELPS}, \text{LOG}_E), \text{dpLog2}(Q_{sub2}))), \text{dpTWO})

// R = k/\exp(Q \times \text{SIN}_B0);
R = \text{dpDiv}(K, \text{dpExp2}(\text{dpMul}(\text{BASE}_E, \text{dpMul}(Q, \text{SIN}_B0))))

// \gamma = (L_0 - L) \times \text{SIN}_B0
\gamma = \text{dpMul}(\text{dpSub}(L_0, L), \text{SIN}_B0)

// N = RB - R \times \text{dpCosine}(\gamma)
N = \text{dpSub}(RB, \text{dpMul}(R, \text{dpCosine}(\gamma)))

// E = E_0 + R \times \text{dpSine}(\gamma)
E = \text{dpAdd}(E_0, \text{dpMul}(R, \text{dpSine}(\gamma)))

// N = N - 200000 \text{ Remove most significant digit }
N = \text{dpSub}(N, \text{twoEFive})

// E = E - 600000 \text{ Remove most significant digit }
E = \text{dpSub}(E, \text{sixEFive})

\text{pos->northing} = \text{dpDouble2Float}(N);
\text{if}(\text{pos->northing} < 0 || \text{pos->northing} > 100000)
\quad \text{return} -1;
\text{pos->easting} = \text{dpDouble2Float}(E);
\text{if}(\text{pos->easting} < 0 || \text{pos->easting} > 100000)
\quad \text{return} -1;
\text{return} 0;

\}

\text{void get\_gps\_input(spcPosition *newpos)}
\quad \text{int i; long j; char received[16]; char sentence[MAX\_SENTENCE]; char *reference; int inputChar; int stringPos; float latitude, longitude;}

\quad i = 0;
\quad \text{stringPos} = 0;
\quad \text{while}(1)\{
\quad \quad \text{inputChar} = \text{serBgetc}();
\quad \quad \text{if}(\text{stringPos} == 0 && \text{inputChar} != '\$')\{
\quad \quad \quad j = \text{MS\_TIMER};
\quad \quad \quad \text{while}(\text{MS\_TIMER} < j + 1); // Time delay
\quad \quad \quad \text{continue};
\quad \quad \}\n\quad \quad \text{if}(\text{inputChar} == 'r' || \text{inputChar} == 'n')\{
\quad \quad \quad \text{sentence}[\text{stringPos}] = 0; // add null
\quad \quad \quad \text{printf("start \_parse \n");
\quad \quad \\}
reference = sentence;

if(strlen(reference) < 4){
    printf("err\0\n");
    break;
}

if(strncmp(reference , "$GPRMC", 6) == 0){
    //parse the 2nd and 4th fields of the
    //RMC sentence which has 11 fields
    for(i = 0; i < 11; i++){
        reference = strchr(reference , ',');
        if(reference == NULL){
            printf("err\1\n");
            break;
        }
        reference++;
    } //first character in field
    //pull out data
    if(i == 2){ //latitude
        if(gps_parse_coordinate(reference , &latitude)){
            printf("err\2\n"); //get_coordinate failed
            break;
        }
    }
    if(i == 4){ //longitude
        if(gps_parse_coordinate(reference , &longitude)){
            printf("err\3\n");
            break; //get_coordinate failed
        }
        printf("Lat:\%f\n", latitude);
        printf("Lon:\%f\n", longitude);
        if(convert_to_state_plane
            (newpos, latitude, longitude)){
            printf("err\4\n");
            break;
        }
        printf("Nor:\%f\n", newpos->northing);
        printf("Est:\%f\n", newpos->easting);
        return;
    }
}

if(inputChar > 0){
    sentence[stringPos] = inputChar;
    stringPos++;
    if(stringPos == MAX_SENTENCE)
stringPos = 0;  // reset string if too large
}
}
serBrdFlush();
}

void output_to_smrtstar()
{
    serCputc ('$');
    serCwrite(&mowerPos.northing, 4);
    serCputc ('$');
    serCwrite(&mowerPos.easting, 4);
    return;
}
B.4 Compass Input Dynamic C Library File

```c
/** BeginHeader */
#ifdef _COMPASS_LIB
#define _COMPASS_LIB
/** EndHeader */

/* START LIBRARY DESCRIPTION */
Library for the SmartStar on the NCSU BAEB mower robot, using an RS-232 link from channel D on the SmartStar to channel C on a Jackrabbit running movercompass.c and connected to a Honeywell HMR3200 compass unit
08/11/2005
N.B. Powell

Written for an SR9150 Using Dynamic C v. 9.21P

END DESCRIPTION */

/** BeginHeader start_compass_serial */
void start_compass_serial() {
    serDopen(19200);       // 19200 Baud
    serDdatabits(PARAM_8BIT); // 8 data bits, 1 start, 1 stop
    serDparity(PARAM_NOPARITY); // No parity bits
    serMode(1);
}

/** EndHeader */

/* GETTING THE COMPASS HEADING */
//get_heading retrieves and parses the heading from the compass provided that the compass is in continuous output mode (default) and stores the heading in the referenced
```
/**GPSPosition**

```c
int get_heading(GPSPosition *pos);
/** EndHeader */

int get_heading(GPSPosition *pos){
    int i, input, value;
    char number[8];
    float result;

    if(serDputc('!')){
        return -1;
    }

    do{
        input = serDgetc();
        while(input != '$');
    }while(serDputc('!')){
        i = 0;
        while(input = serDgetc()){
            if(input == '$'){
                i = 0;
                continue;
            }
            if(input == '*'){
                number[i] = '\0';
                break;
            }
            if(input < 48 || input > 57)
                continue;
            number[i]=(char)(input);
            i++;
        }
    }

    value = atoi(number);
    pos->compassHeading = (float)(value)/10;
    return 0;
}

/** BeginHeader */
#endif
/** EndHeader */
```
B.5 Compass Input Processor Dynamic C Implementation File

/*
mowercompass.c

A program for the BL1820 Jackrabbit to read data from a
Honeywell 3200 electronic compass, use a running average to
filter the data, and output it via RS-232 on request. For
use with the NC State robot mower.

7/27/05
N.B. Powell
*/

#define BINBUFSIZE 31
#define BOUTBUFSIZE 31
#define CINBUFSIZE 31
#define COUTBUFSIZE 31

float avgHeading;

// start_serial_ports opens the two RS-232 ports, B and C.
// B is connected to the electronic compass, C is connected
// to the SmartStar's serial port D.
// 7/27/05
// N.B. Powell
void start_serial_ports();

// get_heading reads the input from the electronic compass
// and converts it into a float
// Returns result: the heading measured by the compass
float get_heading();

// output_to_smrtstar is a function to do serial output to
// the Smart Star on port C. It outputs the global variable
// avgHeading when it receives a signal character from
// the Smart Star.
cofunc void output_to_smrtstar();

int main(void){
    float heading[3];
    start_serial_ports();
}
heading[0] = get_heading();
heading[1] = get_heading();

while(1){
    heading[2] = get_heading();
    avgHeading = (heading[0]+heading[1]+heading[2])/3;
    heading[0] = heading[1];
    heading[1] = heading[2];
    printf("A:%f\n", avgHeading);
    costate{
        wfd_output_to_smrtstar();
    }
}
return 0;
}

void start_serial_ports(){
    serBopen(19200); //19200 Baud to compass
    serBdatabits(PARAM8BIT); //8 data bits, 1 start, 1 stop
    serBParity(PARAMNOPARITY); //No parity bits
    serCopen(19200); //19200 Baud to SmartStar
    serCdatabits(PARAM8BIT); //8N1 as usual
    serCParity(PARAMNOPARITY);
    return;
}

float get_heading(){
    int i;
    long j;
    char received[16];
    int input;
    float result;

    i = 0;
    while(1){
        input = serBgetc();

        j = MS_TIMER;
        while(MS_TIMER < j + 1); //Time delay

        if(input >= '0' && input <= '9') //if input is a numeral
            received[i] = input; //enqueue it into the buffer
        i++;
        continue;
    }

    if(input == '.'){ //the decimal point
        result = 0.0;
        i = 0;
        j = MS_TIMER;
        while(MS_TIMER < j + 1); //Time delay

        if(input >= '0' && input <= '9') //if input is a numeral
            result = result * 10 + (input - '0'); //enqueue it into the buffer
        i++;
        continue;
    }
    return result;
}
received[i] = input; // put the decimal in the buffer
i++;
input = serBgetc(); // get the char after the decimal
received[i] = input;
i++;
received[i] = '\0'; // add the terminating null
result = atof(received); // parse the number to a float
i = 0;
if (result > 360){
    result = -1;
    continue;
}
else return result;
}
}

cofunc void output_to_smrtstar()
{
    int intValue, test;
    char buf[8];

    wfd{test = cof_serCgetc();}
    if(test != '!' ){
        printf("%x\n", test);
        return;
    }
    printf("%x\n", test);
    intValue = (int)(avgHeading*10);
    itoa(intValue, buf);
    serCputc('$');
    serCputs(buf);
    serCputc('*');
    test = 0x35;
    printf("%s\n", buf);
    return;
}