Development of a Low Level Autonomous Machine

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Sciences

In the Department of Agricultural and Bioresource Engineering

University of Saskatchewan

Saskatoon

By

Jason Griffith

Keywords: autonomous machine, tractor controller, vehicle navigation

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Agricultural and Bioresource Engineering

University of Saskatchewan

Saskatoon, Saskatchewan S7N 5A9

ABSTRACT

An autonomous machine is a machine that can navigate through its environment without human interactions. These machines use sensors to sense the environment and have computing abilities for receiving and interpreting the sensory data as well as for controlling their displacement. At the University of Saskatchewan (Saskatoon, Canada), a low level autonomous machine was developed. This low level machine was the sensor system for an autonomous machine. The machine was capable of sensing the environment and carrying out actions based on commands sent to it. This machine provided a sensing and control layer, but the path planning (decision making) part of the autonomous machine was not developed.

This autonomous machine was developed on a Case IH DX 34H tractor with the purpose of providing a machine for testing software and sensors in a true agricultural environment. The tractor was equipped with sensors capable of sensing the speed and heading of the tractor. A control architecture was developed that received input commands from a human or computer in the form of a target heading and speed. The control architecture then adjusted controls on the tractor to make the tractor reach and maintain the target heading and speed until a new command was provided. The tractor was capable of being used in all kinds of weather, although some minor issues arose when testing in rain and snow. The sensor platform developed was found to be insufficient for proper control. The control structure appeared to work correctly, but was hindered by the poor sensor platform performance.

ACKNOWLEDGMENTS

I would like to acknowledge my supervisor, Dr. Martin Roberge, who provided me with guidance and support throughout this project. I would also like to thank my advisory committee Dr. Claude Laguë and Dr. Charles Maulé who gave me council and advice during my studies. Funding for this project was provided by Auto 21 and the tractor that was used was provided by CNH. CNH also supplied a field for testing. I would like to recognize my Auto 21 counterpart at the Université de Sherbrooke, Patrick Frenette and his supervisor Dr. François Michaud. Both these researchers provided me with insight and knowledge into the world of robotics. Special thanks to Christopher Griffith, Kevin Hall and the staff of the Agricultural and Bioresource Engineering Department of the University of Saskatchewan who helped me whenever necessary.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE BLOCKS

LIST OF ABBREVIATIONS

ALV	Autonomous Land Machine
ASCII	American Standard Code for Information Interchange
CNH	Case New Holland
CORBA	Common Object Request Broker Architecture
DOF	Degrees of Freedom
GPS	Global Positioning System
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
JNI	Java Native Interface
LIDAR	Light Detection and Ranging
MARIE	Mobile and Autonomous Robotics Integration Environment
MEMS	Micro-Electro-Mechanical Systems
NMEA	National Marine Electronics Association
PD	Proportional Derivative
PID	Proportional Integral Derivative
RC	Radio Control
RF	Radio Frequency
RMI	Remote Method Invocation
SPA	Sense-Plan-Act
UTM	Universal Transverse Mercator coordinate system
WAAS	Wide Area Augmentation System

CHAPTER 1: INTRODUCTION

1.1 Topic Introduction

Agricultural machinery has played an ever important role in modern agriculture for many years. With growing populations and increased urbanization the demand for larger and more efficient agricultural machines has grown (Noguchi et al. 2002). Agricultural machines have become increasingly comfortable for the operator to reduce operator strain and allow operators to manage the machine for longer periods of time. To improve efficiency and reduce operator strain many systems such as Trimble Autosteer, headland management and steering assist are available on the market. These systems provide the operator with more feedback and less required input which reduces the driver strain. These systems also allow for more accuracy over the day, which reduces misses and overlap increasing the efficiency and reducing application costs.

A common cause of inefficient field practices during an application is human mistakes (Ollis and Stentz 1997). These mistakes are often caused by poor operator feedback, driver fatigue or low visibility due to factors such as dust, darkness and fog. Autosteer and headland management systems reduce these human mistakes by providing a type of semi-autonomous machine in which periodic operator input is still required. With semi-autonomous machines being recognized through the use of autosteer and headland management, the next step forward is to have fully autonomous machines.

The idea of having autonomous agricultural machines is not a new one (Wilson 2000). For many years people have been attempting to create autonomous machines in one form or another. As machines improved many attempts at autonomous control has been made with some attempts

being successful, while others were not. In recent years, improvements in technology have greatly enhanced sensors available for use on autonomous machines. Many of the sensors that are now available are very precise and can be used to provide accurate information about a machine's current state. Coupling these sensors with the greatly improved computing ability of modern day computers has made it possible to create fully autonomous machines.

The study and development of autonomous machines is a rapidly growing area of research (Reid et al 2000). Large amounts of research and testing has been put into autonomous machine development because it has many advantages over human operated machines. An autonomous machine does not require human interaction, which means it can operate for many hours at a time without making mistakes or failures. An autonomous machine can be used to perform very precise actions which can be repeated at rates much more rapidly than if a human were doing them.

The challenge of developing a fully autonomous machine is that it must be able to sense its surroundings and use sensory inputs to determine what actions should be carried out (Brooks 1986). An example of this is when an autonomous machine traveling across a field encounters an obstacle. The machine must sense the obstacle and determine if it is capable of continuing on its current path or if it should travel around the obstacle. The machine must then carry out the actions. If the vehicle traveled around the obstacle, it must correct its course and continue to where it was traveling. These actions are second nature to human operators, but developing them into a machine can be very challenging.

Because the actions an autonomous machine must carry out are quite often similar to those of a human, the structure of an autonomous machine can be viewed similar to that of a human. In a human you have the five main senses that record data from the person's surroundings. This data

is then processed and analyzed in a human's brain. The brain makes a decision on what actions should be carried out and sends the commands to muscles to carry out the actions. In an autonomous machine there are sensors that sense the environment, a computer that analyzes the data to make necessary decisions and actuators or motors that are sent commands on what actions they should carry out (Gat 1998).

Often the sensors, actuators and computing portion of an autonomous machine can be divided into the high and low levels. The high level is similar to the machine's brain because there is a computer which has path planning and object avoidance software. The high level makes decisions to be carried out based on sensor data. The low level is the actuators and sensors. This low level has relatively little computing and without a high level software the low level is not capable of performing very many tasks. The low level is necessary for providing the high level a scheme to interact with the machine and its environment.

1.2 Project Introduction

For this project, the low level design for an autonomous machine was developed and tested. The development of this low level design included sensor development, analysis of data from the sensors, translation of decision commands into machine commands and converting the commands into movements. The developed design was implemented and tested on a Case IH Farmall DX-34H tractor.

The overall design of the autonomous machine consisted of a low level controller and a high level navigation/path planning software application. The high level software was developed at the Université de Sherbrooke in the Department of Electrical Engineering and Computer Engineering. The developer was Patrick Frenette, under the supervision of Dr. François Michaud. The high level software was only partially implemented on the machine. The low

level design included software necessary to receive data from sensors, convert data into a usable form, determine the machine attitude from data and pass that to high level software. The low level was also capable of accepting control commands in the form of speed and heading corrections from high level software. These commands were then translated and used to make appropriate adjustments to the machine. The low level design did not include any path planning or object detection.

CHAPTER 2: LITERATURE REVIEW

2.1 Previous Agricultural Autonomous Machines

The 20th century saw more rapid agricultural machine development than any other historical time period (Britannica 2007). In the early 1900's agricultural machinery quickly advanced following the invention of the internal combustion engine. Agricultural machines have continued to advance, improving production and agricultural efficiency. One of the ideas being developed on agricultural machinery is the idea of autonomous machines. Although this concept is currently being developed and some forms of it are just recently reaching the market, the concept behind it is not a new one. As early as the 1920's a patent for a tractor capable of following furrows based on mechanical linkages existed (Reid et al 1998). Crude attempts at autonomous agricultural machines continued to be invented, but few of these were very successful.

In the 1950's and 1960's automatic control began to grow within the process control industry (Wilson 2000). The potential of automatic control quickly began to be realized in other industrial areas and agriculture was no exception. It was quickly realized that the concepts of automatic control had a wide scope of uses, especially since at this time feedback control was becoming more refined (Wilson 2000). Until the 1970's the majority of the attempts at autonomous agricultural machines used mechanical setups. During this time a system that used wires to carry current throughout the field for a machine to follow was invented (Reid et al 1998). Later in the 1980's a beacon system was used in fields for autosteering systems (Wilson 2000).

In the 1980's with improvements in imagery analysis and sensors it became possible to have vision based guidance systems on machines. This became a popular method for guidance and research groups began developing agricultural machines that were guided based on vision sensors (Reid et al 1998). During this same time as vision based guidance was improving, other methods of vehicle sensing became applicable. The use of wheel position sensors, ground tracking and feelers became useful to add corrections and improved vehicle estimations (Wilson 2000).

In the 1990's Carnegie Melon University developed a vision based autonomous harvester for alfalfa (Ollis et al 1997). This harvester was just one of many autonomous agricultural machines at this time. It used traditional row following techniques based on machine vision to efficiently cut the alfalfa. Ollis et al (1997) also explored the idea of using differentially corrected GPS, which was becoming available at this time. It was decided GPS had too many failures at this time to be applicable, but later when GPS was improved the harvester was modified to become a combination of vision based row following and GPS navigation (Pilarski 2002).

There are many examples of autonomous agricultural machines that utilize GPS and an assortment of other sensors. Some examples are a robotic tractor developed by Noguchi et al. (2002) and a tractor guided by differential GPS (Bell 2000). Vision based guidance is still being explored, but there are some critics of it saying that it is too vulnerable to light levels, dust and other factors that may influence its effectiveness (Wilson 2000).

2.2 Sensors on Autonomous Machines

For the development of autonomous machines a reliable sensor platform is one of the most important components. A sensor platform must be capable of providing accurate and reliable information at high data rates (Payton 1986). With each machine being designed for different

tasks, a different set of sensors is required. There is a wide array of sensors that can be used depending on the type of environment, type of tasks to be carried out, allowable budget and the overall goals of the machine. A machine that operates at high speeds or in small spaces requires sensors that are more accurate and have higher update rates (Payton 1986). Machines operating in larger spaces at lower speeds can often operate effectively with lower accuracies and slower update rates.

Some of the most common navigation sensors used on autonomous machines is a digital compass, Global Positioning System (GPS), Differential Global Positioning Systems (DGPS), and Inertial Measurement Unit (IMU). All of these sensors are available from a variety of manufacturers and all come with different characteristics. There are also many object avoidance sensors that are being used on machines. Some of the object avoidance sensors are laser scanners, ultrasonic devices, infra red devices, computer vision and feelers.

GPS is perhaps one of the most common navigation sensors used today. It has uses in many different applications and one of its uses is for vehicle navigation, which makes it very useful on autonomous machines. GPS is a constellation of 24 satellites orbiting the earth transmitting information that can be received by GPS receivers (Crawford 2005). The first GPS satellite was launched in 1977. Originally it was only for military use, but in 1984 GPS was made available to the general public (Arnold et al 2000). A GPS receiver must locate a minimum of four satellites and determine the distance between itself and each of these satellites. The distance between a receiver and satellites is determined by measuring the time that a signal takes to travel between the satellite and receiver. The receiver then uses trilateration[1] to determine its location

[1] Trilateration is similar to triangulation except it uses distances to find a location instead of angles

on earth (Kaplan 1996). Once the position is determined, this data can be used for calculating other parameters such as speed and heading.

GPS has been used for previous agricultural machine navigation. At Stanford, researchers were able to develop a tractor that used only a GPS for its navigational data and was capable of traveling along spirals, arcs and arbitrary curves. The tractor was also tested along steeply sloped terrain (Bell 2000). Using only a GPS is a rare case for autonomous machines, and normally other sensors are included to make the machine navigate more effectively. One of the downfalls with GPS is the low refresh rate (Wilson 2000). Current GPS receivers have increased the GPS refresh rate, making GPS more useful in control applications. One of the highest refresh rates found on a GPS receiver is 20Hz because of the high amount of signal processing and data configuration required (Farrell et al 1998). Using GPS receivers in combination with other sensors that have higher update rates is another way to more effectively utilize GPS.

An IMU is a sensor that is capable of measuring multiple degrees of freedom by combining accelerometers and gyroscopes into one package. A characteristic that makes an IMU useful is that it is self-contained and only relies on physical laws of motion. This makes an IMU more robust to interference than most navigational sensors (Farrell et al 1998). One of the largest drawbacks to using an IMU is the amount of error they accumulate in short periods of time. The error is caused by accelerometer and gyroscope drift and the amount of drift the sensor experiences relies largely on the quality of the sensor (Crawford 2005).

Due to inherent problems with using only an IMU, they are often coupled with a GPS to provide improved results for both sensors. A GPS relies on line of sight operation, which means that a GPS is not capable of receiving a signal if an object such as a building or tree is between the receiver and satellite (Braasch et al. 1999). For this reason a GPS may experience outages in

which it can not provide navigational data. An IMU on the other hand is capable of continually providing data at high rates, but it requires some kind of correctional device such as a GPS. When these two sensors are used in tandem the GPS provides a correction for the IMU and the IMU can replace the GPS for short periods of time, such as if a signal failure temporarily occurs. The IMU can also be used to provide data between GPS refreshes. This is possible because an IMU can have an update rate of 100 Hz (Bevly 2004) while a GPS commonly only has a 20 Hz or less update rate (Farrell et al 1998).

There are many examples where these two sensors have been used on agricultural machines. One such example involved using a GPS and IMU fusion algorithm to control a tractor for tillage, planting, cultivating and spraying on soybean fields (Noguchi 2002). Each of the tests also involved the tractor transferring itself between a storage shed and the field where the work was to be performed. These tests were based on previously developed maps. A large amount of effort has gone into the problem of examining if a low cost IMU and GPS are capable of performing better than one high cost GPS. Linsong (2002) applied this to a tractor and found that using a Kalman filter to combine the GPS and IMU data resulted in reduced errors from when only an expensive GPS was used.

Another sensor that can be used to complement a GPS and IMU is a digital compass. Digital compasses are an extension of traditional compasses that uses the earth's magnetic north to determine what the machine's current heading is. Digital compasses have no moving components which make them robust and they are capable of providing more accurate heading estimates than traditional compasses (Farrell et al 1998).

A compass is useful in replacing a GPS at very low speeds or when the machine is stationary. Because a GPS uses it's location to calculate other parameters such as heading and speed it can

not know what its current heading is without some movement (Witte and Wilson 2004). This means that the GPS is not transmitting heading or speed data when there is no motion. For this reason a digital compass is useful for finding the heading when stationary or moving very slowly.

The main drawback with a compass is the unknown variable errors caused by external magnetic interference (Crawford 2005). This magnetic interference can be caused by ferrous materials in the surroundings, nearby electrical equipment or engines operating within close proximity. For this reason a digital compass must be mounted in an isolated area, usually above all other materials and away from excessive vibrations. Newer digital compasses are capable of being calibrated for their surroundings to account for some of the hard-iron effects[2].

2.3 Control Architectures for Autonomous Machines

With any autonomous vehicle no matter what the purpose is there must be a clear interface between components to structure the system along the flow of information from sensors to effectors (Bicho and Schöner 1997). This flow of information can be structured in many ways, but the main objective is to separate each machine task into clear and distinct layers. The way that each layer is broken up and how they perform their tasks is often set up to try and optimize certain parameters. With some systems the parameter focused on is reduced computational power, others focus on simplicity of software, while others will focus on the speed at which decisions can be determined. "A common way of dealing with highly complex systems is via hierarchical decomposition of activities to be performed by the autonomous vehicle, and consequently the introduction of a hierarchy of control and decision layers." (Frazzoli 1999) A

[2] Hard iron is the errors caused by permanent magnets or iron in close proximity to the compass. Hard-iron effects are not varying and can be calibrated for.

few common methods for decomposing autonomous vehicles over the years will be briefly discussed.

Common control architectures for autonomous machines from the early 1980's usually developed three layers (Gat 1998). The vehicle was broken into a low, high and middle layer. The low layer software carried out direct communication with vehicle controls. It was said to be the stability and control center. It was often linked with mechanisms that could control components on the vehicle, such as steering or speed. The middle layer was the mode transitioning stage. It was basically for controlling communication between the high layer and low level layers. It directed information and delivered it to appropriate locations. The high level layer was considered a situation and reactive layer. It interpreted sensory data to determine what actions should be carried out and communicated with the low level, through the middle layer, to carry out required actions.

Shortly after the initial three layer approach was introduced, a similar architecture called the Sense-Plan-Act (SPA) approach was introduced (Gat 1998). Similar to the previous three layer approach, the breakdown was to have an execution, planning and sensing layer. The execution layer was similar to the low level and it was necessary for taking a plan developed by the other two layers and generating actions to reach the plan's goal. The planning stage took a model and goal developed by the sensing level and developed a plan to achieve this goal. The sensing level translated sensor data into a real world model and used this to generate a goal.

SPA was a simple method for monitoring the flow of control between components. The flow was always unidirectional and linear, which made the execution much faster and easier to follow. For developers this approach was also very easy to work with conceptually because it was similar to the execution of a computer program (Gat 1998). Having a set flow of control did

have its drawbacks. The system was tightly coupled which meant the process of modifying the system became more involved. Making a change could often require completely recoding parts of the software.

SPA was a slow design which generally worked through open loop control. The approach relied on models of its environments which had to be inputted by the user before the vehicle could navigate. For this reason SPA was not capable of executing in an unknown environment because it was difficult to determine accurate real world models very quickly (Gat 1998).

Later, in the mid-1980's, development of the subsumption architecture (Brooks 1986) "touched off a firestorm of interest in autonomous robots" (Gat 1998). Subsumption was considered to be a much needed diversion from SPA. It attempted to move away from the open ended control and became considered the first "reactive planner" (Gat 1998). Subsumption was sometimes viewed as a radical departure from SPA, when really it attempted to make SPA more efficient by applying task-dependent constraints to the layers. The major shift from SPA was that the layers were composed of networks of finite state machines (Brooks 1990). The downfall of Subsumption was that it was not at all modular and it was quite unreliable in unknown environments due to its reliance on models.

There were many different reactive planners, each being very similar overall, but with many differences at the core of their development. Between layers were relative feedback control mechanisms which acted as verification to ensure data was being delivered to proper components. The layers often, but not always, relied on past data. This was also often used with current data to make future predictions. Using past data to estimate future predictions affected how the vehicle made decisions. This effectively made for a slow, deliberate planner that

reacted well to usual environmental changes, but not very well to very drastic changes (Gat 1998).

The architectures developed in the 1980's were considered classical approaches. They relied on a world model, which limited the tasks that could be carried out. Later in the 1980's the classical approach became somewhat more reactive by developing models based on sensor input. "The general idea was to sense the world, build a model, plan actions with respect to goals, and then execute the plan via motor controller commands." (Yu 2004) These architectures proved to be inefficient and ineffective in unknown or changing environments because of their reliance on real world models.

Early in the 1990's, and still continuing today, is the development of reactive planners, which react to sensor data. Reactive planners do not rely on models, but instead react directly to sensors, which allow them to operate in unknown environments (Yu 2004). Reactive planners are believed to be more advanced than classical approaches because they account for a continually changing environment.

One such example of a reactive planner is Frazzoli's hybrid controller. A hybrid controller is based on the idea of breaking the control into two layers that are discrete in nature. The layer closest to the actual machine is used for interaction with the machine through mechanical connections and sensing the environment through sensors. This layer has a high bandwidth, to the extent of being able to be considered continuous in time for design purposes. The other layer is used for making logical decisions based on collected data. This layer has a lower bandwidth and operates as a discrete time system. The combination of discrete and continuous dynamics is what forms the concept of a hybrid controller (Frazzoli 1999). The concept of a hybrid

controller developed from other hierarchical approaches. A hybrid controller is designed to meet certain operating parameters and for this reason there are different kinds of hybrid controllers.

Quite often the layers of a system are distributed over multiple computers. Having distributed computing allows each computer to focus on one specific task and perform that task as efficiently as possible. Each layer can also be sub-divided to operate on multiple computers. Having this approach maximizes the computing capabilities, but it often does not use the computers to their full efficiency. When too many computers are introduced the slowest component of the system is the data transfer between computers (Goos 1998). For this reason the number of computers should be minimized.

To combine all the computers into one large process the computers must be capable of communicating with one another. Data must be continually passed from one computer to another to provide updates. This can be done through practices such as socket connections, serial connections or with middleware software. Socket connections can be slow and it means some sort of communication protocol must be developed. If a computer or component is changed the communication protocol must also be re-implemented. Serial connections are generally much too slow and require a communication protocol to be developed. High speed serial connections can be purchased, but they are very expensive and are subject to higher levels of errors than most communication methods. Middleware is a type of software program designed to allow for an easily configurable communication protocol between components. Middleware software is often developed for use with distributed computing for computers with different programming languages and operating systems. Middleware encourages the sharing and reuse of code, to speed up the development process (Coté et al 2004). Some examples of middleware software are MARIE and CORBA. MARIE (Mobile Autonomous Robot Integrated

Environment) was developed by François Michaud and his research group at the Université de Sherbrooke. CORBA (Common Object Request Broker Architecture) is opensource object oriented software developed by many developers.

To date there have been many partially autonomous agricultural machines that have focused on different areas of autonomy. Each individual area of autonomy (such as the sensor platform, controlling platform, path planning and object avoidance) has seen advancements, but currently there are very few machines that successfully incorporate all the systems that are required for an autonomous vehicle. This project aims at providing a sensor and control platform that is successful at maneuvering a tractor. By providing these two platforms it becomes possible for other researchers to develop the path planning layer and to continue to develop the object avoidance layer. By incrementally building each layer and thoroughly testing them it should be possible to develop a successful, fully autonomous machine.

CHAPTER 3: OBJECTIVES

The goal of this project was to design an agricultural tractor that could maneuver through previously programmed tests carried out in a field without human interaction. The end result was a low level autonomous machine that could be used as a test bed for high level software. The overall objective was therefore to develop and evaluate a low level control system for an autonomous tractor. More specific objectives were:

- To develop and evaluate a system capable of determining vehicle attitude based on feedback from sensors.
- To develop and evaluate an actuator control system capable of controlling the machines heading and speed.
- To evaluate the low level control system performance in terms of vehicle response to sensor input.

CHAPTER 4: SYSTEM DESCRIPTION

To implement the partially autonomous machine a tractor was used. The tractor had to be modified to incorporate all required sensors, actuators and controllers. The sensors and actuators for the machine were previously purchased so a design had to be developed to incorporate these devices into the machine. The motor controllers had to be researched and purchased. Once they were purchased their mounting and power requirements had to be incorporated. Aside from the physical components a software package had to be developed for collecting data and performing tractor control. A computer, along with some microcontrollers was installed on the tractor for implementing the software. This chapter discusses the final components on the tractor and how they were incorporated. The component specifications and requirements are discussed and some of the component capabilities are also discussed.

4.1 System Components

- Case IH Farmall DX-34H tractor (Section 4.2)

  This was the tractor used for implementing the system on
- Novatel DK-Flexpak SSII 5Hz GPS receiver development kit (Section 4.3.1)

  This was used for collecting position data, speed and heading
- PNI Corporation TCM 2.6 digital compass (Section 4.3.2)

  The compass was used for determining the tractor's heading
- Crossbow Inertial Systems IMU300CC (Section 4.3.3)

  The IMU was used for measuring pitch, roll, yaw, lateral (forward) acceleration, horizontal (side) acceleration and vertical acceleration

The IMU was installed and used for data collection, but was not incorporated into the control system

- SICK LMS-221 laser scanner (Section 4.3.4)

  The laser was used for detecting objects within a 180º range in front of the tractor

  The laser was installed and used for data collection, but was not used for tractor control

- Custom made speed sensor (Section 4.3.5)

  The speed sensor was developed to determine the tractor's speed

- Control System (Section 4.4)

  The control system controlled the tractor by using sensor data to calculate changes and then sending these changes to actuators for adjusting the tractor. The actuators provided a means to translate the voltages produced by the motor controller into a mechanical method for physically moving components on the tractor.

- Software (Section 4.5)

  Software was developed for testing and tuning the system. This software provided a simple method of interacting with the tractor. Middleware software was also developed to allow path planning software to interact with the control system.

4.2 Mechanical System and Instrumentation

To implement the system a Case IH Farmall DX-34H tractor was used (Figure 4.1). The tractor had a hydrostatic drivetrain with three gear selections and a hydraulic steering system. The tractor was capable of operating in either two wheel drive or four wheel drive. For this project it was used in two wheel drive to maintain consistency between tests. Using four wheel drive may have been beneficial when operating in the mud and snow, but it may also have

affected steering in dry conditions. A comparison between operating in two wheel drive versus four wheel drive was not performed. To be able to install all necessary components a few modifications had to be made to the tractor. The operator seat was removed to provide adequate space for installing a DC linear actuator for controlling the hydrostatic lever as shown in Figure 4.2. The steering wheel and steering column had to be removed and were replaced by a DC rotary actuator with a worm gear reduction system for increasing the motor torque and reducing the motor's rotational speed. An adapter had to be manufactured to adapt the motor to fit into the tractor's steering pump spleens (Figure 4.3). This adapter was necessary for the rotary actuator to be able to drive the tractor's steering motor.

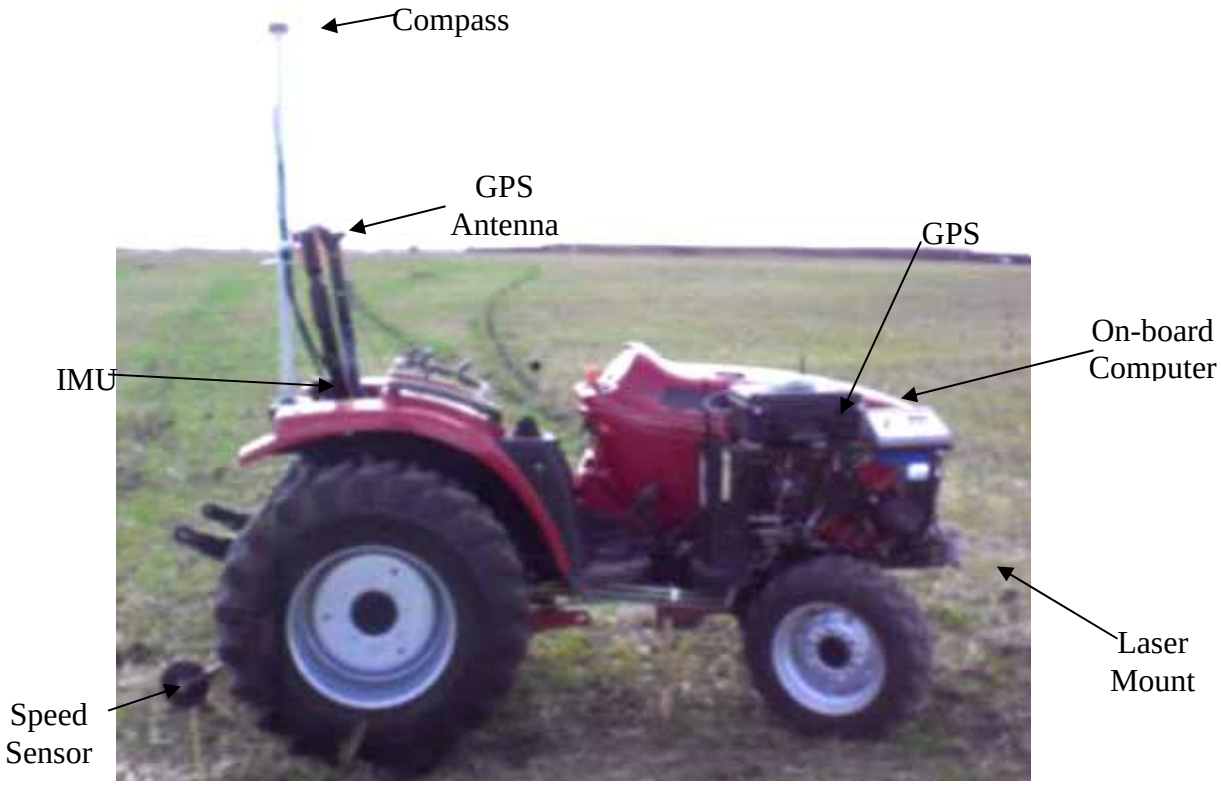

Figure 4.1: The Case IH Farmall DX-34H tractor with sensor locations

Figure 4.2: The tractor with the seat removed and the hydrostatic DC actuator installed

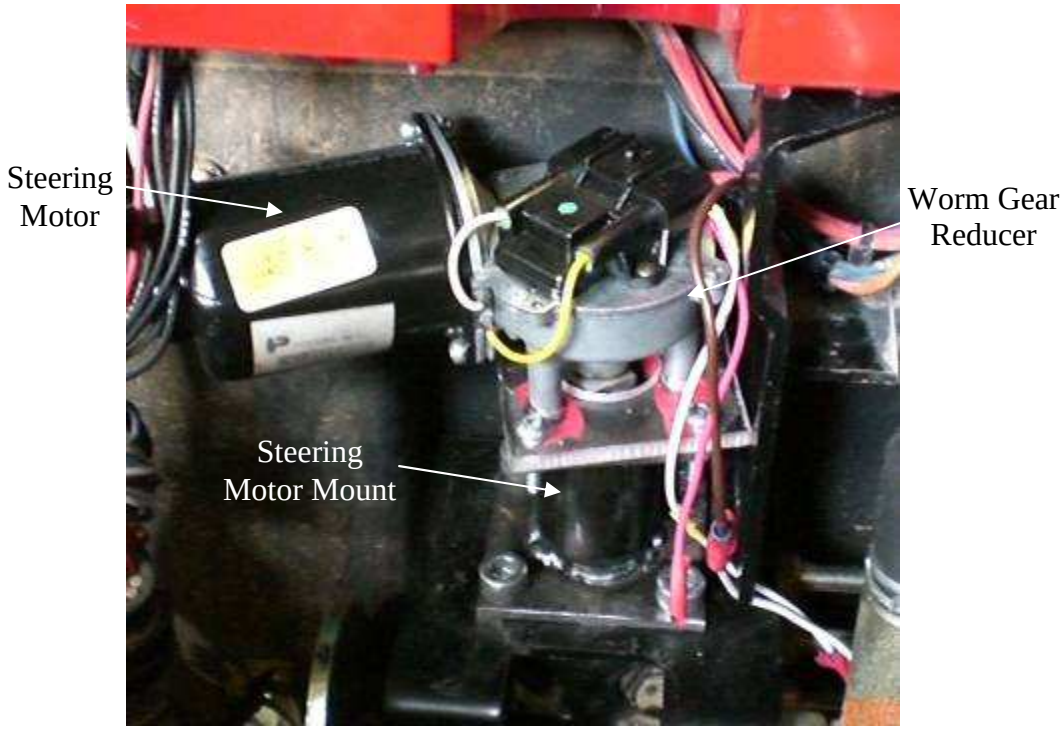

Figure 4.3: The steering apparatus installed on the tractor

If required, an actuator could have been placed onto the gear shift to allow the computer to shift the tractor or to allow the tractor to be shifted remotely. The actuator was not installed on the tractor because for this project it was not necessary. For the tests that were carried out it was possible to place the tractor in a chosen gear and leave it there for the entire test. The ignition also could have been modified to be operated by the computer or remotely, but this was not a necessity. A remote shutdown was designed and installed to disable the tractor if a malfunction occurred. This shutdown was a remote switch that was wired into a preexisting safety switch. The switch was activated from a remote control panel that was carried whenever the tractor was operating.

For mounting the computer and other electronics a platform was manufactured to mount on the front-end loader mounts. This situated the components in a position that made it simple to work on. The top three-point hitch linkage was removed, as well as the slow moving vehicle sign to install another mount for the IMU and compass. The compass and IMU were mounted onto a weight that was on top of isolators. This isolator weight reduced vibrations experienced by the sensors, which improved their data. The SICK laser mount was modified so the mounting holes would line up with holes that were already on the front-end loader frame. The top half of the tractor's roll hoop was removed because it was made of steel, which caused magnetic interference for the compass.

A deep cycle battery was added to supplement the tractor's battery. This was added because during engine cranking, the starter drew high amounts of current from the battery, reducing the amount of voltage available for the autonomous system. This would cause the autonomous system to reset. Adding a second battery provided insurance that the system would not have electrical failures during engine cranking. The two batteries were connected in parallel with a

diode installed to allow the tractor's alternator to charge both batteries. This configuration also allowed the tractor's starter to draw current from only one battery (Figure 4.4). With this configuration the autonomous system could receive power from both batteries and when the alternator was operating it was possible for both batteries to charge simultaneously.

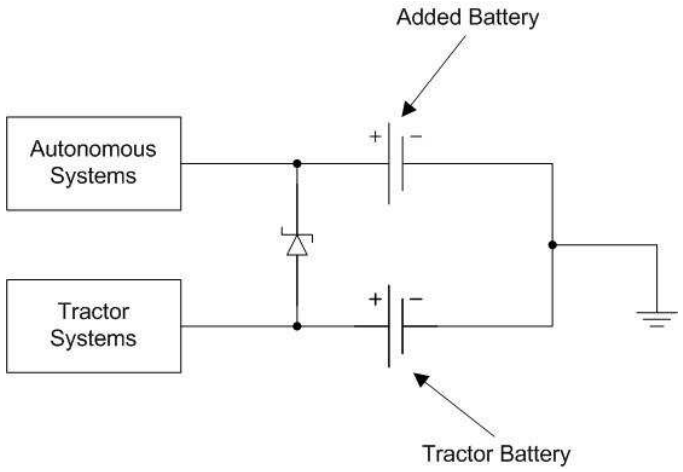

Figure 4.4: Wiring layout for the tractor's batteries

4.3 Sensing System

4.3.1 Global Positioning System (GPS)

The tractor's sensor platform was necessary to determine the tractor's heading and speed. There were multiple methods for determining these parameters and as such multiple methods were used. The most used sensor on the tractor was a Novatel DK-Flexpak SSII 5Hz GPS receiver development kit. The GPS receiver was capable of operating using the Wide Area Augmentation System (WAAS) differential correction and provided accuracy to within 1.5 meters. The GPS reported data in either National Marine Electronics Association (NMEA) format or binary over an RS232 connection. Only NMEA format was used in this architecture. The receiver was designed for on vehicles, which meant its power input could be from 6 V to 18 V and the receiver had an internal fuse as well as a fuse in the power chord. The data that was

used from the GPS receiver was geographic position, speed estimation and heading estimation. Other data was available from the GPS, but it was not utilized.

The GPS receiver was a development kit, purchased already installed in a mounting case. The receiver was installed under the computer and power for it came directly from the 12 volt batteries. The antenna for the GPS had to be as high as possible to prevent other objects from obstructing the signal. When the tractor Roll Over Protection (ROP) was removed a small platform was created on top of the lower portion of the ROP. The antenna had a magnetic base so it sat directly on top of this platform. The compass was still higher than the GPS antenna, but it was small enough that it did not interfere with GPS signals.

4.3.2 Compass

To complement the GPS, a TCM 2.6 digital compass manufactured by PNI Corporation was used. This compass was a tilt compensated compass which means it was capable of providing a heading, even when tilted to an angle up to 70˚. The compass also had accelerometers for measuring roll and pitch, and a thermometer for measuring temperature. The mount built for the compass caused large movements of the compass, which introduced errors into the pitch and roll measurements. For this reason the pitch and roll measurements were not used. All data was sent from the compass over an RS232 cable in ASCII format.

When the compass was received it was not in a protective casing so a case was built that incorporated a mount into it. The compass was mounted onto a plastic tube that was 122 cm long, which put the compass 55 cm above any metal objects. The plastic tube was mounted on top of a 50 pound weight that had dampeners under it to reduce vibrations. The power requirement for the compass was 5 V so a 5 volt regulator was prepared to protect the compass

from receiving too much voltage. The compass power line also had a fuse installed to prevent power surges.

When the compass was installed on the vibration reducing mount it still received errors due to vibrations. To reduce errors the compass internal filtering algorithm was used. The internal filter applied a time constant to the raw data before the heading was calculated. The filtering provided a more stable reading, but made the data acquisition slower. The damping rate was set to 16.

4.3.3 Inertial Measurement Unit (IMU)

The IMU that was used was an IMU300CC manufactured by Crossbow Inertial Systems. The IMU measured six degrees of freedom as shown in Figure 4.5 and could also measure temperature. The IMU used three bulk micro-machined vibratory Micro-ElectroMechanical Systems (MEMS) sensors for measuring angular rate and three micro-machined silicon MEMS devices for its accelerometers. Because all the sensors were MEMS based sensors the IMU could experience 30 meters of drift error in approximately 20 seconds (Crawford 2005). The IMU also had a digital signal processor to attempt to compensate for deterministic error sources.

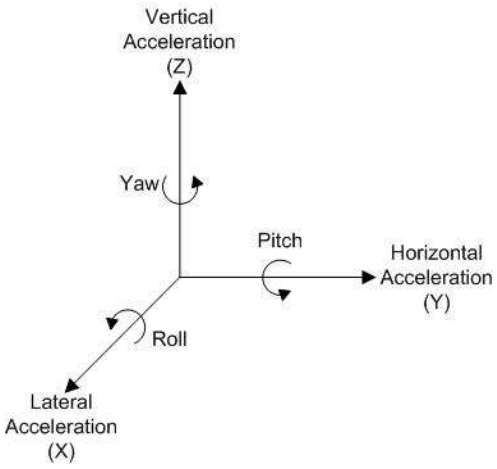

Figure 4.5: The IMU coordinate system showing all six DOF

The voltage requirement for the IMU was 9-30 V so the power connection that was installed was a line from the battery with a switch and fuse installed. The IMU was mounted on the same dampening mount that the compass stand was on. This was necessary because with the engine operating and tractor stationary, the vibrations were high enough to saturate the IMU sensors. This meant that when the tractor started moving there was no way to extract any useful data because the accelerometers were saturated.

By installing the IMU on a dampening mount the vibrations were reduced, but the forces could still be measured. When performing preliminary tests it was found that the accelerations experienced by the tractor were so small that errors within the signal saturated the sampled data. Figure 4.6 shows two graphs of collected data when a typical test path was traveled. Both data sets are from the same test where the speeds were between 0 and 1.2m/s and the tractor turned to travel at multiple headings. From the figure it can be seen that the lateral acceleration was extremely small and it was saturated by engine noise and field vibrations. The yaw rate was also quite noisy, but it was possible to detect when the tractor turned. Because of time constraints the IMU was not incorporated into the sensing platform. IMU data was still collected for each test with the goal of incorporating the IMU into the sensing platform in the future.

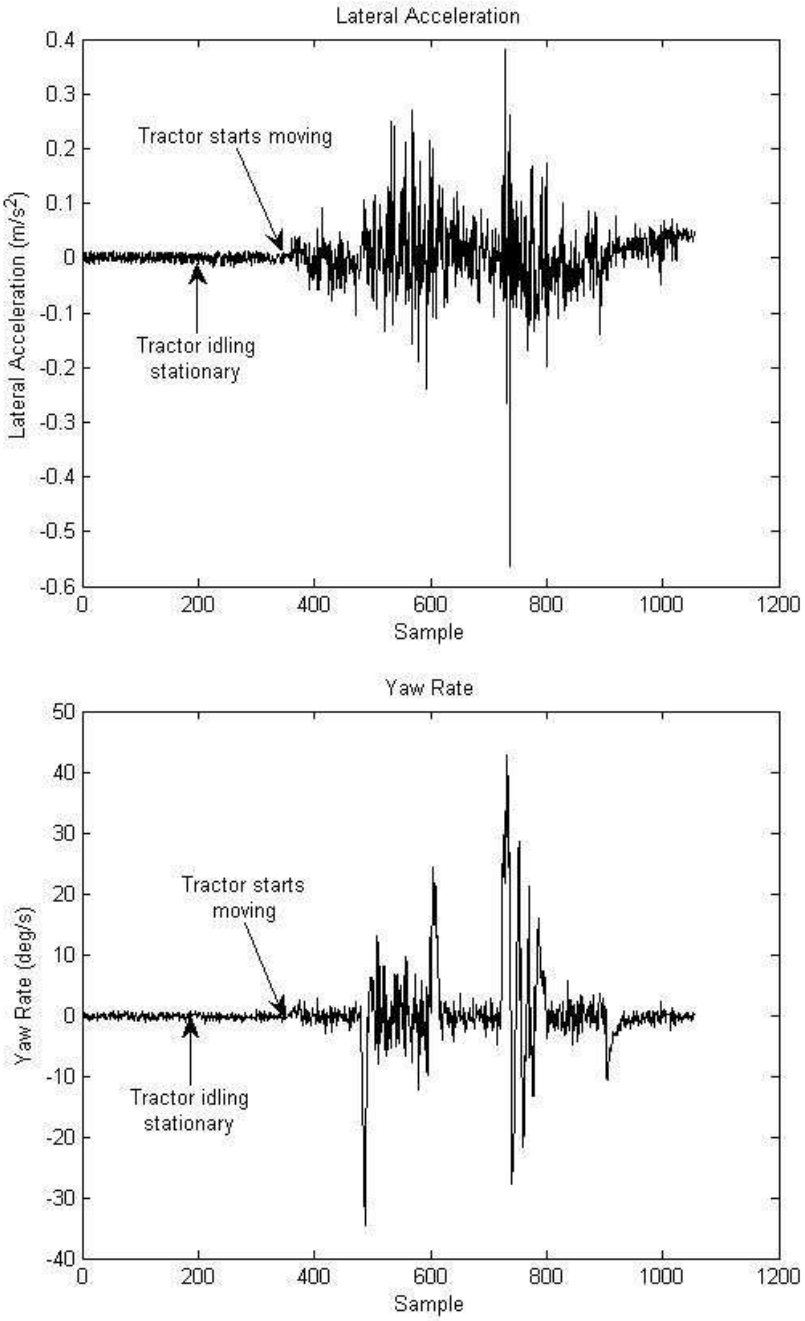

Figure 4.6: Lateral acceleration for speeds between 0 and 1.5m/s and yaw rate when traveling at different headings

4.3.4 Laser Obstacle Detection

The laser used on the tractor for obstacle detection was a SICK LMS-221 (Laser Measurement System). The LMS-221 is an outdoor laser that scans an 180˚ range and can be set

for 1, 0.5 or 0.25˚ angular resolution. The laser could scan up to 80 meters and had a 10 mm resolution. The data interface was an RS232 connection. The laser's power requirement was 24 V with a current draw of 6 Amperes. For this reason an external power supply capable of providing 24 V was installed on the tractor. The mount for the laser was modified so it would bolt onto the tractor's front-end loader mounts. This put the laser at a height of 55 cm off the ground, in front of the tractor with no obstructing obstacles. A shield to protect the laser from obstacles and to allow for a sun shield to be installed was also constructed. The laser was not used for object avoidance in this project, but data was collected to be used in the future development of obstacle avoidance algorithms.

4.3.5 Speed Sensor

To provide a second speed reference a speed sensor was developed. The sensor consisted of a drop wheel with a 48 tooth sprocket and a digital Hall Effect sensor. The drop wheel was mounted to the drawbar of the tractor and every time a gear tooth passed the Hall Effect sensor a digital pulse was sent to a counting board. This board counted the pulses and every 200 ms outputted the number of pulses to a computer over an RS232 connection. To convert the pulses into a speed (m/s), equation 4.1 was used. In this equation the pulses were multiplied by four because each number of pulses was over a 250 ms time duration. It was then divided by 48 because there were 48 teeth on the sprocket. The wheel diameter was 17.7 cm so the circumference of the drop wheel was 0.558 m. This meant the equation then had to be multiplied by 0.558 m. This conversion was carried out in the software. The Hall Effect sensor required 12 V and the counting board required 5 V to power them.

$$velocity = \frac{4 * pulses}{48} * 0.558 \tag{4.1}$$

The reason this type of speed sensor was chosen was that it was a low cost, simple sensor. It was a very simple method for performing a second speed measurement. After preliminary testing of the speed sensor it was found that the sensor reported a speed that was lower than the actual speed. The wheel that was used for this sensor had a low mass and when traveling at low speeds the wheel did not turn smoothly. The result of the wheel not turning smoothly was a low speed reading. The wheel did not rotate as well when it became covered in mud and straw. Another problem caused by the wheel having a low mass was at higher speeds, when the wheel hit a bump it could bounce off the ground and momentarily slow down. The result of this was once again a low speed reading. For this reason it was found that the speed sensor was only reliable at speeds less than circa 0.3 m/s. Later during the testing period the speed sensor became even less reliable because it was damaged.

4.4 Control System

The tractor's control architecture was broken into two separate systems. One control system was for controlling the steering while the other was for hydrostatic control. These systems were treated as completely decoupled systems. In actual fact, these two systems could be considered to be coupled because the tractor's rate of turning changes as speed changes. An optimal steering controller should take into account both the required heading change and the vehicle speed, but for simplicity only the required heading change was accounted for in the steering controller.

Both control systems consisted of similar components. There were sensors to measure the parameters being manipulated, a computer that used sensor data and determined any required changes, a motor controller that received control signals from the computer and converted the signals into required pulse widths, and actuators that controlled the mechanical system. The same computer and motor controller were used in each control system but the controllers were still separate. On the computer two separate controllers were operating simultaneously and the motor controller had two separate channels, but only one PID controller.

The motor controller that was used was a Roboteq, model AX2550. This motor controller was capable of controlling two motors on separate channels. The motor controller had two inputs, one for each channel, and two outputs, one for each actuator. The inputs for the motor controller were unit less values between ±127. The motor controller could be setup for either speed control or position control. When the motor controller was in speed control the values between ±127 signified a target speed for the actuator to move. When it was in position control the values represented a position that the actuator should be held at. The motor controller was used in position control for this project.

When the motor controller received a control input (value between ±127) it converted that input into a pulse that was outputted by its pulse width modulator. The output of the motor controller was ±12 V and the current could reach as high as 120 Amperes. To control the actuator direction the motor controller switched the output voltage polarity.

The motor controller had two analog channels for receiving actuator feedback. When the motor controller was used with feedback it utilized an internal PID controller for maintaining proper settings. A PID controller is a closed loop controller where the Proportional function (P) is used for determining the amount of change that must be made based only on the calculated

controller error and preset gain. The proportional control affects the time taken to reach steady state (rise time), but it also affects the system's steady state error and the system overshoot. The Integral component of the controller (I) is used for making small changes to the system to get it closer to the setpoint. The Integral control is for reducing the amount of steady state error. The final component is the Derivative component (D), which is used for controlling the amount of overshoot that occurs, or simply how aggressively the controller approaches the setpoint.

The same PID controller was used for each channel which meant that the PID had to be tuned to try and achieve adequate control of both channels. To get the PID controller set to a level that could provide satisfactory control of both the steering and hydrostatic systems, manual tuning was used. The manual tuning method used was trial and error. This method sets the integral and derivative functions to zero while the proportional function was set as high as possible. The proportional function was reduced until the two systems became stable. Integral and derivative controls were then introduced to try and reduce the overshoot and oscillation.

For the steering controller, the computer controlled the heading by sending commands to the motor controller for adjusting the front wheel angle. Data from the compass was used by the computer if the tractor speed was below 0.7 m/s. If the tractor was traveling faster than 0.7 m/s the computer used GPS data to determine the current heading. The computer also used potentiometer feedback to determine the tractor's current front wheel angle with respect to the tractor's chassis. To measure the front wheel angle a potentiometer was mounted on the right side front wheel's kingpin axis. This provided a voltage feedback that corresponded to a wheel angle. From these inputs the controller could determine how the front wheels should be adjusted to achieve the target heading.

The steering controller on the computer was a PD controller. Originally it was designed as a PID controller, but there was little performance variation between using a PID or PD controller so a PD controller was used. The controller received a target heading and compared that to the tractor's current heading. Based on this, the controller determined the heading error which was then used to determine which way the tractor had to rotate to have the correct heading.

This controller could not calculate an error term by using a simple target heading minus the current heading because this problem was a circular problem instead of linear. In a typical controller when calculating a controller change, if the setpoint is larger than the current plant setting the plant setting must be increased to reach the setpoint. If the sepoint is less than the current plant setting the plant setting should be decreased. In the steering controller the plant setpoint was the target heading and the plant setting was the current heading. With this controller the target and current headings could be any value between 0˚ and 360˚. When calculating the required change the tractor may have to turn either to the left or the right to reach the target heading in the least amount of time. For this reason a coordinate system such as the one in Figure 4.7 was setup to be used for calculating the error.

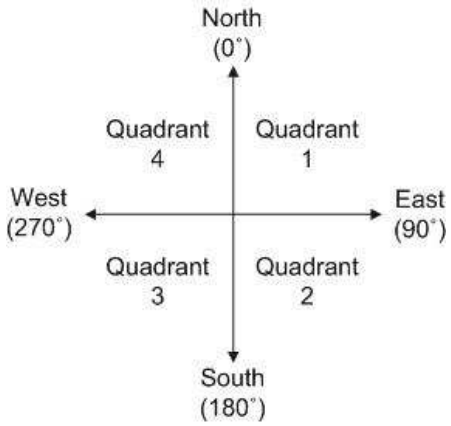

Figure 4.7: Coordinate setup for calculating heading changes

The way the coordinate system in Figure 4.7 was used was that the tractor's current heading was used to determine what quadrant the tractor was in. The quadrant that the target heading was in had to also be determined. The controller used this to determine which direction the tractor should turn and what the error term was. This error term was then used in the PID controller. Some pseudo code examples of finding the quadrants and using the quadrants to calculate error terms can be found in 0and a block diagram of the error calculation is shown in Figure 4.8.

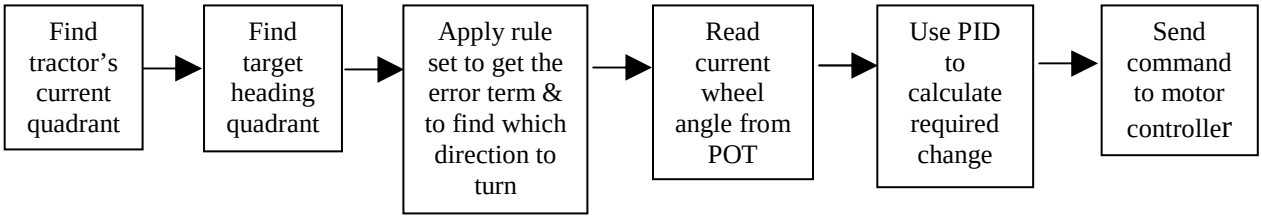

Figure 4.8: Block diagram of the steering controller on the embedded computer

Once the error term was known a basic PD controller was used. For the PD controller the current wheel angle had to be known. To find this wheel angle the potentiometer mounted on the front wheel was read. This was read by using an analog to digital converter within the motor controller. With the error term and current wheel angle known the PD calculations could be performed to determine what the new control signal should be. This control signal was a value between ±127 and was applied to the motor controller.

The motor controller used its internal PID controller with the potentiometer feedback to determine what voltage had to be applied to move the wheel to its required angle. The voltage was outputted by the motor controller and applied to a rotational actuator that was installed in such a way to replace the steering wheel. This stimulated the hydraulic steering system which caused the wheels to turn to the target angle. The heading controller on the embedded computer

was updated every 200 ms and the PID controller within the motor controller was updated every 16 ms (Roboteq 2005).

The speed controller had the same structure as the steering controller, but the speed controller's error calculation was much simpler and the controller used was a PID controller instead of a PD controller. A PD controller was used because it was found that the Integral portion of the controller was not required to remove the steady state error. The steady state error was quite small. By not including the integral portion slightly decreased the time to calculate a controller change. To calculate the error, the tractor's current speed as well as the hydrostatic actuator position was used. The tractor's current speed was measured using the speed sensor if the speed was below 0.3 m/s and if the speed went above 0.3 m/s the controller used the GPS speed. The hydrostatic actuator position was measured from a potentiometer built into the actuator.

Calculation of the hydrostatic error term was straight forward. If the current speed was less than the target speed the actuator had to be extended so error was positive. If the current speed was greater than the target speed the actuator had to be withdrawn and error was negative. Once the error was calculated and the current actuator position was read through the motor controller's analog to digital converter, the PID control could be applied. The result of this was the new actuator position which could then be sent to the motor controller. The motor controller calculated the required voltage and output that voltage to an actuator that was attached to a hydrostatic lever. This actuator pushed or pulled the hydrostatic lever as required, causing the tractor's speed to increase or decrease.

4.5 Software Overview

Two software packages had to be developed for this project. The first package (ALV Analyzer) was used for development and testing of the vehicle. Once this was working appropriately its user interface and other unnecessary components were removed to make it a skeleton package. It was then transformed into a package (ALV Middleware) that could receive commands and send information to and from higher level software. Both ALV Analyzer and ALV Middleware are discussed here.

4.5.1 Testing Software

The software package developed for implementing the tractor and testing the algorithms was called ALV Analyzer. ALV Analyzer was required for the development and testing of the tractor's autonomous system. The software was originally developed for use on a custom built autonomous land machine, but because the software was designed for use on multiple machines it was easily adapted to the tractor. The software included a layer for communicating with sensors, a layer for providing vehicle control and a data logging layer that was all made accessible through a GUI. The software was developed with the ability to be operated on the tractor's embedded computer or it could be operated on a client/server type setup.

When operated as a client/server system, the server operated on the tractor's embedded computer while the client operated on a remote computer. The client and server operated over a wireless network and allowed for remote control of the vehicle as well as remote data logging and viewing of data while the tractor was operating. For the remote capabilities, Java Remote Method Invocation (RMI) was utilized. A general overview of the software package can be seen in Figure 4.9. The package was broken into two functional units being the server (embedded computer) and the client (remote computer).

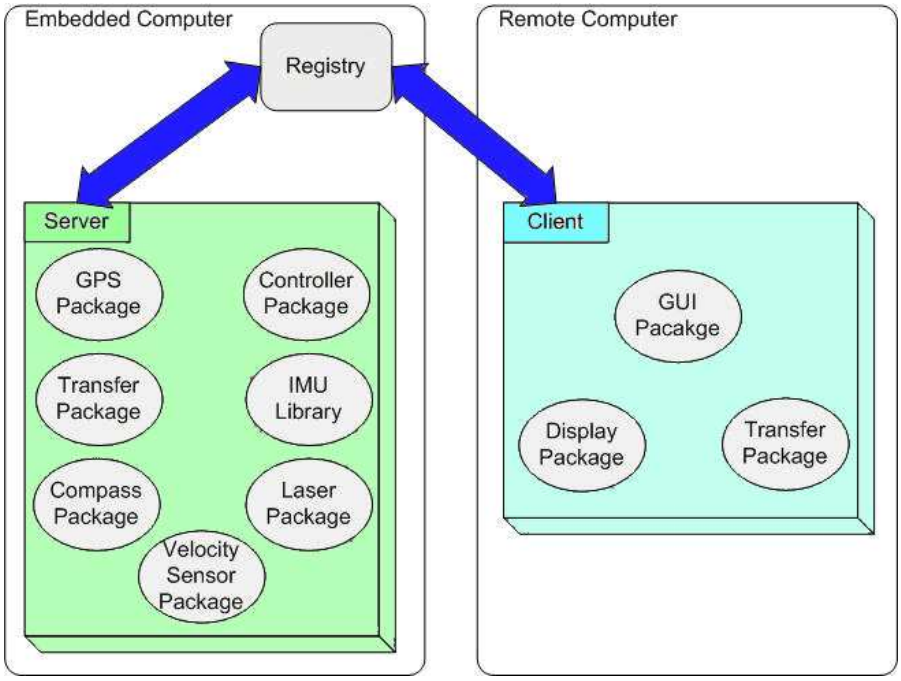

Figure 4.9: Overview of ALV Analyzer software package

4.5.1.1 Onboard software

The onboard software was designed to operate on the embedded computer in a Linux environment with kernel version 2.6.8. The main purpose of the server was to collect sensor information to transfer to the client as well as receiving control commands from the client and carrying out the commands. All sensor communication was over an RS232 connection so the server had to setup the communication port for each sensor and begin communicating with it. To control the RS232 ports the RXTX 2.1.7 native library was used. This library provided Java with the ability to communicate with, obtain ownership of and setup the RS232 communication ports.

To make it simpler to add and remove sensors, each sensor had its own Java package. These packages retrieved the required RS232 communication port settings, used this to establish

communication with the sensors, received data from the sensors and formatted the sensor data. Each package then made the data available for when the client requested it. For setting up the RS232 ports each package contained a properties file. This file stored all RS232 port settings and any other information that was necessary for sensor communication. By using a properties file it made it simple to modify port settings if the sensor was attached to a different port or if the communication protocol changed.

Each sensor package was developed in Java except for the IMU package. A driver for the IMU developed by the Canadian Space Agency located in St-Hubert, Québec (Canada) was made available. This driver was developed in C so to make the driver usable in Java, the Java Native Interface (JNI) was used. JNI is a resource that is built into Java to facilitate using native libraries within Java. By converting the IMU driver into a native library it made it possible to incorporate it into the Java server, which allowed it to operate similar to the other sensor packages.

Each of the sensors that had Java packages developed for it operated over interrupt service routines. In this way the sensor's package sat idle until a new data packet was received. The data packet was read and stored in memory in its raw format. Storing data in its raw format reduced the time required to receive the data packets. This was advantageous because many of the sensor packages were receiving data at a higher rate than what it was being used by other parts of the program. Data that was not used by the rest of the program did not have to be converted so any conversion time would have been wasted. When other parts of the program required new information it would make a call to the sensor package that converted the data into a usable form and then returned it. Whenever new data was received the old data was removed from memory and the new data was stored in memory.

To provide tractor control, the server had both heading and speed controllers. Each of these controllers was developed into a control package. The way the control architecture worked was by having a thread that looped from the heading controller to the speed controller. The loop was carried out every 200ms and each time the loop repeated the front wheel angle and hydrostatic actuator positions were updated. The thread continued to loop until the test was completed. Once the test was completed the tractor was brought to a stop, all data files were closed and the thread was terminated. Figure 4.10 shows a flow diagram of the control loop.

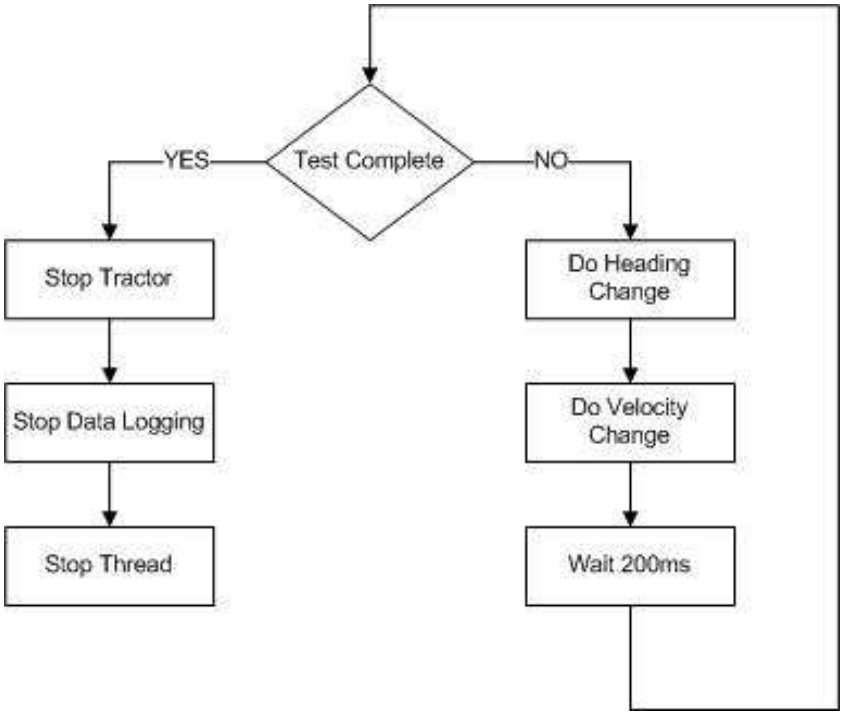

Figure 4.10: A flowchart of the control system

When performing the control loop shown in Figure 4.10 the heading and speed controllers were each developed into their own class. When the heading controller was initially started a properties file containing the heading PID constants as well as the steering properties was loaded. The required parameters were stored in memory and the file closed. From then on each subsequent call was to one function that determined the current and target quadrant and then

performed the PID calculations to find the target changes to be made. The newly calculated change was then sent to the motor controller package to update the motor controller settings.

The speed controller operated in a similar fashion. When the control loop thread was first started a speed controller object was created. Upon object creation the properties file was opened, all required information extracted from it to be stored and the file was then closed. Each call to update the speed controller was then done through one function call. The function determined the speed error based on the current and target speed. Using the PID constants the controller calculated the required change and determined what the new actuator setting should be. This setting was then passed to the motor controller package to update the motor controller.

When the heading and speed controllers update their settings they pass the updates to a motor controller package that was used for communicating settings with the motor controller. The motor controller package handled all communications with the motor controller. The motor controller communicated with the computer through an RS232 communication port. A class similar to the sensor packages was established for initiating communication with the motor controller as well as sending and receiving instructions between the motor controller and computer. This class read the port settings from a properties file, established communication and then sent and received commands as required. The information sent to the motor controller was the target hydrostatic lever position and target front wheel angle. The information received from the motor controller was the potentiometer voltages for determining the hydrostatic lever position and the front wheel angles. This information was used in the heading and speed controllers.

To incorporate the server into a RMI application it had to have an interface as well as an implementation. The interface was used to make methods available for remote invocation. The

implementation was used for implementing the methods that could be remotely invocated, as well as for implementing other necessary methods. The interface provided the ability to call methods for retrieving sensor information, starting sensor communications, remotely operating the tractor, performing autonomous tests, changing program settings and stopping the server.

4.5.1.2 External software

The external software portion of the application was primarily a graphical user interface (GUI) that allowed a user to interact with the server either remotely or from the same computer. The external software was developed for use on Debian Sarge kernel version 2.6.8 or Windows XP and was developed completely in Java using Netbeans 5.0. The client had seven separate windows that were each accessible by tabbed panes. Each window provided a function necessary for testing or tuning the tractor sensors and controllers. The external software windows are discussed in detail in Appendix B.

When the program started a main window was displayed that allowed the user to connect to the server. Before the user connected to the server it was not possible to perform any other tasks. When connecting to the server the user could either choose from two preset servers to connect to or connect to a new server. The preset servers were for the tractor and a previously developed ALV, while the new server option was used when installing the system on a new vehicle. Once connected, the main window began displaying the tractor latitude and longitude, X and Y coordinates, heading and speed. The user could also setup the data logger and begin logging data into user defined files.

A sensor window was available for displaying data. This window always showed the tractor's distance traveled in the East/West distance (X coordinate) and North/South distance(Y

coordinate). This distance was displayed in meters, with respect to the starting position of the test. The screen also showed the current tractor heading in degrees and the tractor's speed in meters per second (m/s). A drop-down selector was available for choosing which sensor to show data from and another drop-down menu allowed for changing the refresh rate. The only sensor that was not available for viewing was the SICK laser because there were too many data points to have any visual significance. Analog feedback from the front wheel angle potentiometer and hydrostatic potentiometer was also available on the sensor window.

It was possible to drive the tractor from a remote computer using the controller window. This window had two slider bars for adjusting the steering and hydrostatic commands or these parameters could be adjusted by using the keyboard arrow keys. A display of the current hydrostatic and steering commands was present below the slider bars. Another display showing a list of all commands sent to the motor controller was also present. When this window was open if the 'Enter' key was pressed the hydrostatic lever would immediately return to its neutral position. Similarly if the Shift key was pressed the tractor's front wheels would turn to their straight position.

The move window was used for tuning the heading and speed controllers. This window allowed for each controller to be tuned independently of the other. Three options were available on this window. The first option was a distance test which set the wheels to straight and made the vehicle travel in a straight line for a target distance with the hydrostatic lever set at a constant position. This test was used as the first autonomous test, but no further testing on this was performed. This test relied only on the GPS for its data and no heading or speed control was performed. The second option was to travel in a straight line while using the speed controller. This set the front wheels straight and attempted to reach and maintain the target speed. A

distance was entered and when this distance was reached the tractor stopped. This option was used for manually tuning the speed controller. The final option on this window was a heading controller test. This set the hydrostatic lever to a constant position and made the tractor turn to a target heading and follow that heading. This test had to be stopped manually when it completed. This option was used for manual tuning of the heading controller.

Also on the move window was the speed controller settings and heading controller settings. These values were read in from a properties file and this section could be used for changing the values in the file. The values were saved so that when the program was shutdown it maintained the correct PID parameters. These values were also the PID settings used in the autonomous control so when these controllers were tuned it automatically tuned the autonomous PID controllers.

The dynamic tests window was the window used for testing the system when it was all put together. This window used the previously tuned hydrostatic and speed controllers to control the tractor. Inputs for the controllers were read in from a file with each line containing a time, heading and speed. The heading and speed would be applied to the controllers while the time was used for determining the time duration of each instruction. Once this time was surpassed a new line containing a new time, heading and speed was retrieved from the file. When the end of the file was reached the tractor was brought to a stop and all data logging files were stopped. This window used the file input to mimic receiving instructions from a path planning software.

Screenshots of ALV Analyzer can be seen in Appendix B and the sourcecode for the software package is included on an attached CD. 0is a paper that goes into more detail of ALV Analyzer as it was being developed for use on the previous ALV. This paper is a non-peer reviewed paper that was developed for a class project.

By using this software structure the autonomous system was developed and tested. All data used in the analysis was collected using this software and all control of the tractor during testing was done through this software.

4.5.2 Middleware Software

Once the system was developed and tested using ALV Analyzer it was transformed into a more basic software package to make it possible for path planning and obstacle avoidance software to interact with it. The path planning software that was to be used with this system was developed in C++ so some type of communication protocol had to be developed. For this the client was removed and the server was put into a basic form. All sensor packages and the controller package were used, but the interface and implementation were redeveloped. The interface to the software was redeveloped to allow socket connections so that programs written in other software languages could still use the sensor and controller packages. The interface was modified to have reduced method calls that only allowed for receiving heading and speed as well as sending heading and speed commands. A method for stopping the server was also included. The implementation was similar to the ALV Analyze implementation except it had reduced capabilities as well.

There is no direct link to communicate between Java and C++. For this reason a C++ client was developed for providing methods for calling the Java methods in the server interface. For the server and client to communicate a communication protocol had to be developed because RMI is only used for applications with both the client and server in Java. For this a socket connection was used between the server and client. The server would open a port and wait for a connection from the C++ client. The client would attempt to contact the server and the two would create a connection. Once the two were connected sending and receiving of data could

begin. All data was sent in an ASCII format with predefined characters used for calling the methods. The socket connection was a handshake style in which all messages had to be acknowledged by the receiver to ensure that the client and server would not enter into a deadlock.

The C++ client was developed with only a few methods so anybody that developed software they wished to test on the tractor could use the system by creating method calls to these methods within the client. This approach was successfully used to perform some basic tests and maneuvers of a path planning software developed at the Université de Sherbrooke, Department of Electrical Engineering and Computer Engineering by Patrick Frenette, under the supervision of Dr. Franois Michaud.

CHAPTER 5: SYSTEM TESTING

5.1 Preliminary Testing

When developing the system, basic testing had to be performed to ensure the sensors and controllers were working close to what was expected. These tests were performed from June 25, 2006 until August 30, 2006 on the University of Saskatchewan campus in Saskatoon, Saskatchewan (Canada). The testing location was next to the engineering building and can be seen in Figure 5.1. The testing area was a level, hard packed area with short grass.

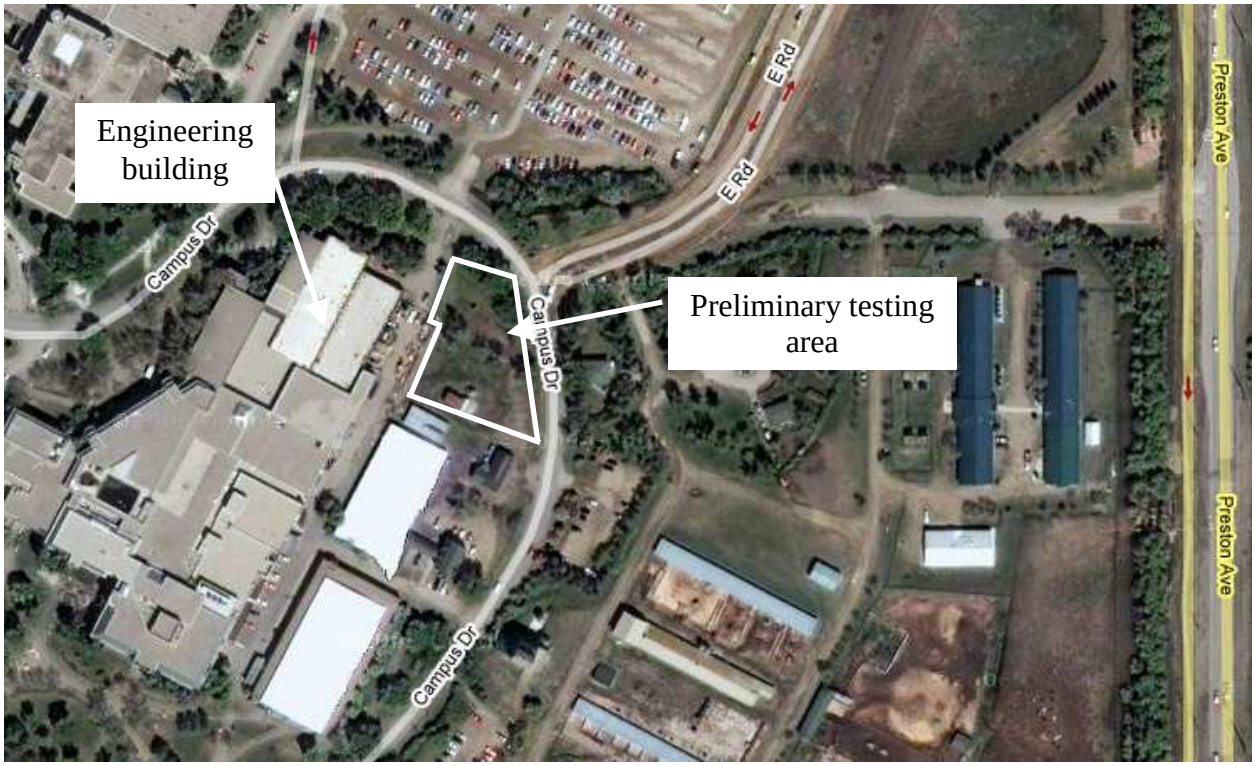

Figure 5.1: Location of the preliminary tests at the U of S (adapted from Google Earth©)

These tests were considered as part of the development process. Basic visual analysis of these tests was performed and based on the tests any required adjustments to the sensors and

controllers were made. These tests consisted of stationary tests and mobile tests carried out under Remote Control (RC) control, then remote computer control and finally some basic autonomous testing. During these tests the sensors were adjusted and mounts were modified to improve the system. These tests were also used for testing the controllers. The result of these tests was the completed system that was transported to a field where more space was available for more in depth system testing.

The first tests performed were carried out under RC control. This was useful to determine if the mechanical components added to the tractor were functioning correctly and if they needed any adjustment. It also proved useful in determining the restrictions of the tractor while under electronic control. During this testing a remote shutoff switch was developed. The switch was developed so that if a problem occurred, a switch on the RC panel could cause the steering to be deactivated and the engine to shutoff.

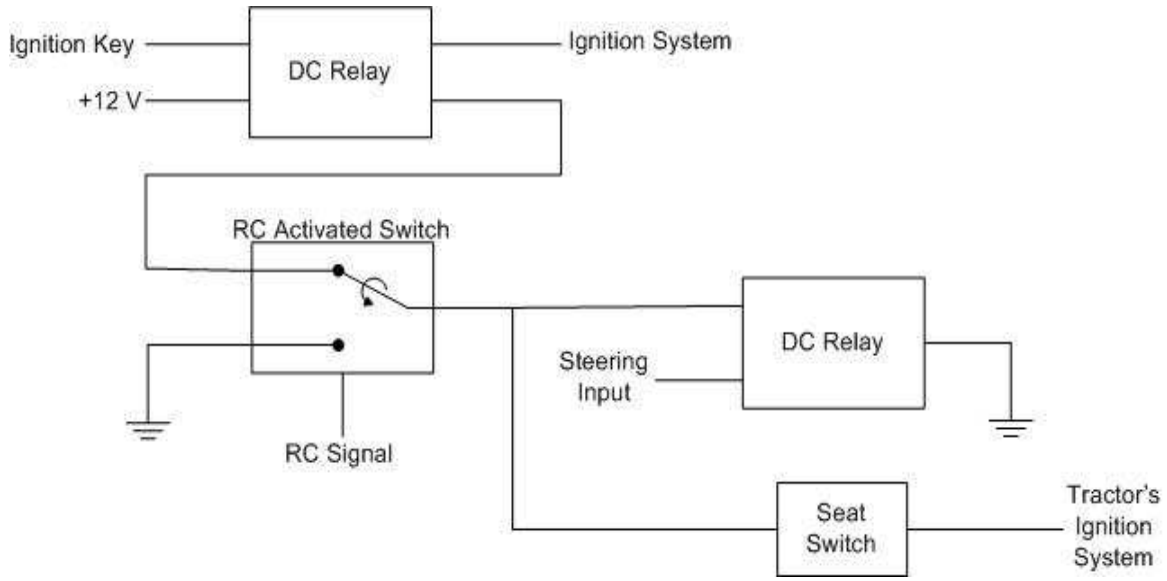

Figure 5.2: Circuit for the tractor's safety shutoff switch

Once the system was operating correctly under RC control it was tested with remote computer control. For these tests all instructions were passed to the tractor from the laptop that was

mounted on the tractor. The laptop received the commands via a wireless communication link with a remote laptop. The wireless communication was carried out over a wireless local area network (WLAN) using a Trendnet 54Mbps wireless-G router and compatible wireless network cards. These tests were useful for determining if the software was sending appropriate commands to the actuators and ensuring that the software was functioning correctly. During these tests the remote shutoff switch previously described was used, as well as a timer watchdog (see 0).

The timer watchdog was a timer that was implemented within the software. This timer noted each time an instruction was received and kept track of the time duration since the last instruction. If the time crossed a set threshold the timer signaled the program to stop the tractor and set its steering wheels to a straight position. This was required because it was possible for the wireless communication to lose connection which meant no instructions could be received. In this case the tractor was safely stopped before it could travel too far. The user always had the option of halting the tractor sooner by using the remote shutoff switch.

5.2 Field Testing

5.2.1 Test Field Description

The field where all final testing was completed was a Case New Holland (CNH) testing field located north of Saskatoon at the intersection of 71st Street and Millar Avenue. All tests performed were between September 29, 2006 and October 19, 2006. The field consisted of small rolling hills and the tractor was traveling on oat straw stubble. The test conditions varied from dry soil to rainy weather where the soil was slippery and muddy. One day of testing was performed on a day with a light drizzle and another day of testing was performed in a mild snow storm.

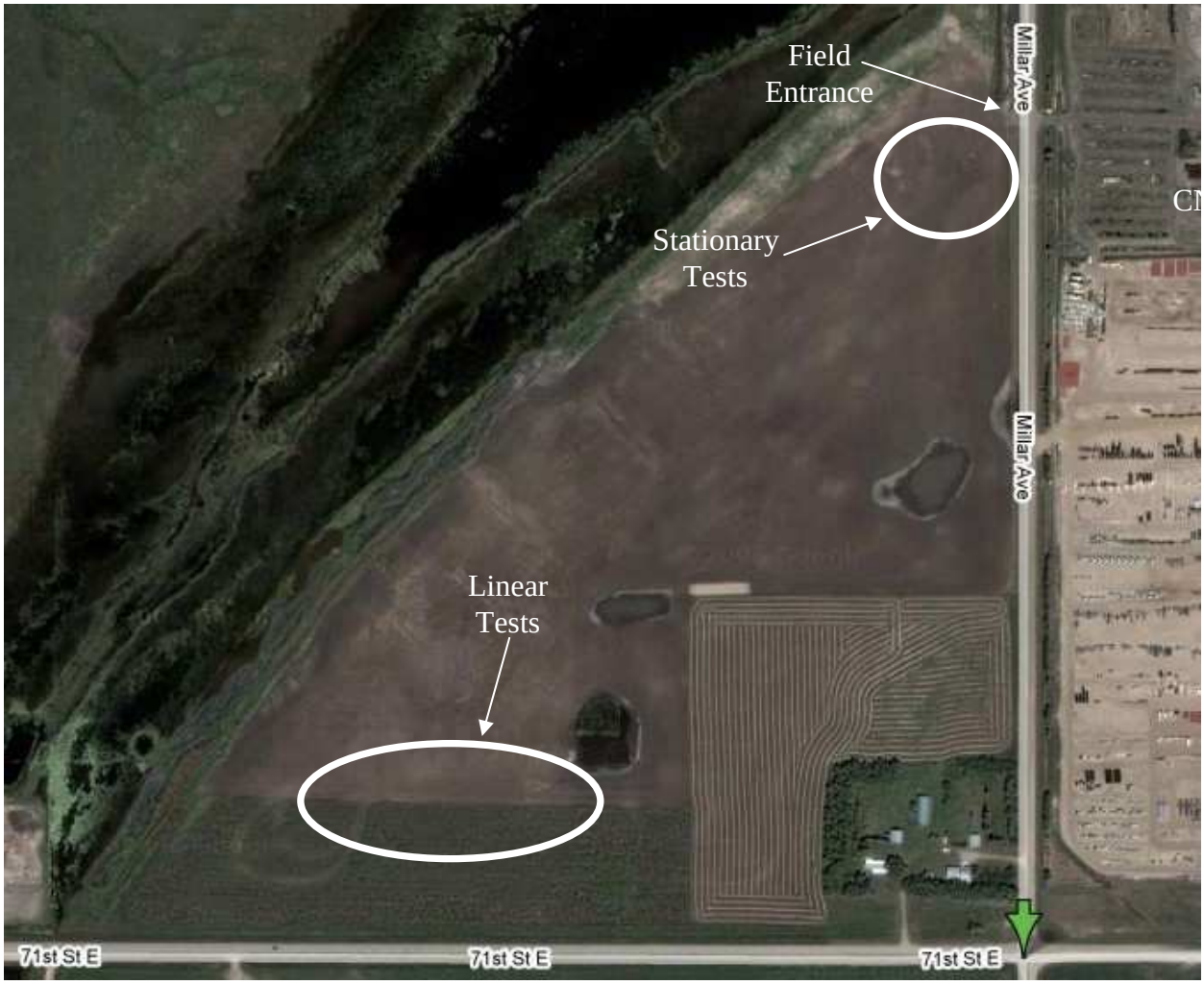

Figure 5.3: Location of the test field with the location of each test (adapted from Google Earth©)

5.2.2 Stationary Sensor Tests

Stationary tests were the first tests performed. These tests were used as a last check to ensure the sensors were operating correctly and to be sure nothing was damaged during transport. The test also provided a method to see if it would be possible to collect better sensor data when away from the city. When testing on the university campus there were many buildings and obstacles to interfere with sensor readings. This field was wide open and there were no obstacles that could interfere with the data.

When performing these tests the tractor was set at a 90º heading and the sensors ran for 30 seconds. This test was repeated five times and then the test was carried out five times at a 180º heading. The tractor remained stationary with the engine operating at 1500 RPM.

5.3 Human Driven Linear Tests

Human driven linear tests involved having the tractor driven along a straight line at different speeds while collecting data. The direction of travel was approximately 90º and 270º and the three speeds used were 0.5 m/s, 1 m/s and 1.5 m/s. These speeds were chosen because 1.5 m/s was close to the maximum safe operating speed and the other two were at equal intervals below this. The distance traveled was 100 m for the 0.5 m/s test and then 50 m for the 1 m/s and 1.5 m/s tests. The distance was shortened to 50 m after the 0.5 m/s tests were performed because it was found that it was difficult for the operator to accurately drive the tractor and perform other necessary tasks consistently for such a long distance. By shortening the distance there was less chance of the test becoming corrupted and having to be performed again. The most common cause of the test failing was losing connection over the wireless network connection. Five tests for each speed, in each direction were carried out.

For this test the tractor started from a standstill and accelerated until the target test speed was reached, it then traveled along a straight line. Stakes were driven into the ground to mark the required path. A stake was used to indicate when the timing of the test was to begin. At this stake the tractor would have to be at the target speed. The tractor would travel along the straight path until a stake marking 100 m was reached at which point the timing would stop. This stake was later moved to 50 m instead of 100 m. Once this stake was reached, the tractor was halted and turned around to repeat the test. The basic layout of the test is shown in Figure 5.4. Each of

these tests was timed with a chronometer and the time was recorded. This time was later used to provide an estimate of the tractor's average speed.

For this test the tractor was driven by a person walking behind or beside it. The driving was done from a laptop with a wireless connection. The person used the speed sensor as the speed reference and the GPS for a heading reference. The line marked out was measured and the direction was estimated from a hand held GPS. This meant the line direction was not exact and the exact heading was unknown.

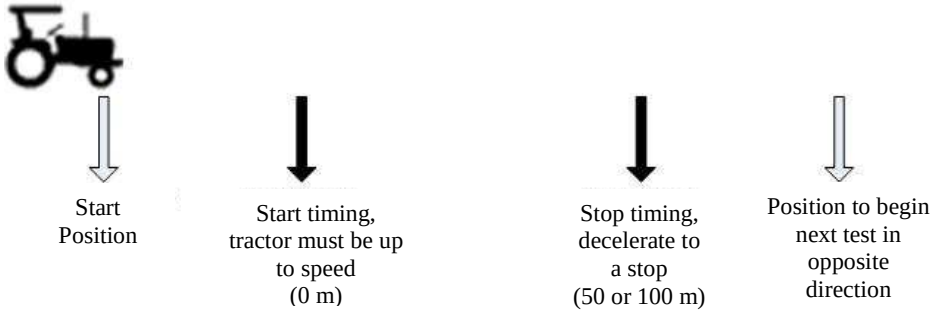

Figure 5.4: A visual display of the linear tests

This test was used for characterizing the sensor data. It was used to observe if the sensors were producing correct data, which sensors were producing the best data and how reliable the sensors were. This was useful during the analysis of tests where the computer was performing the control because it could then be determined if the errors were caused by the sensor data or the controllers themselves.

5.4 Computer Driven Linear Tests

The computer driven linear tests were performed in a similar manner to the human driven linear tests as shown in Figure 5.4. The tractor traveled the same line using the same stakes to mark the start and end points. The same headings and speeds were used, but for this test only 50

m distances were used. In the human driven linear tests a distance of 100 m was used for the 0.5 m/s tests, but to stay consistent with all other tests this was changed to 50 m. Again, five tests were carried out for each speed in each direction. Once again a chronometer was used to time the test and the times were recorded. These tests had a computer controlling the heading and speed. To control the heading and speed an input file was created that had the required heading, speed and estimated time it would take for the tractor to travel the distance.

The data from this test was used to determine if the controllers were operating correctly. When errors in the tractor control were found the linear tests were used to determine what was causing these errors and where the errors occurred.

5.5 Autonomous Tests

The autonomous tests had the tractor follow a preprogrammed course that included multiple headings and speeds (see Figure 5.5). The tractor operated with no human input. The input commands were received from a file and instructions were carried out sequentially based on a timer. The tests included seven instructions and the total test time was six minutes. To start each test the tractor was driven to a position that had four stakes marking the precise position where the tractor was supposed to start. This put the starting position in the same location for each test so that each test could be compared. The user then started the test and the tractor carried out the remainder of the test by itself. At the end of the test the tractor came to a stop and closed all data files. This test was carried out five times.

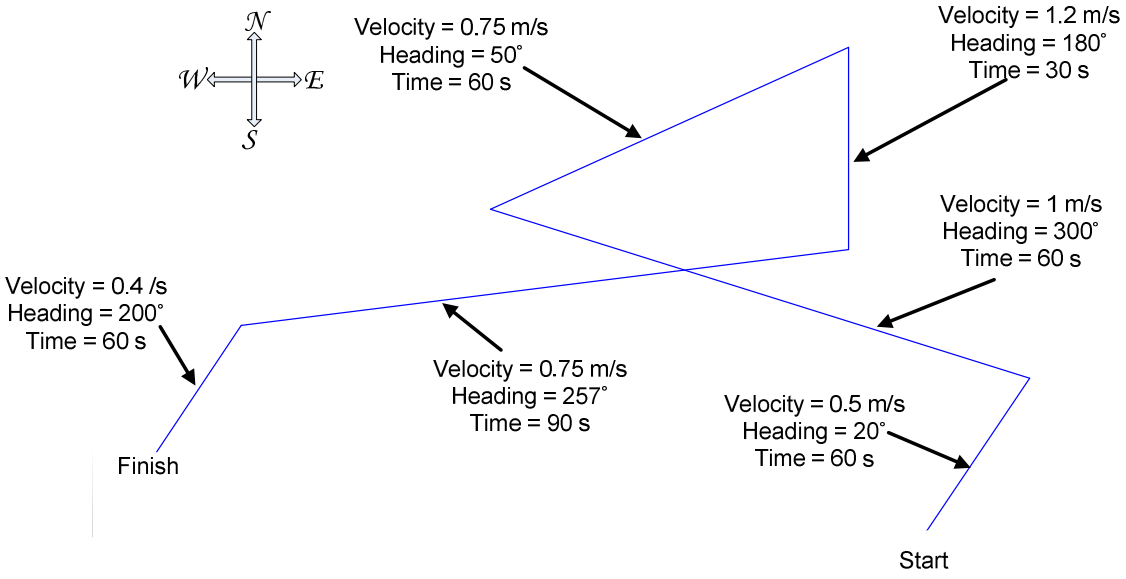

Figure 5.5: Planned path for the autonomous tests

During the tests all sensor data was logged as well as data from the heading and speed controllers. A function was also created that converted the GPS latitude and longitude into a 2-D projection to make it easier to perform positional analysis on the tests.

CHAPTER 6: RESULTS AND DISCUSSION

6.1 Preliminary Experiments

6.1.1 Remote Controlled Tests

The RC controlled tests were the first tests performed. These tests generally included taking the tractor out and driving it to see if any of the added components failed. During these tests the majority of the problems were with the steering system. Early in the testing, a set screw holding the steering shaft connecting the motor to the hydraulic pump came close enough to the motor mount that current could arc between the two pieces. When this happened the LoveJoy™ coupler in the steering shaft would melt from the heat and the steering fuse would blow. After performing some tests it was found that the steering motor was grounding through the metal chassis. To prevent the motor from grounding a plastic isolating pad was mounted between the motor and the motor mount. This prevented the motor from grounding and causing the fuse to blow.

After many hours of testing the steering fuse began blowing again. This was due to a failure in the LoveJoy™ steering shaft coupler. A LoveJoy™ coupler is a jaw-type coupler used for coupling two shafts. This specific coupler consisted of two sides, each having two jaws. The jaws from one side of the coupler mated with jaws on the other side of the coupler. Between the jaws was a rubber disc that acted to provide cushioning during shock forces and it allowed for slight misalignment of the shafts.

It was found that the rubber used in the LoveJoy™ connector was too soft so if the force required to turn the steering wheels was high, the rubber would contract to the point where the metal jaws could touch. When this happened the current again arced and caused the LoveJoy™

rubber to melt. To prevent this, a new steering shaft was made where the LoveJoy™ coupler was replaced with a solid Lexan™ piece that acted as the coupler. This prevented any metal components from coming in contact and so the steering motor could not incorrectly ground. After this there were no more problems with the steering motor grounding. The final steering motor mount and isolator is shown in Figure 6.1.

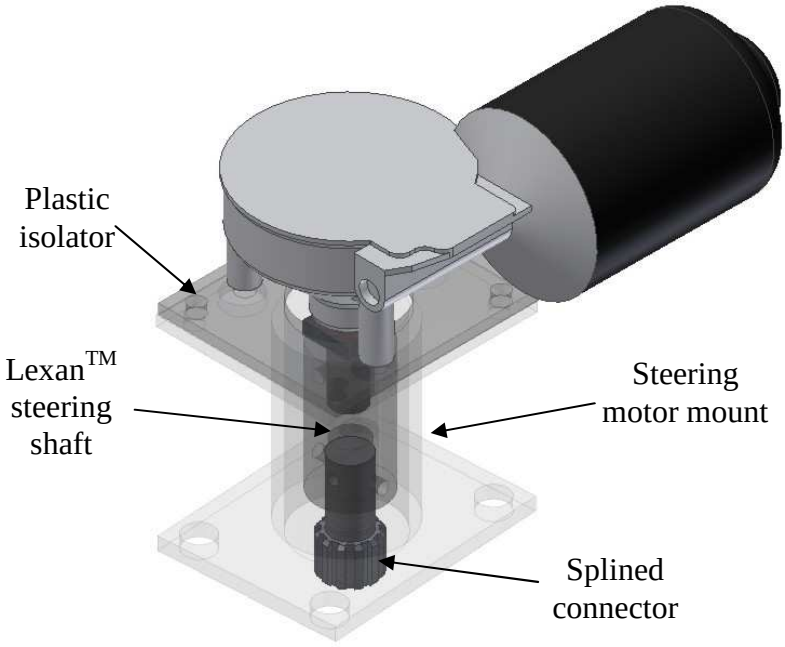

Figure 6.1: Final steering motor mount and isolator

Another problem was that when the front wheels turned all the way to one side they would reach the end of their rotation and stop. Even after the wheels reached the end of their turning abilities the steering motor would continue to try to turn them. This would cause the fuse to blow and it would not be possible to turn the wheels. To prevent this limit switches were installed on the front wheels to prevent the wheels from turning too far.

6.1.2 Remote Laptop Controlled Tests

Remote laptop was the second set of tests. During these tests it was fount that the wireless connection was poor. This was due to the large overhead with RMI, which is further described in 0. One workaround to this was to use realVNC to remotely log into the onboard computer and operate it from there. It was decided to continue to work with RMI.

During remote laptop testing it was found that some of the data files were not storing as target, so they were modified so that they were storing correctly. Data was logged and adjustments made until it was determined that data logging was working properly. It was during the remote laptop testing that the poor compass data was noticed. Once this was noticed the compass was remounted as described in Secion 4.3.2.

When testing with a remote laptop the onboard computer would sometimes freeze and stop operating. As time went on this problem became worse. It was finally decided that the problem was due to vibrations, which were causing the laptop hard drive to skip. To resolve this issue the computer case was mounted atop four dampeners. The dampeners removed enough vibration that the computer operated correctly.

The final problem was to have the software operating before the tractor engine was operating. If a setting was changed on the laptop that caused the motor controller to turn the front wheels, damage was often the result. In simple cases the steering fuse was blown, but there were also times when damage was done to the steering motor mounts. To prevent this from occurring, a relay was installed onto the tractor's ignition switch. When the ignition was turned on the steering was able to turn. When the ignition switch was in the OFF position, the relay prevented any current from going to the steering motor, which in turn prevented the tractor's steering from operating. The circuit for this is displayed in Figure 5.2.

6.1.3 Computer Controlled Tests

By the time computer controlled tests started many of the mechanical components had been thoroughly tested. The main adjustments made during these tests were to the controllers. The controller constants were adjusted to make the tractor perform as close to the laptop control as possible.

During this testing data was collected and reviewed. It was found that below 0.3 m/s the GPS did not return speeds as accurately as the drop wheel. This is because the GPS calculates its speed from position changes. The GPS has a low refresh rate (5 Hz) and is susceptible to positional errors. The combination of the low refresh rate and positional errors is the cause of the poor speed calculations at low speeds. It was decided that the speed sensor's data would be used at speeds below 0.3 m/s and after that the GPS would be used. For the heading controller it was decided that the compass data would be used at speeds below 0.7 m/s and after that the GPS data would be used. The compass was the most reliable heading sensor, because it did not rely on satellites or other outside data sources like the GPS. It also had a higher refresh rate. The problem with the compass at higher speeds was that the mount was constructed to put the compass above the tractor to avoid magnetic interference. Because the compass mount was long and high above the tractor, when the tractor traveled at higher speeds the rough terrain caused the compass to experience large amounts of movement. This movement introduced errors into the compass data. During this testing many errors in the heading controller were found due to miscalculating error in different cases. These were corrected, but there were still four times that error was miscalculated and this became more obvious during the field testing. This is described in more detail in Section 6.3.1.

6.2 Field Testing

6.2.1 Stationary Sensor Tests

The stationary tests showed that the compass had an average standard deviation of 0.21º when facing 90º (East) and 0.27º when facing 180º (South). These standard deviations support the idea that the compass mount was successfully reducing most vibrations that could deteriorate the compass' data. The average heading when facing 90˚ was 90.15˚ and the average heading when facing 180˚ was 179.82˚. When performing tests the directions of the tractor were observed and the distance between the two directions was very close to 90º. This shows that the compass was not receiving magnetic interference that could cause it to saturate in one area. To be sure that there was no interference with the compass, the tractor was driven in a slow circle. The compass measured a data point at each heading at a steady rate, showing again that the compass was not saturating in any one area.

When performing the stationary test the GPS was not capable of providing a heading to compare with the compass. It was however beneficial to observe the GPS positional data to determine the amount of drift that was occurring. For this the latitude and longitude were converted into the UTM coordinate system which acted as a 2-D projection of the latitude and longitude. At the start of the test the North and East positions were set to zero so that all data collected after the start time were relative to this point on the 2-D projection. Using this method it was possible to observe how far the GPS drifted in the North/South and East/West directions.

When facing 90˚ the GPS drift remained between 0 m and -0.75 m in the North/South direction and it was between 0.1 m and -0.85 m in the East\West direction (Figure 6.2). The North/South standard deviation was 0.17 m and the East/West standard deviation was 0.30 m. When facing 180˚ the North/South drift was between 0.8 m and -0.45 m with a standard deviation of 0.39 m. The drift was between 0.55 m and -0.1 m with a standard deviation of 0.18

m in the East/West direction. GPS receivers with Wide Area Augmentation System (WAAS) capabilities are required to be within 7.6 m or better 95% of the time, but often they are within 1 m (Crawford 2005). From the stationary tests it can be seen that the GPS stays within this region so the GPS must be correctly using WAAS to improve its data.

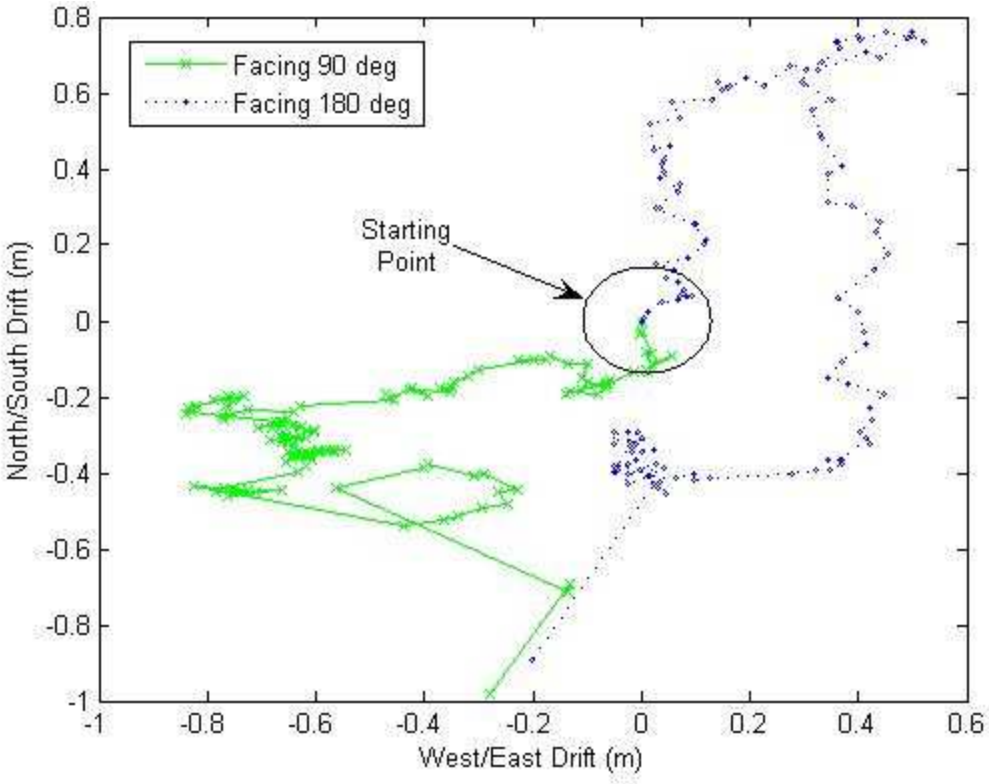

Figure 6.2: Average GPS drift during stationary testing at 90º and 180º

Figure D.1 shows the average heading measured by the compass for each of the stationary tests. Figure D.2 displays the measured compass data when the tractor turned in a circle.

6.2.2 Human Driven Linear Tests

The human driven linear tests were used to collect sensor data that could later be used for comparison with the computer driven linear tests. Data collected here was necessary for further characterizing the sensors and ensuring proper sensor operation. When performing the linear

tests human errors were obvious. It was difficult to keep the tractor traveling in a straight line and the speed was seldom held constant. The difficulty of maintaining a consistent tractor course increased as tractor speed increased. Maintaining the correct speed was difficult because the speed would constantly change as the tractor traveled over small hills. Maintaining the proper heading was even more challenging because it was common to either understeer or oversteer the tractor. As the speed increased it became more common to oversteer the tractor. Even with these errors, general heading and speed was close to the target path and the majority of the data errors could be explained from human errors.

Figure 6.3 presents the average speed for the GPS, speed sensor, theoretical speed and the target speed. Figure 6.4 also display the standard deviation for the GPS and speed sensor at each of the three test speeds. At each test speed, five tests in each direction were performed. Figure 6.3 shows the average of each of the five tests. Because there are two directions, there were actually ten tests performed at 0.5 m/s, ten at 1 m/s and ten at 1.5 m/s for a total of thirty linear tests. Figure 6.4 was produced from the same data, but it displays the average speed standard deviation of all tests taken at each test speed. The theoretical speed that is displayed in Figure 6.3 was found by using a stop watch to time the test duration. The test distance was 50 m so the test duration could be used to find an average speed that the tractor traveled during the test.

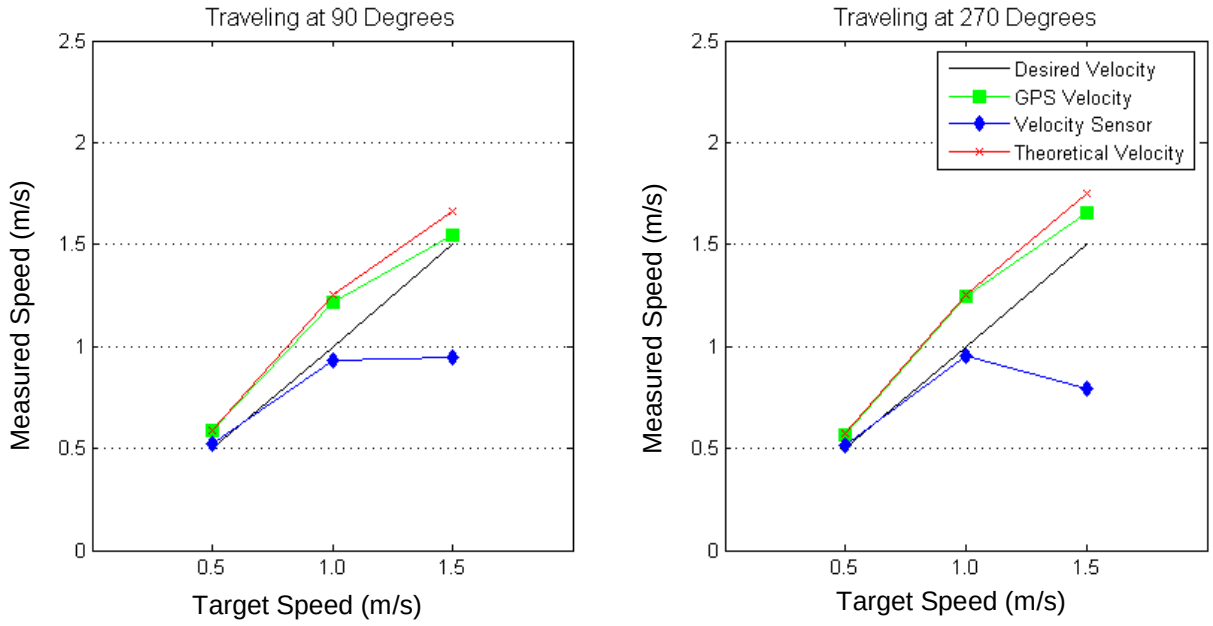

Figure 6.3: Average speed for the three test speeds while traveling at 90˚ and 270˚

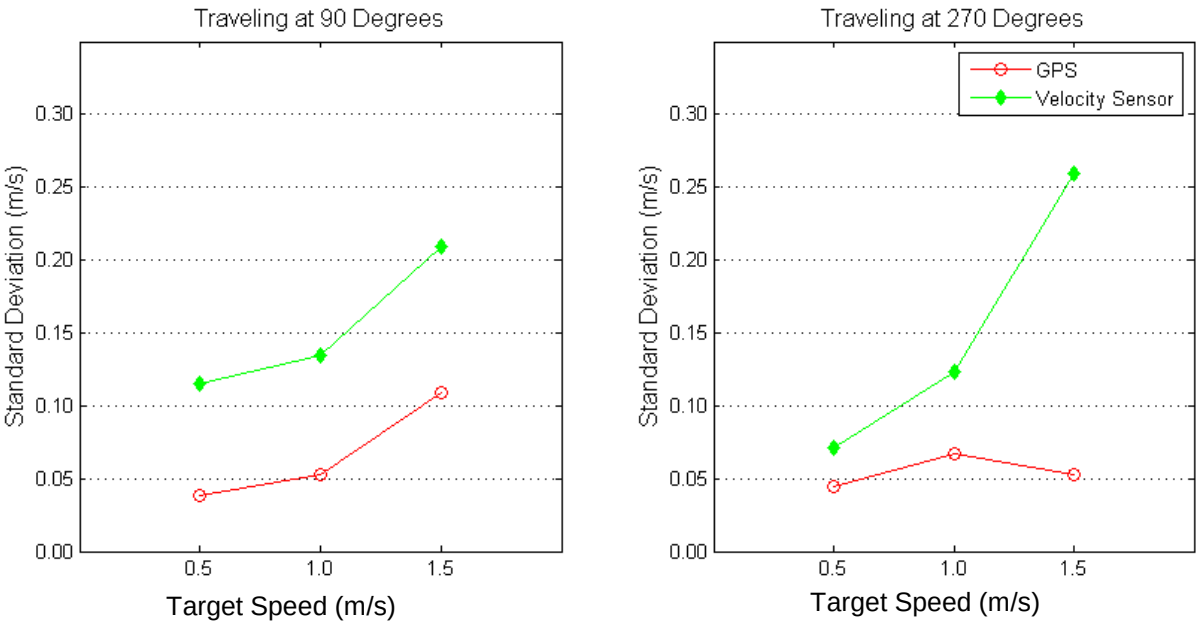

Figure 6.4: Average standard deviation for the GPS and speed sensor during human linear tests while traveling at 90˚ and 270˚

When performing these tests the speed sensor was used as feedback for the human operator. This was a problem, because as can be seen in Figure 6.3 the speed sensor did not work very well at speeds of 0.5 m/s and higher. When the speed increased the drop wheel began bouncing as it hit small bumps. This caused the sensor to quite frequently leave the ground which meant it could not register the proper speed. Every time the drop wheel left the ground the wheel would slow down slightly which would cause less pulses to be counted by the Hall Effect sensor. Also contributing to the speed sensor's poor performance at higher speeds was sensor damage as testing continued and the higher speed tests were performed later in the testing sequence. The sensor changes that caused poor data were that the Hall Effect sensor moved slightly causing the spacing between the Hall Effect sensor and gear tooth to be at the extremities of the allowable 2 mm spacing which meant it was more common to miss gear teeth during rotations. Another problem was that the sensor became encrusted in mud and debris which also helped to reduce the number of pulses registered by the gear tooth.

In Figure 6.3 the average difference between the speed sensor and GPS when performing the 0.5 m/s test was 0.07 m/s at 90˚ and 0.06 m/s at 270˚. The theoretical speed was the same as the GPS at 90˚ and it was 0.01 m/s higher than the GPS at 270˚. The reason the theoretical speed was higher than the 0.5 m/s target speed was that the speed sensor was being used for human feedback. This meant the human was adjusting the tractor speed based on what the speed sensor was displaying. When traveling at 90˚ the speed sensor was measuring 0.52 m/s and at 270˚ it measured 0.51 m/s. From this it can be seen that the human operator was very close to the target speed, but because the speed sensor was producing data that was slightly lower than the actual speed the tractor was actually traveling closer to 0.59 m/s and 0.58 m/s which can be seen by the theoretical speed.

When the speed was increased to 1 m/s, the difference between the speed sensor and other speed measurements increased even more. The GPS was still almost identical to the theoretical speed with only 0.01 m/s difference between the two for each test. At 90˚ the theoretical speed was higher than the GPS and at 270˚ the GPS had a higher speed than the theoretical speed. The difference between the speed sensor and theoretical speed was approximately 0.3 m/s at both headings.

When the speed was increased to 1.5 m/s the speed sensor's performance became even worse than at the other two speeds. This is shown in Figure 6.3, where the speed sensor's average speed stayed the same from the 1 m/s test to the 1.5 m/s test. For the 270˚ heading the average speed reported by the tractor actually decreased from 0.95 m/s during the 1 m/s test to reporting 0.79 m/s for the 1.5 m/s test. The speed sensor actually measured a lower speed at 1.5 m/s during the 270˚ heading than it did at 1 m/s. For the 90˚ heading the speed sensor produced a speed estimate that was almost identical to the 1 m/s test. At 1.5 m/s the GPS speed and theoretical speed differed slightly. The GPS speed was 0.11 m/s lower than the theoretical speed at 90˚ and 0.09 m/s lower at the 270˚ heading. It was not expected that the average GPS speed would be lower than the theoretical speed at 1.5 m/s. The tractor was traveling at a higher speed which made it more difficult to properly time the test with a stop watch; so it is possible that the difference between the GPS and theoretical speed was caused by human error.

From Figure 6.4 it can be seen that the average standard deviation for each sensor during each trial increased as the speed increased. The exception to this was at a 270˚ heading because the GPS standard deviation at 1.5 m/s was lower than at 1 m/s. The standard deviation of the GPS was quite low, remaining below 0.1 m/s, other than during the 90˚ heading test at 1.5 m/s where the standard deviation became 0.11 m/s. The speed sensor had a standard deviation that was also

low (0.11 m/s and less), but this is deceiving because the speed sensor was not operating correctly. The speed sensor measured low values so it was expected that the standard deviation would be lower.

The average heading for all five tests at each of the test speeds is shown in Figure 6.5. This plot is similar to Figure 6.3 and Figure 6.4 in that it is an average of all tests. In Figure 6.5 the compass data has error bars at each of the test speeds. These error bars were determined by using the average standard deviation from the stationary compass data. The error bars are 0.48˚ in height because the standard deviation of the compass heading during the stationary tests was 0.24˚. Figure 6.6 is the average standard deviation of the heading data for the GPS and compass during the linear tests.

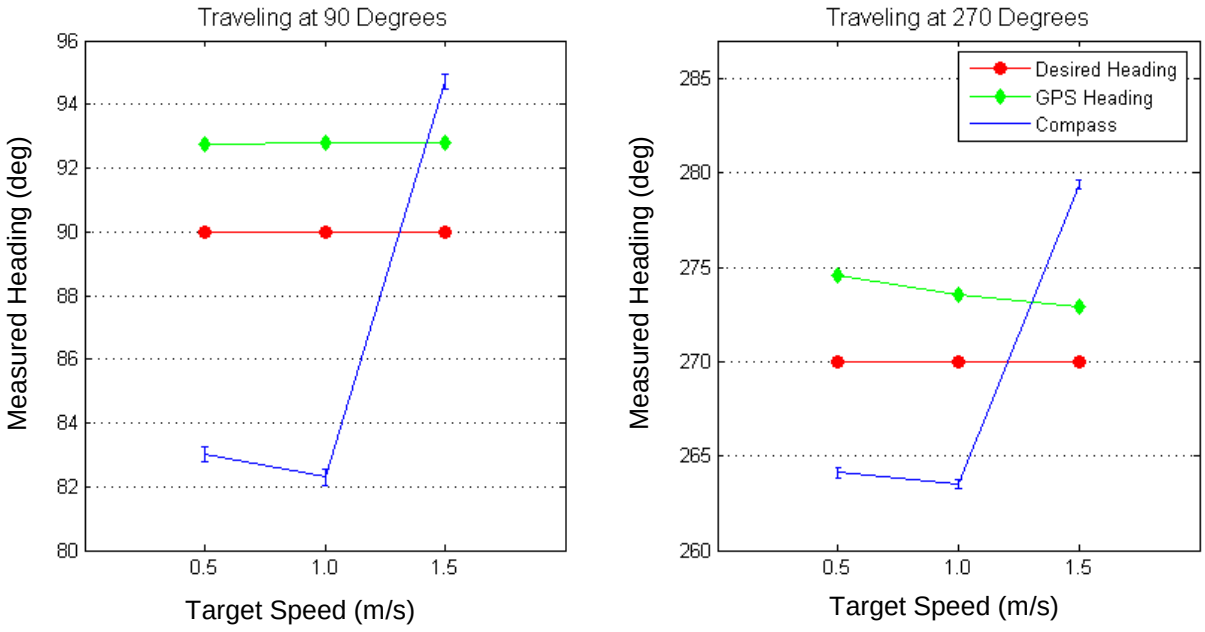

Figure 6.5: Average heading for all tests at each of the three test speeds while traveling at 90˚ and 270˚

From the data shown in Figure 6.5 it can be noted that the GPS heading was always higher than the target heading, while the compass heading was generally lower than the target heading.

The reason neither of the sensor headings were the same as the target heading was that the line used to drive the tractor along was marked out using a hand held GPS and compass to estimate the direction. Neither of these hand held sensors were very accurate sensors so the exact direction of the line was not known. The reason the GPS and compass are not exactly the same was that the compass was misaligned on the tractor by approximately 10˚.

When observing the GPS data it seemed to follow the target heading except that it had an offset. At 90˚ the target heading and GPS heading are almost parallel to each other, while at 270˚ the GPS heading moves slightly closer to the target heading with each test. The average GPS offset from the target heading at 90˚ was 2.78˚ higher and at 270˚ the offset was an average of 3.65˚ higher. The change from the 0.5 m/s test to the 1.5 m/s test was only 0.05˚, whereas at the 270˚ heading the change was 1.67˚. Neither of these variations in heading was very significant considering the vehicle was being driven by a human.

The compass did not produce data that was as constant as the GPS. From 0.5 m/s to 1 m/s the compass heading slightly moved away from the target heading, increasing the offset. For the 1.5 m/s test the compass heading became larger than the GPS data, which was unexpected. This can be explained by the standard deviation which is plotted in Figure 6.6. The compass standard deviation grew quite rapidly as the speed increased. For the 270˚ tests the growth of standard deviation was almost a linear function of the target speed. For the 90˚ heading it was also somewhat linear, but not to the same extent as the 270˚ heading. The increase in standard deviation would have caused the compass heading to be less precise.

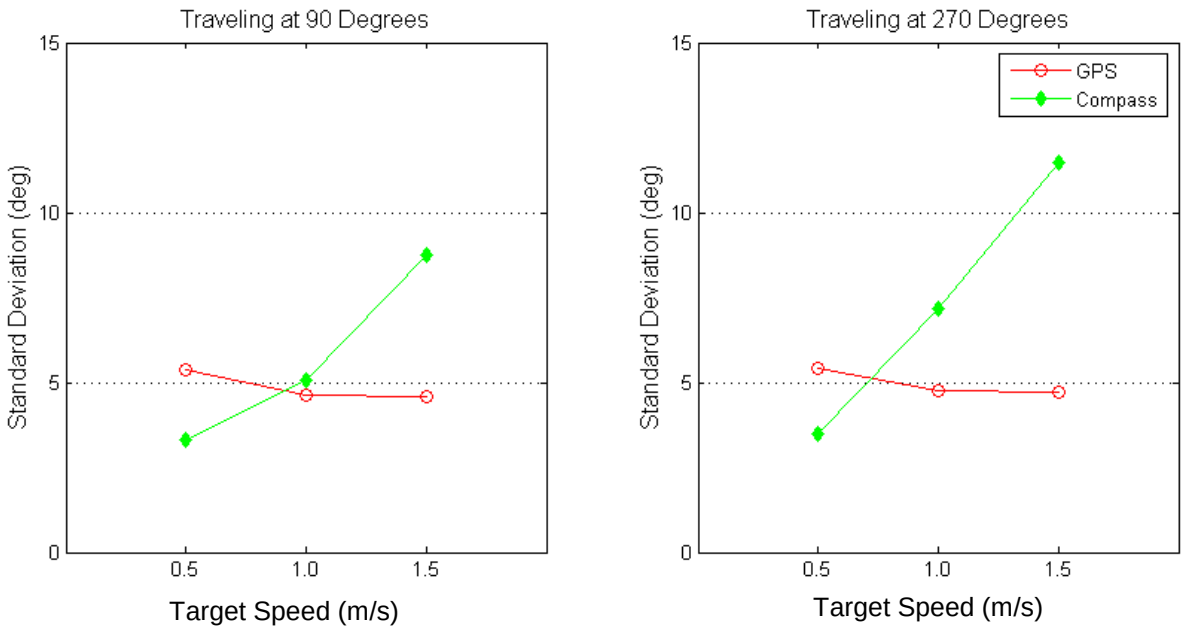

Figure 6.6: Average standard deviation of all tests heading for the GPS and compass at each test speed while traveling at 90˚ and 270˚

The reason the compass standard deviation increased with speed was compass movements that were apparent at higher speed. The compass was mounted atop a plastic pole that was fixed to a weight. The weight was mounted on top of four rubber isolators to provide dampening, which reduced vibrations. The weight was steady at lower speeds, but as the tractor speed increased block movements also increased. Any movement in the weight was magnified at the compass because of the long pole it was on. When the tractor was traveling at high speeds the compass had a very large amount of movement in all directions, which increased the standard deviation.

When observing the GPS and compass data it was observed that the standard deviation of each sensor was quite similar at 0.5 m/s with the compass having a slightly lower standard deviation. At 0.7 m/s the two sensors had approximately the same standard deviation and from this test on, as speeds were increased the GPS produced data with a lower standard deviation.

This can be seen in Figure 6.6 and this was the reason the heading controller used the compass data at speeds lower than 0.7 m/s and after the speed was higher than 0.7 m/s the GPS data was used.

It should be noted that when performing the 1.5 m/s test some of the data was corrupted due to computer errors. For the 270˚ heading there were only two valid data sets for the compass and three data sets for the GPS. At the 90˚ heading the compass had three data sets and the GPS had four data sets. The speed sensor and GPS only had four sets of speed data for both headings at this speed. The absence of data most likely had an effect on the average data and standard deviation. Appendix D.2 displays figures that shows the average speed and heading for each of the five tests performed at each velocity. A table summarizing the data is also presented.

6.2.3 Computer Driven Linear Tests

The computer driven linear tests were for testing and observing the performance of the controllers. The tests were performed in a similar fashion to the human driven linear tests so the results could be compared. When observing the data from these tests it is obvious that errors in the controllers were causing the tractor to not perform as well as expected.

When observing speed data colleted during each of the tests, the results were similar to the human driven linear tests. From Figure 6.7 it can be seen that the theoretical speed and GPS speed match each other almost perfectly. The points overlie each other in the plot. The GPS and theoretical speed are only slightly above the target speed and the difference between these speeds increases as the test speed increases. This can be explained by the speed sensor data. As can be seen the speed sensor produced a very low estimate. The speed sensor estimate became worse when the target speed was increased from 1 m/s to 1.5 m/s. This was due to the speed sensor errors discussed in Section 6.2.2.

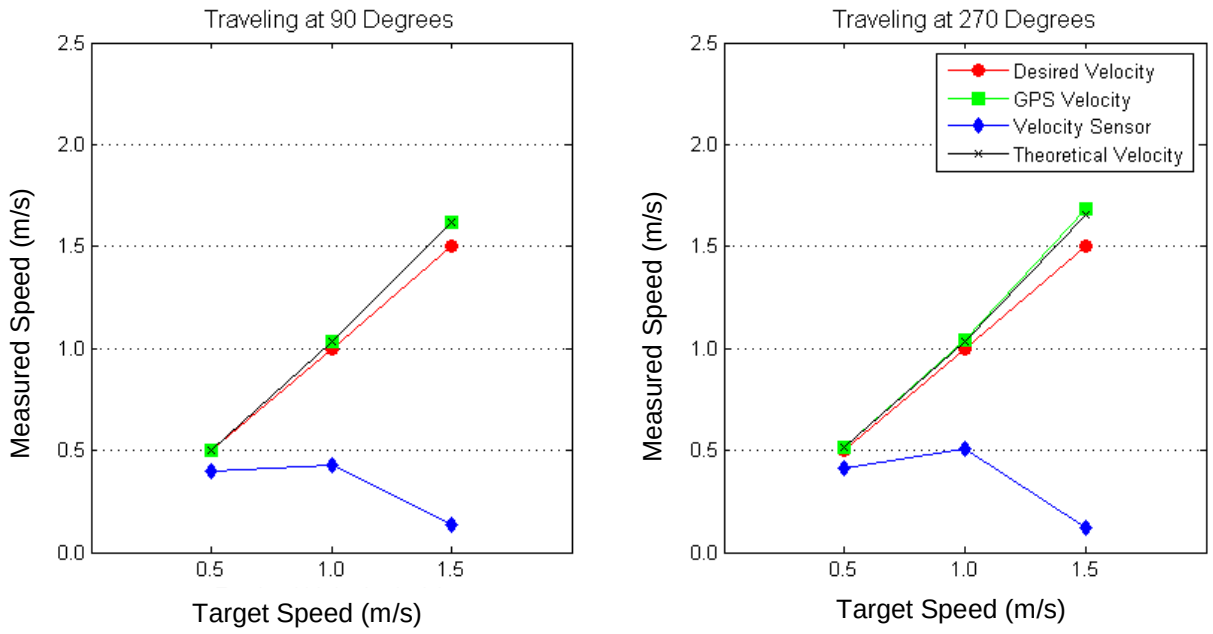

Figure 6.7: Average speed of each sensor for all three test speeds while traveling at 90˚ and 270˚

The way the speed controller operated was that it utilized the speed sensor data at lower speeds, and once the speed sensor measured a speed above 0.3 m/s the speed controller began using the GPS speed for its feedback. As can be seen in Figure 6.7 the speed sensor did not report speeds higher than 0.5 m/s, even when the tractor was traveling at a speed that was much higher than 0.5 m/s. For this reason the speed controller was receiving feedback reporting that the tractor was traveling too slowly. This caused the speed controller to continually try to increase the speed of the tractor. Whenever the speed sensor measured more than 0.3 m/s the GPS was used and the speed controller would reduce the tractor speed. As the tractor speed increased the speed sensor returned a value below 0.3 m/s more frequently so at higher speeds the speed controller used the speed sensor, even though its data was poorer than the GPS. This is why the theoretical and GPS speed was more offset from the target speed at 1.5 m/s.

The standard deviation for the speed sensor and GPS speed was very similar with the maximum difference between them being 0.04 m/s at a 90º heading and 0.018 m/s at a 270º

heading. This is displayed in Figure 6.8. For both sensors the standard deviation varied less than 0.05 m/s between test speeds. The standard deviation for the speed sensor was not representative because the sensor was not operating correctly. Because the speed sensor was not operating correctly it always remained at speeds of 0.5 m/s or less. This speed should have been much higher and if the speed was higher, the standard deviation of the speed sensor most likely would have increased.

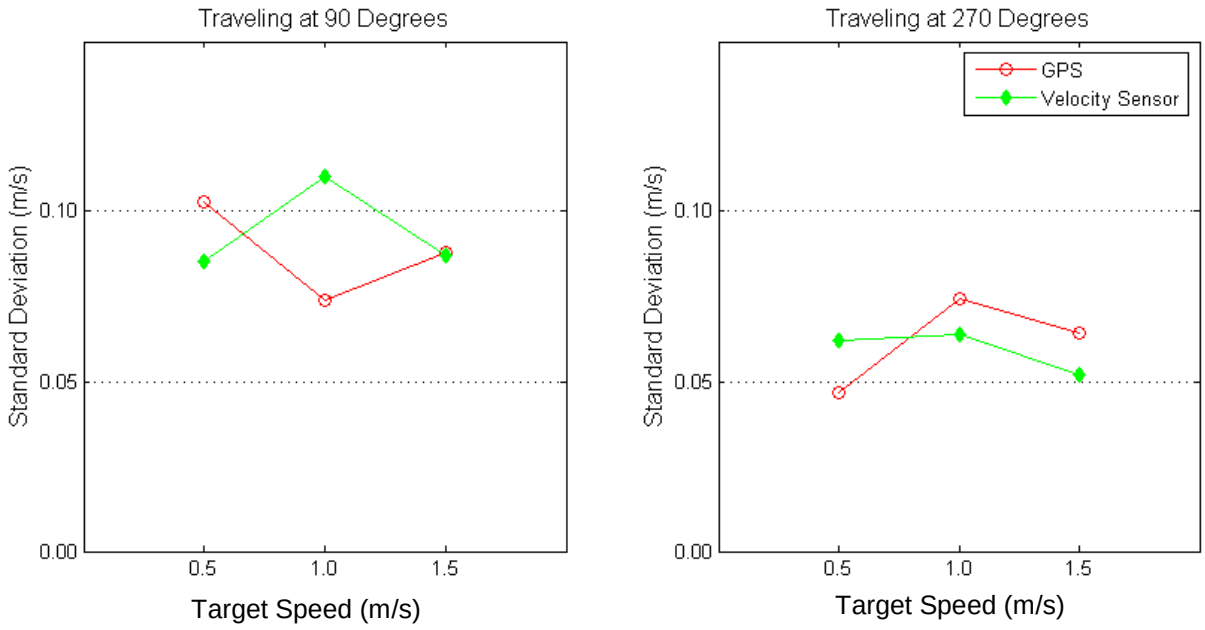

Figure 6.8: Average standard deviation of the GPS and speed sensor at each test speed while traveling at 90˚ and 270˚

The heading controller produced errors during the linear test. Figure 6.9 is a plot of the average heading for the compass and GPS at each of the test speeds. It can be noted that as the speed increased the tractor's heading offset from the target speed increased. This was due to an error in the heading controller. The heading controller was calculating the error correction incorrectly at four different locations and two of these locations were on the 90º and 270º heading. Whenever the tractor crossed either of these headings the error was calculated to be

very large, which caused the corrective action to be extremely drastic. Because an aggressive action was taken, the tractor veered off course very quickly. Once the tractor was out of this region the controller would begin to slowly correct itself.

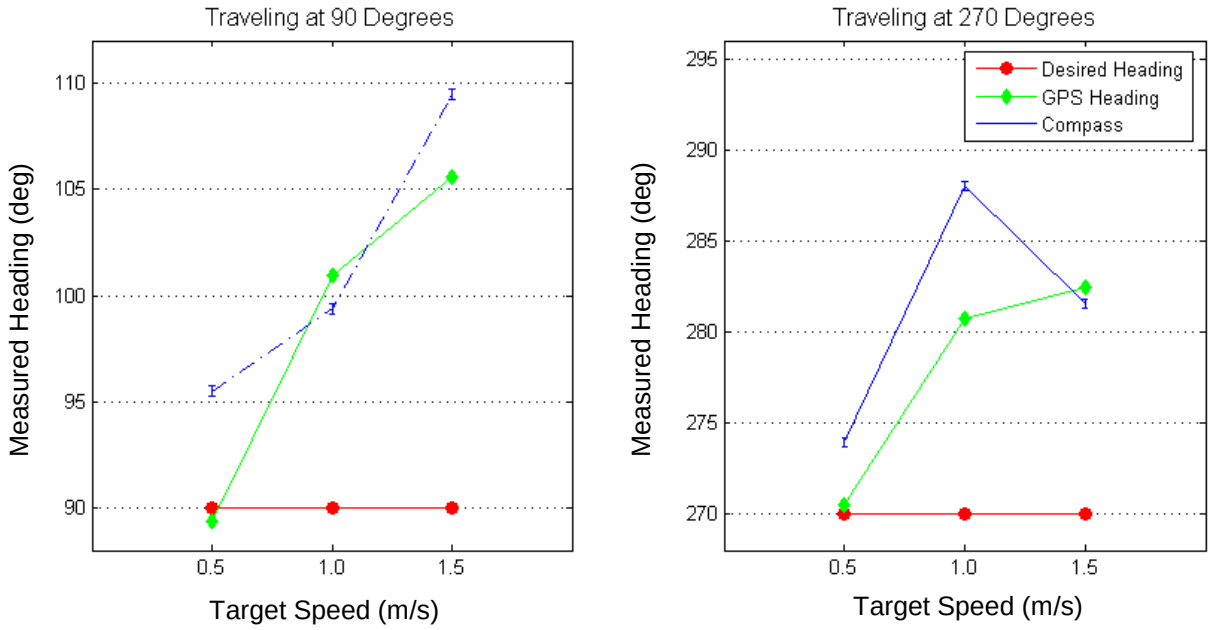

Figure 6.9: Average heading of each test for all three test speeds while traveling at 90˚ and 270˚

The reason this error became more noticeable at higher speeds was that the aggressive turn was amplified due to the increased speed. The tractor turned the wheels all the way to one side and because the tractor was traveling at a higher speed it traveled more distance before the controller could begin to correct itself. This error will be discussed in more detail in the Section 6.3.1.

From Figure 6.9 it should be noted that the compass and GPS produced heading estimates that varied by approximately 5˚. In the human driven linear tests the compass heading was lower than the GPS heading, except during the 1.5 m/s test, where the compass became inaccurate. The reason the compass produced a higher heading estimate than the GPS is that the errors found

during the human driven tests were observed and the compass was realigned. The realignment was not successful and the compass was still misaligned. The misalignment was reduced from approximately 10º to 6º or less.

During the tests the heading standard deviation was similar to the standard deviations during the human driven linear tests. Figure 6.10 shows the average standard deviation for each test at each test speed. As can be seen the GPS standard deviation was quite constant during the 270º heading, but for the 90º heading there was a large improvement in the standard deviation when going from 0.5 m/s to 1 m/s. The difference between the 1 m/s and 1.5 m/s tests was insignificant for the GPS standard deviation. As was expected the compass standard deviation increased from approximately 5˚ during the 0.5 m/s tests to 23˚ during the 1.5 m/s tests. This was once again caused by compass movement due to the tractor having more rapid roll and pitch movements at higher speeds. Appendix D.3 displays figures that shows the average speed and heading for each of the five tests performed at each velocity. A table summarizing the data is also presented.

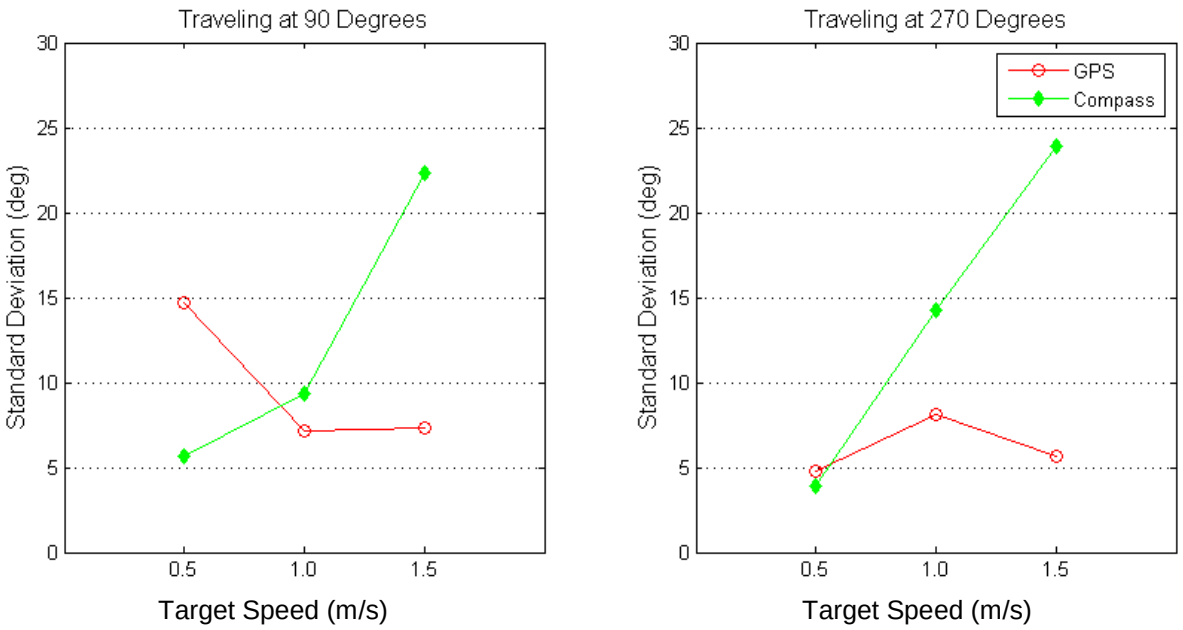

Figure 6.10: Average heading standard deviation for all tests at each of the test speeds while traveling at 90˚ and 270˚

6.2.4 Autonomous Tests

When performing the autonomous tests an input file contained the instructions shown in Table 6-1. Figure 6.11 shows the path followed by the tractor when given these instructions. It also shows the calculated path that would have been followed if the tractor control was perfect. The calculated path was found by using the input file commands to calculate the theoretical distance traveled and heading during each instruction. The calculated path does not account for turning or accelerating times so it was not possible to follow this path perfectly. The tractor path was found by using the position data in the UTM format. From Figure 6.11 it appears as though the actual path deviates from the target path as the test gets closer to completion. This is because as the test continues any heading and speed errors sum up and form the positional error. Also the tractor control was not based on position, whereas this figure shows the tractor position. As the

tractor position deviated from the calculated position there was no correction to try to make the tractor return to the path. This is why the deviation increased as the test continued.

Table 6-1: Input commands used for the autonomous tests

Time (s)	Heading (degrees)	Speed (m/s)
30	0	0.00
60	20	0.50
60	300	1.00
60	50	0.75
30	180	1.20
60	257	0.75
60	200	0.40

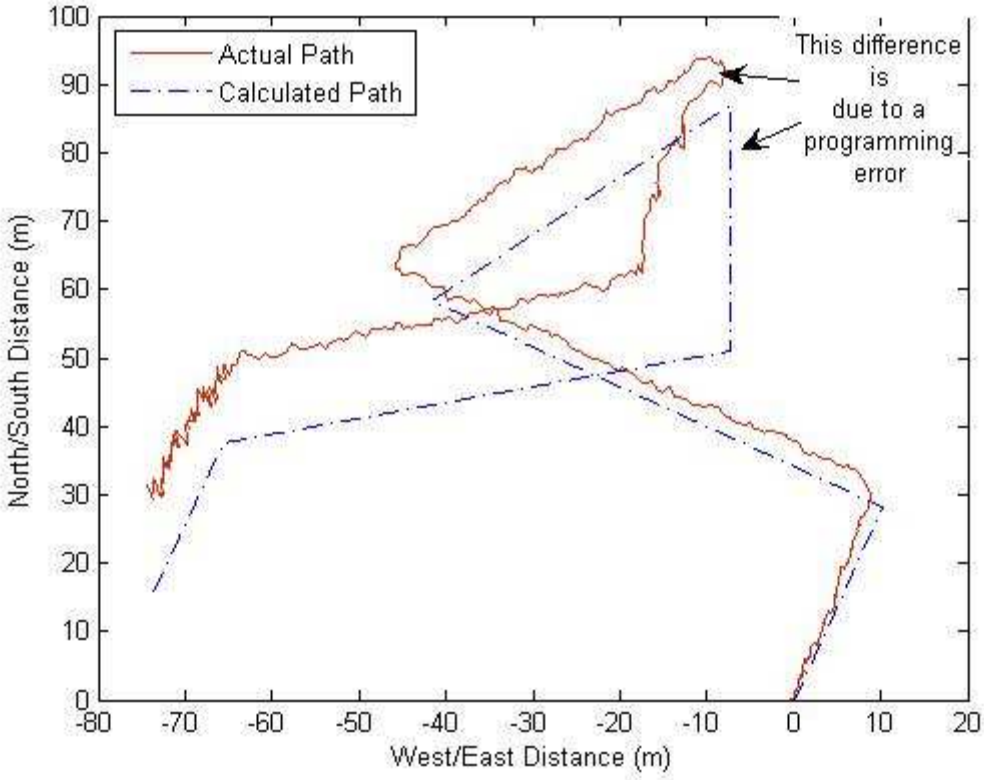

Figure 6.11: Path followed during the autonomous tests as well as the expected path

At the very top of Figure 6.11 the target path and actual path are quite different because of a programming error in the heading controller. The error was computed incorrectly by the heading

controller at this location, which made the tractor turn too aggressively. This put the heading controller into an unstable situation in which the tractor rotated from side to side causing the speed controller to also become unstable. In Figure 6.13 the points where the controllers became unstable is labeled. Both the heading and speed controllers remained in an unstable state until a new command was received. Once a new command was received the tractor resumed as it was supposed to.

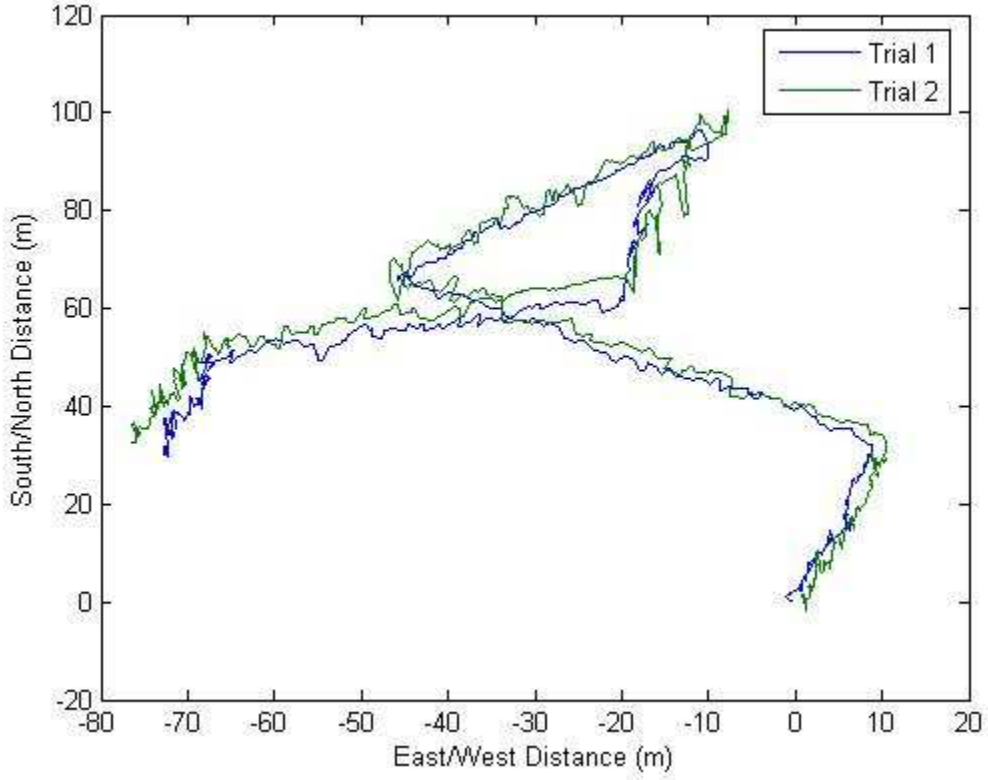

Figure 6.12: Path followed by the tractor during two separate tests

The offset caused by the unstable controllers caused the actual path and calculated path to deviate even further than what they should have. Even if there had not been an error the calculated path and actual path would have been different. This difference can be equated to the controllers not physically being able to respond fast enough to reach the perfect situation as displayed in the calculated path. It is not possible for the controller to have a perfect response

because it was constrained by physical limitations, but the calculated path did not take into account any of these limitations. Whenever a command was received there was some lag time between receiving the command and the command actually being implemented. Once the command was implemented there was more lag time to when the tractor actually reached the target setting. The controllers could be tuned to better reduce this lag time, but it could never fully be removed.

Figure 6.12 is a positional plot of the raw data showing the path followed by the tractor during two tests. From this plot it can be seen that the two tests followed a path that was very similar. The other three tests, which are not shown in this plot, also followed a similar path. This demonstrates controller repeatability, which means that if the controllers were better tuned and were receiving better sensor data then the tractor should be capable of following the target path more closely.

The difference between the maximum final East/West direction and the minimum was 8.73 m while the North/South difference was 4.81 m. This suggests that the further the tractor traveled in a direction the further off its position estimation became. The controllers were not based on position control so if any heading or speed errors occurred the controllers would not try to correct for these errors. Based on the other tests, if position control had been used these final offsets could have been reduced.

When summarizing the final positions the average West/East position was -74.04 m with a standard deviation of 3.25 m, while the North/South position was 31.52 m with a standard deviation of 1.8 m. This test was approximately 5 minutes and 30 seconds in duration so to accumulate errors of this magnitude would be unacceptable for most agricultural uses. On the other hand, errors of this proportion were achieved even though there were mistakes within the

heading and error controllers so by fixing the controllers, acceptable results should be achievable.

Figure 6.13 shows the heading and speed data collected during the first test. The positional data for this test is shown in Figure 6.12. From Figure 6.13 it can be noted that the heading performed quite well, except for where the controller went unstable. The heading controller responded quickly and had a very short settling time. There was some oscillation about the target heading when steady state was reached, but not very much. Once the tractor reached the target heading there was no noticeable heading offset.

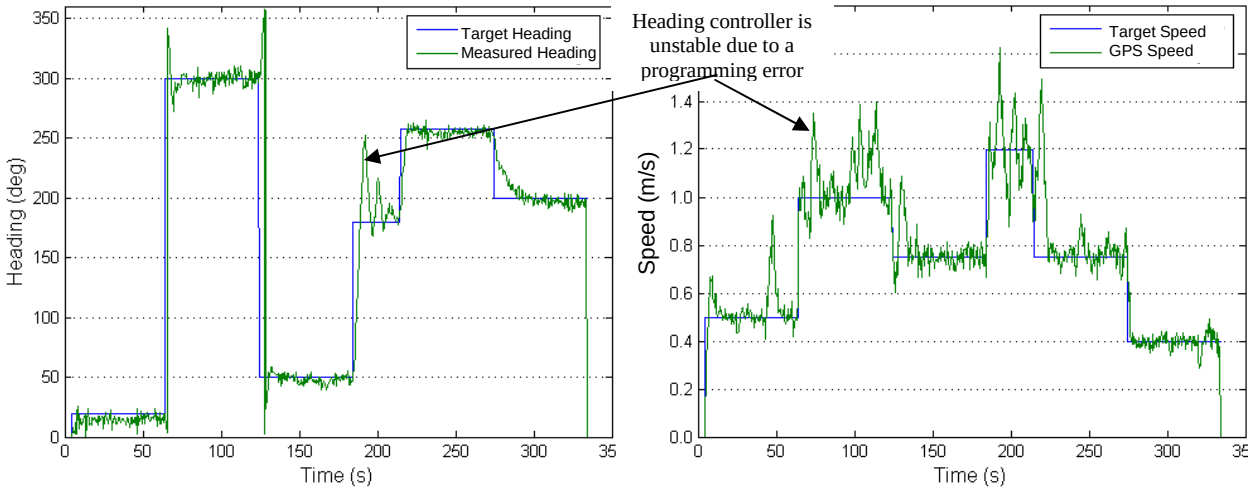

Figure 6.13: Raw speed and heading data from Trial 1 of the autonomous test

When comparing the standard deviations in Table 6-2 to Figure 6.13 the data seems to contradict each other at 300º and 50º headings. The reason the standard deviations are so high in Table 6-2 for these headings was due to the oscillations that occurred as the tractor turned. The tractor had to turn a lot to reach these headings which meant that there was a higher overshoot. This overshoot was quickly accounted for and the heading settled out to the target heading. This overshoot did have a large affect on the standard deviation though. When the tractor did not have to turn as much a less aggressive approach was taken and there was less overshoot, which

resulted in a lower standard deviation. Another reason for the high standard deviations in Table 6-2 was that the standard deviation was taken from the time the new command was received to the time that the next command was sent. This meant the standard deviation started being calculated before the tractor even started to adjust and included the data during the entire adjustment to reach the new instruction.

Table 6-2: Analysis of the data from Trial 1 of the autonomous tests

Time (s)	Target Speed (m/s)	Average Speed (m/s)	Std Dev (m/s)	Target Heading (deg)	Average Heading (deg)	Std Dev (deg)
0	0.00	0.00	0.00	0	0.00	0.00
5	0.50	0.50	0.04	20	14.62	4.31
65	1.00	1.04	0.12	300	291.74	46.00
125	0.75	0.76	0.05	50	63.84	65.70
185	1.20	1.19	0.20	180	183.00	44.30
215	0.75	0.79	0.09	257	252.05	13.01
275	0.40	0.41	0.05	200	205.41	13.11

The speed controller also performed effectively, but it often oscillated about the target speed. Table 6-2 has the average speeds compared to the target speed for each time instance and from this it can be seen that the average speeds are quite close to the target speed. Also the standard deviations are quite low, except for the 1 m/s and 1.2 m/s. This coincides with the findings of the linear computer driven tests in that the higher the target speed, the poorer the speed control becomes, due to the speed sensor errors. The average speed at 1 m/s and 1.2 m/s was still quite close to the target speed, but the standard deviation was high because of spikes in the speed data. At 1.2 m/s the speed controller was being affected by the heading controller's instability which was another reason for the high oscillations and high standard deviation.

6.3 Controller Analysis

6.3.1 Heading Controller Analysis

When observing heading data during linear tests, it was obvious that the controller was not performing correctly. The tractor heading was always offset from the target heading and as the tractor speed increased, the offset from the target heading increased. The reason for this was that when the tractor had to travel from quadrant 2 to 1, quadrant 3 to 4, quadrant 2 to 3 and quadrant 3 to 2 the error was calculated incorrectly. The error calculated in these situations was very large which caused the heading controller to apply the maximum change possible to the front wheels. This would force the tractor to turn as sharply as possible to one side until the heading controller determined that the tractor needed to turn in the opposite direction. The reason the offset became large at higher speeds was that the tractor was traveling faster so more distance was traveled before the controller calculated a correction.

Figure 6.14 is a portion of data taken from the first 90º heading test traveling at 1m/s. From this plot it can be seen that between 13.6 seconds and 14.5 seconds the tractor was at a heading less than 90º and had to turn to the right to correct the problem. The controller calculated the error to be approximately 360º when really the error was only between 1º to 3º. If the controller had calculated the error correctly the applied change to the front wheels should have been small, but instead the applied change was the maximum change possible. This meant the front wheels were turned all the way to one side which caused the tractor to quickly veer course. Once the controller determined the heading was above 90º it attempted to turn it to the left to correct the problem. Because the calculated error was small the adjustment to turn the tractor to the left was small and the tractor slowly turned back towards 90º, without turning too aggressively. The

tractor's heading would slowly approach the 90º mark until eventually it crossed to a heading that was once again smaller than 90º and the heading error was again calculated incorrectly.

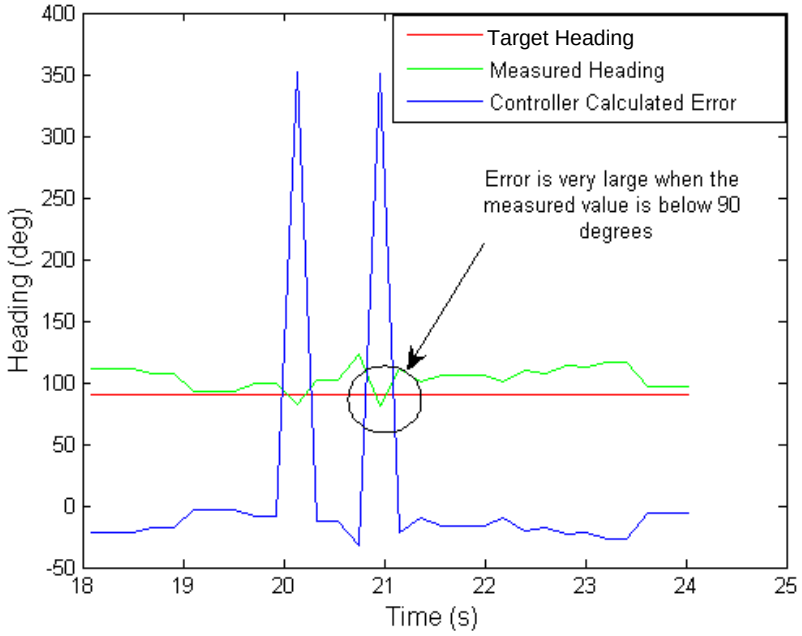

Figure 6.14: Display of where the heading controller was producing an error

Proper controller performance would have calculated correct error terms. This would have resulted in small control adjustments being made, which would have made the controller perform more desirably. At all other tested headings, other than the ones previously listed, the controller did perform correctly by making small adjustments when the error was small. When observing data from the autonomous tests (Figure 6.13) it can be seen that all headings were followed quite closely, except for the 180˚ heading, which was on the boundary of quadrant two and three.

6.3.2 Speed Controller Analysis

The speed controller operated well, but its performance was degraded by poor speed feedback data. The speed controller used a combination of the speed sensor and GPS speed for its feedback data. The controller used the speed sensor until a speed of 0.3 m/s was reached, it then

used the GPS speed. The controller was designed this way because at low speeds the GPS did not produce speed data. The speed sensor on the other hand produced data as soon as the tractor began moving, even if it was moving very slowly.

The problem with the speed sensor was caused by using the speed sensor until a speed of 0.3 m/s was reached. The speed sensor began to fail and as the tractor speed increased the sensor's performance decreased. This meant that even when the tractor was traveling at speeds over 1 m/s the speed sensor would often return speeds less than 0.3 m/s. Because the speed sensor was estimating a speed less than 0.3 m/s this was the estimate used for the speed controller. This resulted in an incorrect speed error calculation, in which the error was much larger than it actually should have been. To correct this, the speed controller would increase the tractor speed causing it to travel at a speed higher than the target speed. When the speed sensor would finally produce data above 0.3 m/s the actual tractor speed was much higher than the target speed so the controller would quickly slow the tractor down to the correct speed. The speed controller was capable of maintaining the correct speed until the speed sensor produced poor data again.

The result of poor data from the speed sensor can be seen in Figure 6.15 where the speed controller was using speed sensor data from 15 seconds until 19.5 seconds. The speed sensor was returning a speed that was generally less than 0.1 m/s so the speed controller caused the tractor to increase its speed. It can be seen that the speed being traveled was much higher than the target speed of 1.5 m/s. At approximately 20 seconds the speed sensor produced data that was above 0.3 m/s, which caused the controller to switch to GPS data. The GPS was returning a speed of 1.8 m/s so the controller reduced the tractor speed to get it close to the target 1.5 m/s. At 23 seconds the speed sensor produced poor data again which meant the speed controller had to compensate for this and once again the tractor speed had to be increased.

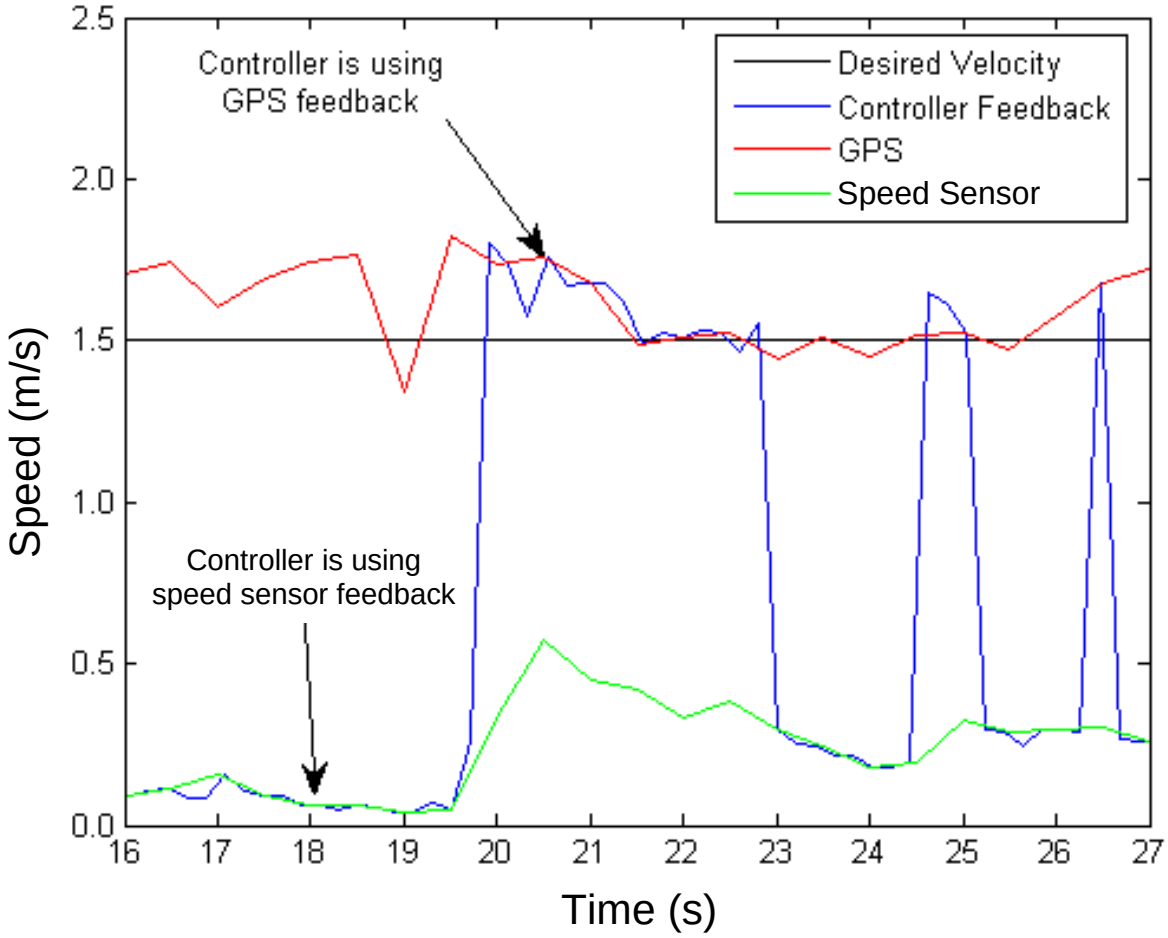

Figure 6.15: Display of where the errors occurred with the speed controller

CHAPTER 7: SUMMARY

7.1 Sensor Platform

When examining the sensor platform it was divided into two components, the heading sensors and speed sensors. The GPS provided data for both components, but the compass was only used for heading and the speed sensor was only used for speed. Each of these sensors had their strengths and weaknesses that were affected by different operating conditions and environmental parameters. After analyzing the data it was decided that the compass should have only been used primarily for measuring stationary headings, not so much reliance should have been put on the speed sensor at low speeds and when the tractor was in motion the majority of data used should have been from the GPS.

This section discusses each of the sensors and controllers. A brief discussion of the sensor and controller failures and successes is also included.

7.1.1 Compass Testing Summary

The compass was used for determining the tractor heading and worked best when traveling at lower speeds. As the tractor speed increased the compass' standard deviation also increased. When increasing the tractor speed from 0.5 m/s to 1.5 m/s the increase in standard deviation was 16˚ for the 90˚ heading and 20˚ for the 270˚ heading. At a speed between 0.5 m/s and 1 m/s the GPS heading data had a lower standard deviation than the compass heading. After performing some basic tests it was found that at approximately 0.7 m/s the GPS had a lower standard deviation than the compass. For this reason the compass was only used when the tractor speed was below 0.7 m/s.

The major factor affecting the compass was vibration, which was introduced by the tractor moving through the field. When performing the stationary tests the compass produced results with a very low standard deviation, but as the tractor started moving the standard deviation increased with speed. This was because the compass was mounted atop a long pole that was used for removing the compass from any magnetic interference. The compass was situated well above the tractor's center of gravity. This caused any change in the tractor's pitch or roll to be magnified by the compass mounting pole. The result of this was excessive movement due to the long pole. This movement was the cause of a higher standard deviation. Some of this variation in compass data was removed by the compass' filtering algorithm, but it was not possible to remove all of the data's variation. The compass would have performed better at a higher speed if the mount was properly designed. The proper compass mount would be capable of removing most of the tractor's vibration and it would remain steady when traveling over bumps.

7.1.2 GPS Testing Summary

The GPS was used for both heading and speed data. The GPS did not produce heading or speed data when it was stationary, which was why the compass was necessary. Once the tractor started moving and reached a speed greater than 0.2 m/s the GPS began calculating the heading and speed data. When traveling at a 90˚ heading at 0.5 m/s during the human driven tests the standard deviation of the GPS was 22.9˚, but this decreased to 4.59˚ when the tractor speed was increased to 1.5 m/s. The decrease in standard deviation was not as significant when changing from 0.5 m/s to 1 m/s. A similar trend was seen during the 270˚ heading and fairly similar results were seen during the computer driven linear tests. The variation in this trend seen in the computer driven linear tests was caused by the poor performance of the speed sensor at higher

speeds. The reason the GPS became the most reliable sensor as the tractor reached higher speeds was because it was not susceptible to vibrations or tractor bouncing.

Initially it was believed that the GPS showed similar results to that of the speed sensor. As the speed increased the standard deviation of the speed data became lower, but only slightly. The main drawback to the GPS was that it did not produce speed data when the tractor speed was below approximately 0.2 m/s. For this reason the speed sensor was used for the controller feedback at speeds under 0.3 m/s and when the tractor speed was above 0.3 m/s the GPS speed was used. During initial testing at the University of Saskatchewan, the speed sensor produced reliable data at low speeds. Once the speed sensor had been used under field conditions, the sensor became damaged and the data produced was not very reliable. This was true even at low speeds. At the end of testing it was decided that the GPS should have been used all the time because it was not as susceptible to being damaged in the field.

After performing a more thorough analysis it was found that there was very little GPS improvement from low speeds to higher speeds. When traveling at 90˚ at a speed of 0.5 m/s, the GPS speed standard deviation was 0.10 m/s whereas at 1.5 m/s the standard deviation was 0.09m/s. This was not a substantial difference and so the GPS should have been used more extensively at the lower speeds. At a 270˚ heading the result was similar with the standard deviation being 0.04 m/s at 0.5 m/s and 0.06 m/s when traveling 1.5 m/s. This once again was an insignificant difference.

7.1.3 Speed Sensor Testing Summary

When the speed sensor was initially developed it was tested at the university. While at the university the tractor was operating at low speeds on level ground and there was an absence of debris that could cause interference. In this situation it performed better than the GPS because

the GPS signal was degraded due to interference from buildings and trees. Once in the field the speed sensor became damaged due to the environmental conditions, which affected its performance.

The speed sensor did not produce the correct data at any speed, but it did become worse as the tractor speed was increased. When traveling both types of linear tests at 0.5 m/s, the average speed from the speed sensor was offset from the actual speed by approximately 0.07 m/s for both directions. When the speed for these tests was increased to 1 m/s, the average speed from the speed sensor was approximately 0.3 m/s slower than the actual speed. During the human driven linear tests, the speed sensor's average speed was between 0.7 m/s and 0.9 m/s below the actual speed.

Some of the reasons the speed sensor became worse as the speed was increased was because the lower speed tests were performed first and the speeds were increased for each test, with the 1.5 m/s test being the final one. This meant that during the 0.5 m/s test the speed sensor was still operating correctly. As the tests continued the speed sensor was damaged due to mud and straw, which caused the data quality to be reduced. The sensor also moved slightly due to the rough ground which meant it was not measuring all the pulses correctly.

7.1.4 Heading Controller Testing Summary

The heading controller was capable of operating effectively under most circumstances, but there were specific situations in which the controller operated incorrectly. The situations where the controller operated incorrectly were common when the tractor had to turn from certain quadrants into another quadrant. In these situations the error was calculated incorrectly and the result was the tractor turning too aggressively in one direction. The controller was capable of slowly correcting this error, but the result was that the tractor's heading was often offset from the

target heading. The situations where this arose were while following a 0˚, 90˚, 180˚ and 270˚ heading, which were also the quadrant boundaries.

During the autonomous tests the heading controller operated well, except when it had to follow an 180˚ heading. At all other headings the tractor turned to the target heading quickly and had little offset. The controller was capable reaching steady state at the target heading within approximately 10 seconds for large turns and it took less time when less adjustment was required.

The tractor was tuned at the University of Saskatchewan where there was very little space to carry out proper tuning techniques. The controller was also tuned at speeds below 0.5 m/s, with no tuning being performed at speeds above 0.5 m/s. The controller should have been retuned when more space was available for performing more turns and when it was possible to tune at different speeds.

7.1.5 Speed Controller Testing Summary

The goal of the speed controller was to receive a speed command and make the tractor accelerate or decelerate to this speed. Once the target speed was attained, the controller was to maintain this speed until another speed instruction was received. The speed controller took an average of 3.2 seconds to accelerate from 0 m/s to 0.5m/s and an average of 1.85 seconds to accelerate from 0 m/s to 1.5 m/s. The larger the change in speeds, the more aggressive the correctional approach that was taken. Using this criterion the tractor could be accelerated to higher speeds very quickly and accelerating to a speed that was 1.5 m/s higher than the current speed would never take more than 4 seconds, but achieving a steady state speed did take more time.

Analysis of the speed controller proved difficult because it was often receiving poor feedback data from the speed sensor. When the tractor was traveling at higher speeds the speed sensor calculated incorrect speed estimations. Often times the speed sensor would calculate a speed lower than 0.3 m/s, when really the tractor was traveling much faster than this. Because the controller received a speed feedback that reported the tractor speed to be much less than the target speed it would attempt to increase the tractor speed. When the speed sensor next calculated a speed above 0.3 m/s the GPS would be used as feedback and the controller would receive a speed feedback that was higher than the target speed. This caused the controller to reduce the tractor's speed. For this reason the tractor speed controller generally did not maintain a constant speed and it oscillated around the target speed. When the computer was controlling the speed, the average speed was often close to the target speed, but the standard deviation was high due to the switching between sensors.

When the tractor was traveling at lower speeds, such as 0.5 m/s, the speed sensor did not report speeds below 0.3 m/s and so the GPS was used as the speed feedback. In these cases the tractor's speed was held at a more constant rate and the speed data did not oscillate as much about the target speed.

The structure of the speed controller should be adjusted to account for the errors produced by the speed sensor. The speed sensor should either be completely removed or the speed controller should be adjusted so it only uses the speed sensor when the GPS is not producing speed data. If the speed sensor had only been used when the GPS was not producing speed data; the speed controller would not have had periods where it was receiving incorrect data from the speed sensor. This would also mean the speed controller could still receive data from the speed sensor when the tractor was traveling less than 0.2 m/s.

CHAPTER 8: CONCLUSION

At the end of this project the tractor was capable of maneuvering through previously programmed tests carried out in a field without having human interaction. The preprogrammed tests performed were 6 minutes long and the tractor operated correctly, except for when traveling at a 180^{o} heading. This was due to a problem with the heading controller programming. During the computer driven linear tests the heading and speed controllers operated correctly, other than a few errors. Most of the errors with the controllers were due to the malfunctioning of the speed sensor and a programming error at three separate headings.

During all tests data from sensors was required. The tractor was capable of using the sensory system to determine the vehicle attitude and carry out commands based on the sensor information. The sensor platform was hindered due to the speed sensor not operating correctly. The speed sensor's performance became worse as more testing was performed and this caused more errors in the controllers.

Based on the performance of the tractor, which is shown in Chapter 0, the actuator control system could successfully control the tractor. The controllers received the instructions either remotely or from a preprogrammed set of instructions and applied these instructions to the tractor. The best example of the controller's performance was shown during the autonomous tests and is displayed in Figure 6.12.

Basic tests were done to integrate a path planning software, developed at the Université de Sherbrooke, into the low level software. It was possible to successfully have the path planning software communicate with the low level software, but no further testing was performed.

CHAPTER 9: RECOMMENDATIONS

9.1 Sensor Improvements

During testing of this system, the speed sensor proved to be a large source of error. This sensor could have been improved by having it weighted or spring loaded to reduce the amount that it was able to bounce when traveling at higher speeds. The wheel that was used had a 17.7 cm diameter, which meant as it traveled over small hills and valleys the wheel speed would increase or decrease even though the tractor speed was staying constant. The small wheel was used to increase the number of revolutions to provide more pulses which increased the resolution of the sensor. To avoid using this diameter of wheel while still maintaining sensor resolution, a larger wheel could have been used to drive a chain that was attached to a sprocket. The Hall Effect sensor would be mounted to measure pulses off the driven sprocket. The two sprockets could have been setup to increase the driven wheel rotations. This also would have moved the Hall Effect sensor further away from the ground where it would have had less contact with debris and should not have been affected by the environmental conditions.

Another sensor improvement that should be made is the compass mount. The isolator block was effective at removing the vibrations which could affect the compass. The isolator mount became an issue at speeds above 1 m/s, because it introduced errors due to excessive movement, in all directions. The isolator block should be improved so it would continue to remove the vibrations, but not cause so much movement. This would improve the compass errors and compass standard deviation at higher speeds.

9.2 Software Improvements

The software developed for this system worked well, but the wireless connection used for collecting data and controlling the tractor would often time out. This timing out was caused by the large overhead involved with RMI. When an RMI client connects to a server a series of handshaking packets must be transferred between the two and this can take many seconds if the network connection is slow. This results in excessive waiting when connecting and if a time-out occurs. For this reason Campadello et al. (2000) recommend not using RMI over a wireless connection. For the wireless connection other methods could have been used such as RF, CORBA or MARIE. This would have improved the testing because there would not have been so many interruptions during the tests due to the poor wireless connection.

9.3 Controller Improvements

The heading and speed controllers performed relatively well. It was difficult to determine their actual performance because of the sensor errors. The heading controller should have the errors corrected. These errors have been corrected within the software, but no testing was performed after making the modifications. The next step in correcting the heading controller is to test the corrected software and verify that it did correct the problem.

The speed controller should have been modified to either only use the GPS or to use the speed sensor and GPS in a different combination. This would have eliminated the errors caused by the speed sensor's feedback, which would have allowed the speed controller to calculate the correct speed errors.

It would be advantageous to develop a model for the tractor's steering system and hydrostatic system. This would allow more robust controllers to be developed and compared to the current

PID controllers. It also would have allowed for testing of the controllers to be performed before implementing them on the tractor, which should have produced better tuned controllers.

CHAPTER 10: REFERENCES

Agriculture, history of. (2007). *In Encyclopædia Britannica*. Retrieved January 25, 2007, from Encyclopædia Britannica Online: http://www.britannica.com/eb/article- 10677.

Arnold J.A. Persaud R. Smallen D., 2000. A more precise sense of where we are. *Public Roads. National Technical Information Service.* 63(4).

Bell T. 2000. Automatic tractor guidance using carrier-phase differential GPS. *Computers and Electronics in Agriculture*. 25: 53-66

Bevly D.M. 2004. Global positioning system (GPS): a low-cost speed sensor for correcting inertial sensor errors on ground vehicles. *Journal of Dynamic Systems, Measurement and Control*. 126: 255-264

Bicho E. and Schöner G. 1997. The dynamic approach to autonomous robotics demonstrated on a low-level vehicle platform. *Robotics and Autonomous Systems.* 21: 23-35

Braasch M.S. and Van Dierendonck A. J. 1999. GPS architectures and measurements: *Proceedings of the IEEE*. 87(1): 48-64

Brooks RA. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA. 2(1):. 14-23.

Campadello S.A., Koskimies O., Raatikainen K., and Helin H.. 2000. Wireless Java RMI. *Fourth International Enterprise Distributed Object Computing Conference. EDOC:* pp 114.

Côté C. Létourneau D. Michaud F. Valin J.-M. Brosseau Y. Raievsky C. Lemay M. and Tran V. 2004. Code reusability tools for programming mobile robots. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems. 2(28)*:1820-1825.

Crawford S. 2005. Performance evaluations of sensor combinations for mobile platoon control. Unpublished M.Sc. thesis. Calgary, AB: Department of Geomatics Engineering, University of Calgary

Farrell J.A. Barth M. 1998. *The Global Positioning System & Inertial Navigation*, New York, NY: McGraw-Hill

Frazzoli E. Dahleh M.A. and Feron E. 1999. Robust hybrid control for autonomous vehicles motion planning. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems. 1(2000): 821-826*

Gat, E. 1998. On three layer architectures. *Artificial intelligence and mobile robots: case studies of successful robot systems.* Dayton, Ohio, March 20, pp 195-210

Goos G. Hartmanis J. van Leeuwen J. 1998. Lecture Notes in Computer Science, Brussels, Belgium: Springer

Kaplan E.D. 1996. *Understanding GPS Principles and Applications*, Norwood, MA: Artech House Inc.

Linsong G. Yong H. Qin Z. Shufeng H. 2002. Real-time tractor position estimation system using a kalman filter. *Transactions of the CSAE*. 18 (5): 96-106

Noguchi N. Kise M. Kazunobu I. Terao H. 2002 Field automation using robot tractor. *Proceeding of the Automation Technology for Off-Road Equipment*, 239-245, Chicago, Illinois, July 26-27

Ollis M. Stentz A. 1997. Vision-based perception for an automated harvester. *IEEE International Conference on Intelligent Robots and Systems*. 3: 1838-1844.

Pilarski T., Happold M., Pangels H., Ollis M., Fitzpatrick K. and Stentz A. 2002. The demeter system for automated harvesting, *Autonom. Rob.* 13: 9–20.

Payton D. 1986. An architecture for reflexive autonomous vehicle control. *Proceedings of the International Conference on Robotics and Automation*. 3: 1838-1845.

Reid J.F. Zhang Q. Noguchi N. Dickson M. 2000. Agricultural automatic guidance research in North America. *Computers and Electronics in Agriculture.* 25: 155-167 .

Roboteq. 2005. *AX2550/2850 Dual Channel High Power Digital Motor Controller User's Manual*, version 1.7, Feb 1.

Wilson, J.N. 2000. Guidance of agricultural vehicles – a historical perspective. *Computers and Electronics in Agriculture.* 25: 3-9

Witte, T.H. Wilson, A.M. 2004. Accuracy of WAAS enabled GPS for the determination of position and speed over ground. *Journal of Biomechanics*, 38(8): 1717-1722.

Yu, H. Chen, Q. and Ozguner, U. 2004. Control system architecture for TerraMax – the off-road intelligent navigator. *5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 378-381. Lisboa, Portugal July 5-7

APPENDIX A: STEERING CONTROLLER PSEUDO CODE

A.1 Quadrant Determination

To determine what quadrant the tractor is currently in and what quadrant the target heading lies in a set of checks can be used. The pseudo code for these checks is in Code Block A-1. This pseudo code can be applied for either target heading or current heading. Once the target and current heading quadrants are known the direction to turn and error terms can be found. The remainder of this appendix shows the four rule sets used when calculating the direction to turn and the error term to be used in the PID controller.

```
if(heading >= 0 && heading < 90)
        quadrant 1

else if(heading >= 90 && heading < 180)
        quadrant 2

else if(heading >= 180 && heading < 270)
        quadrant 3

else if(heading >= 270 && heading < 360)
        quadrant 4

else
        ERROR
```

Code Block A-1: Pseudo code for determining heading quadrant

A1.1 Same Quadrant

When the target heading and current heading is in the same quadrant the controller must determine which direction the tractor must turn and by how much to reach the target heading. In this coordinate system the current heading is larger than target heading as shown in Figure A.1

so the tractor must turn to the right. The pseudo code for this calculation is shown in Code Block A-2. If the tractor must turn to the right the error is positive and if it must turn to the left the error is negative.

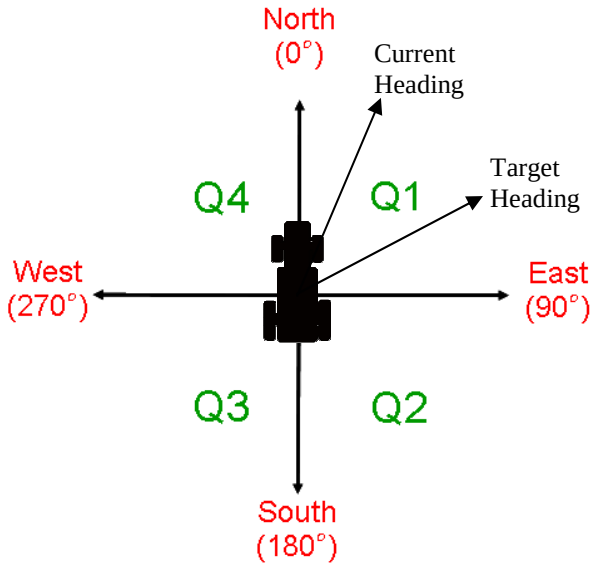

Figure A.1: Coordinate system for when current heading and target heading are in the same quadrant

```
error = target - current
```

Code Block A-2: Calculating heading error when target and current heading are in the same quadrant

A1.2 Target to Right of Current Heading

When the target heading is in the quadrant to the right of the current heading the tractor generally must turn to the right as shown in Figure A.2. In this case the error determination is straight forward because the target heading is larger than the current heading so it is positive and the tractor turns right. In a situation such as Figure A.3 the tractor must turn to the right, but the target heading is less than the current heading so it is a special case. The pseudo code for

calculating error when the target heading is in the quadrant to the right of the current heading is shown in Code Block A-3.

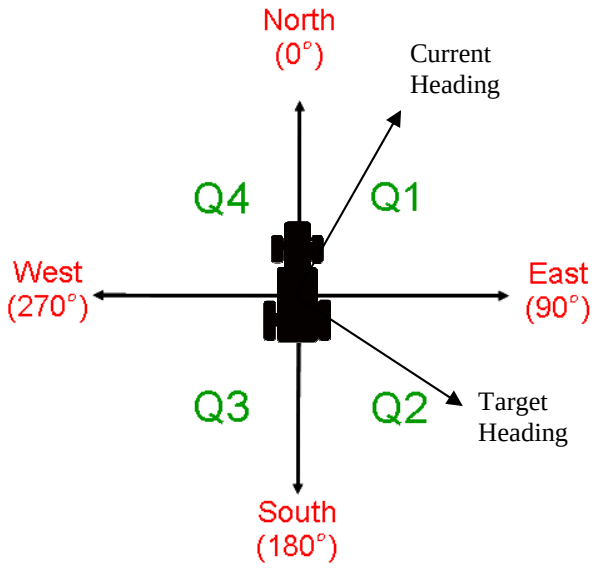

Figure A.2: Coordinate system with target heading in quadrant to right of current heading

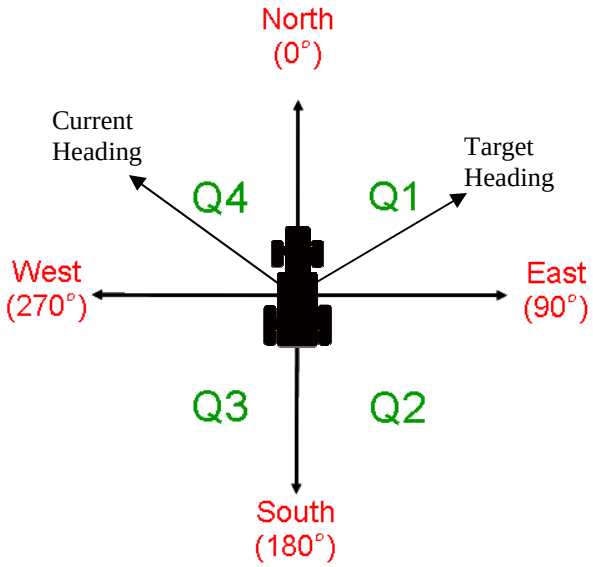

Figure A.3: Coordinate system with target heading in quadrant to right of current heading

```
// normal case
if(current < target)
        error = target – current
        turn right

// special case
else if(current > target)
        error = 360 – (target + current)
        turn right
```

Code Block A-3: Pseudo code to calculate heading error when target heading is to the right of current heading

A1.3 Target to Left of Current Heading

When the target heading is in the quadrant to the left of the current heading the process used for turning the tractor is opposite that used when the target heading is in the quadrant to the right of the current heading. In this case the tractor must turn to the left as shown in Figure A.4 so the heading error must be negative. Here the error determination is straight forward because the target heading is smaller than the current heading so it turns left. In a situation such as Figure A.5 the tractor must turn to the left but the target heading is larger than the current heading so it is a special case. The pseudo code for calculating error when the target heading is in the quadrant to the left of the current heading is shown in Code Block A-4. In the special case the error term is multiplied by negative one because it must be negative to cause a left turn.

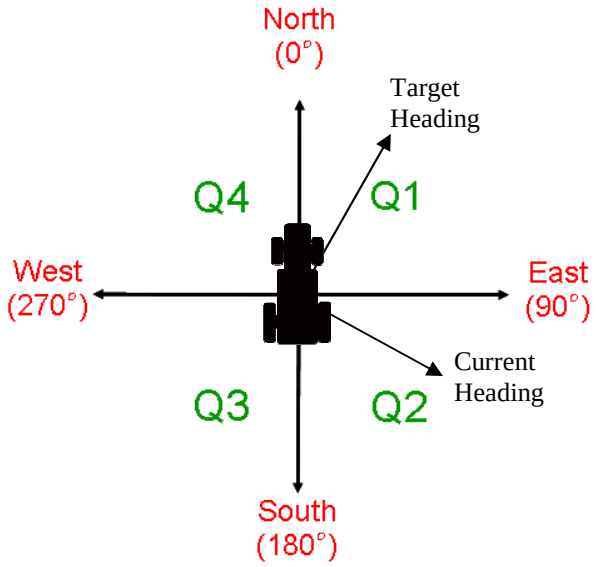

Figure A.4: Coordinate system with target heading in quadrant to left of current heading

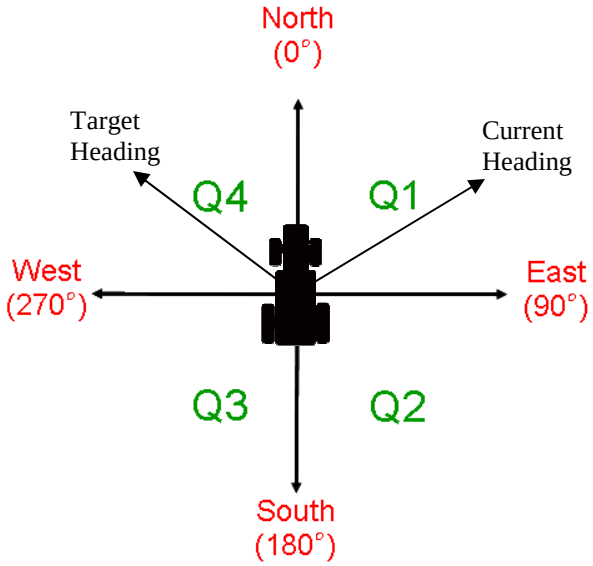

Figure A.5: Coordinate system with target heading in quadrant to left of current heading

```
// normal case
if(current > target)
        error = target - current
        turn right

// special case
else if(current < target)
        error = (-1) * (360 – (target + current))
        turn right
```

Code Block A-4: Pseudo code to calculate heading error when target heading is to the left of current heading

A1.4 Target and Current Heading in Opposite Quadrants

The final case is when the target and current heading is in opposite quadrants. In this case the controller must determine which way has the least distance to turn and then calculate the appropriate error to be used for making the turn.

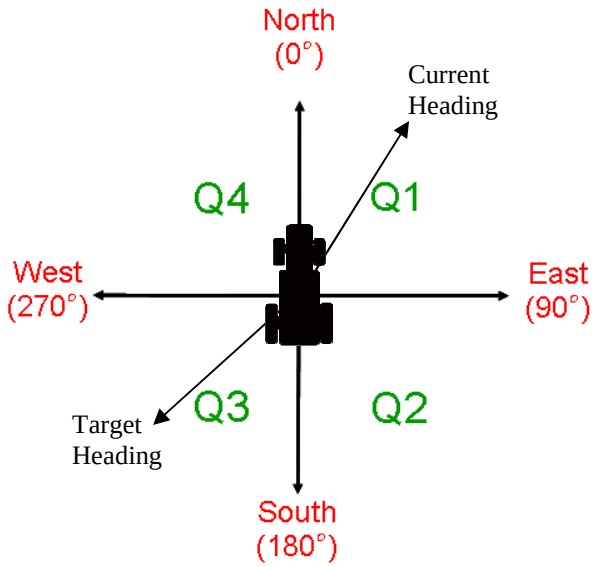

Figure A.6: Coordinate system where the target and current headings are in opposite quadrants

```
if(current > target)
        if((360 – current + target) > (current – target))
                // turn left
                error = (360 – current + target) * (-1)
        else
                //turn right
                error = (current – target)

else if(current < target)
        if((360 – target + current) > (target – current)
                // turn right
                error = 360 – target + current
        else
                // turn left
                Error =  (target – current) * (-1)
```

Code Block A-5: Pseudo code to calculate heading error when target and current headings are in opposite quadrants

The rules above remain the same for each quadrant and are used the same from quadrant to quadrant.

APPENDIX B: ALV ANALYZER DIALOG BOXES

The ALV Analyzer GUI was broken into multiple windows that allowed for easier use of the software application. The most important windows are displayed and briefly discussed here.

B.1 Main Window

The main window is the window that opens when the client side software is started. Nothing can be done on this window until the Connect button is pushed under the Vehicle Link panel. This creates a connection to the server that is operating on the tractor's embedded computer. Once a connection has been established the tractor's navigational information will be displayed in the Vehicle Navigation panel. The Reset button in the Vehicle Navigation panel is used for resetting the X and Y coordinates to zero.

Under the Data Logging panel it is possible to choose which of the five sensor data should be logged. It is possible to log whatever number of sensors you wish to. A filename must be chosen for each sensor and a sampling time can be chosen. To start logging the Start button is pushed and the Close button is used for stopping the data logger and closing the files.

The Disconnect button under the Vehicle Link panel disconnects the client from the server. The Shutdown button disconnects the client from the server and then shuts down both the client and server. The Reconnect server can be used to connect to the server if it has been disconnected.

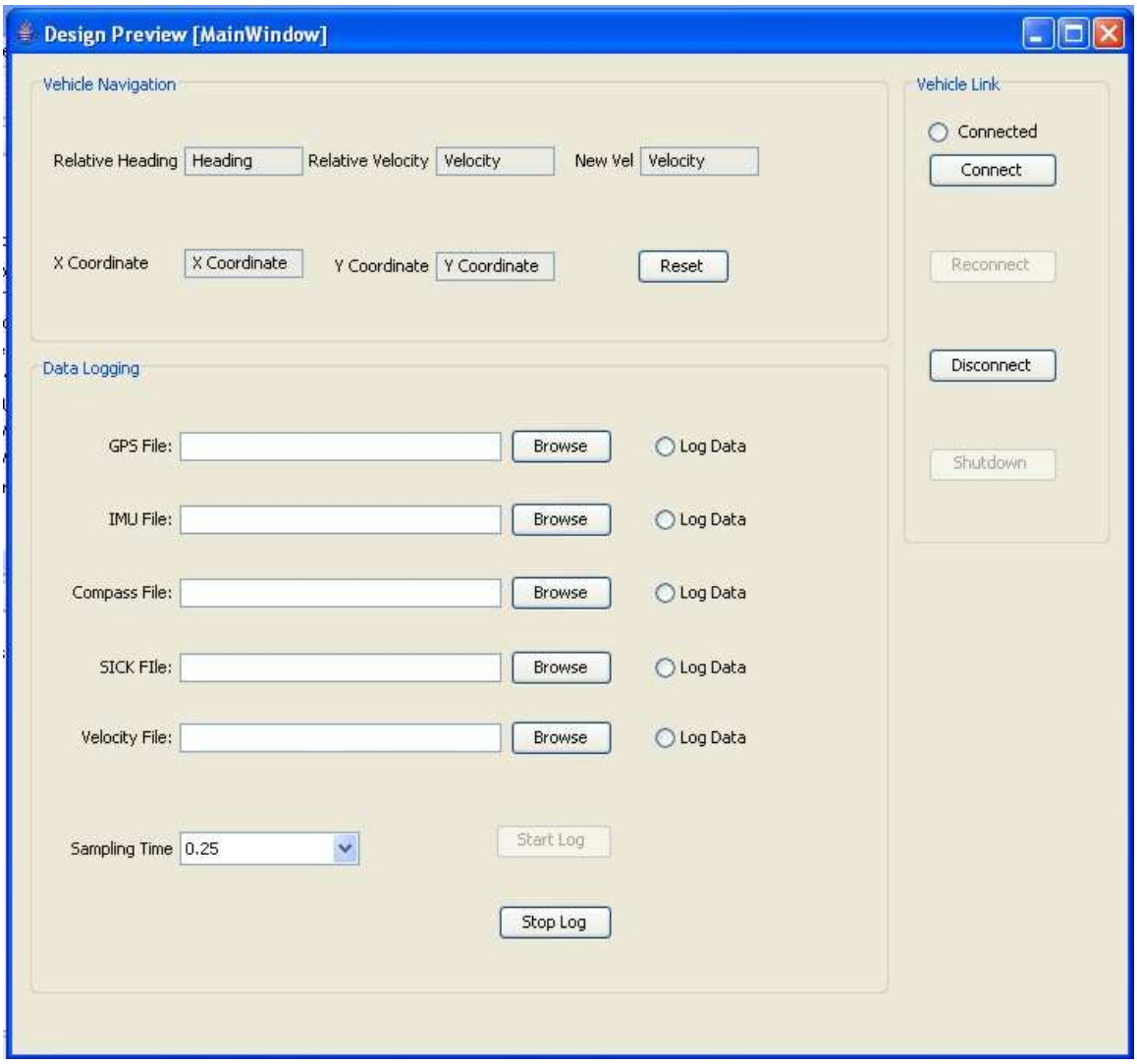

Figure B.1: The main window of ALV Analyzer

B.2 Sensor Display Window

The sensor display window is used for viewing sensor data. When the window is opened after a five second delay is begins displaying navigational data in the Vehicle Navigation panel. The Sensor Display Properties panel is used for selecting what sensor to display, and the refresh rate. The display can then be started, paused and cleared using buttons on the Sensor Display Properties panel. The Sensor Display panel displays a

heading labeling each piece of information and beneath that is a window for showing the actual data. The data continues to appear on the screen until the window becomes full. It then flushes itself and starts again at the top. This window does not log the data, it only displays it.

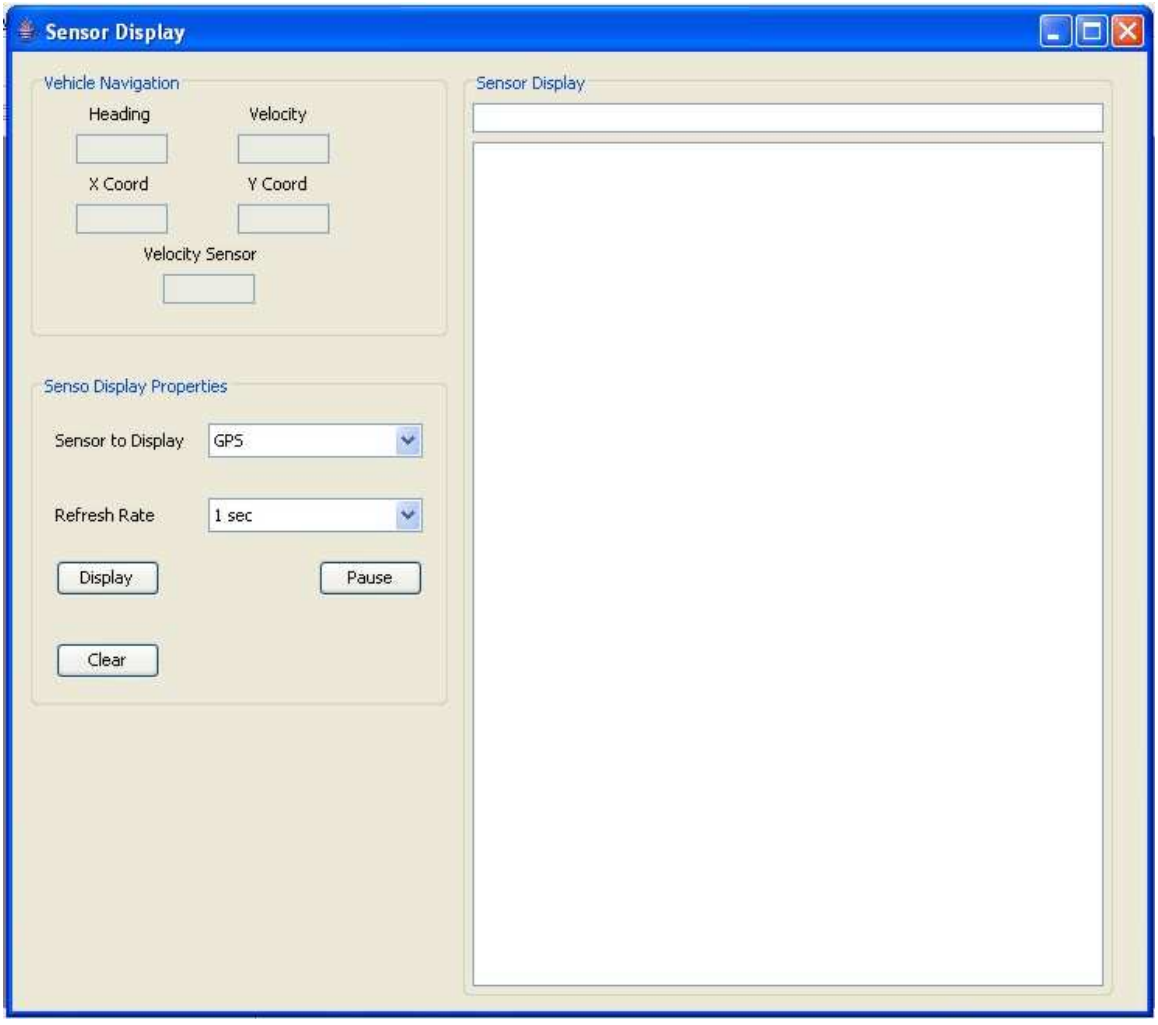

Figure B.2: The sensor display window used for viewing sensor data

B.3 Controller Window

The controller window is used for remotely operating the tractor. This allows for control of the hydrostatic and steering. Under the Hydrostatic and Steering panels the

slider-rails are used for adjusting the commands. This can also be adjusted by using the arrow keys on the keyboard. It is also possible to click on the buttons in the Controls panel to adjust the commands. The Enter key causes the hydrostatic to immediately be moved to neutral and the Shift key forces the front wheels to straighten to an angle of zero. As these values are changed the actual value sent to the motor controller is displayed in the small window at the bottom of the panel.

The Actuator Command History panel to the right of the window displays what values have been sent to the motor controller. Every time a new value is sent to the motor controller these values are updated.

The Properties panel is used to adjust the neutral position of the hydrostatic and the straight position of the front wheels. The controllers use the position of the hydrostatic actuator and front wheel angle to determine the neutral positions. Because the potentiometers that provide this information is susceptible to changes caused by power changes or equipment degradation the controllers do not actually know for sure if the tractor is not moving or the wheels truly are straight. For this reason it is necessary to adjust these properties over time. To adjust the properties the new values can be entered into the textfields in the Properties panel. When the Save button is pushed these values are saved to file so the next time the software is started these parameters are at the correct values.

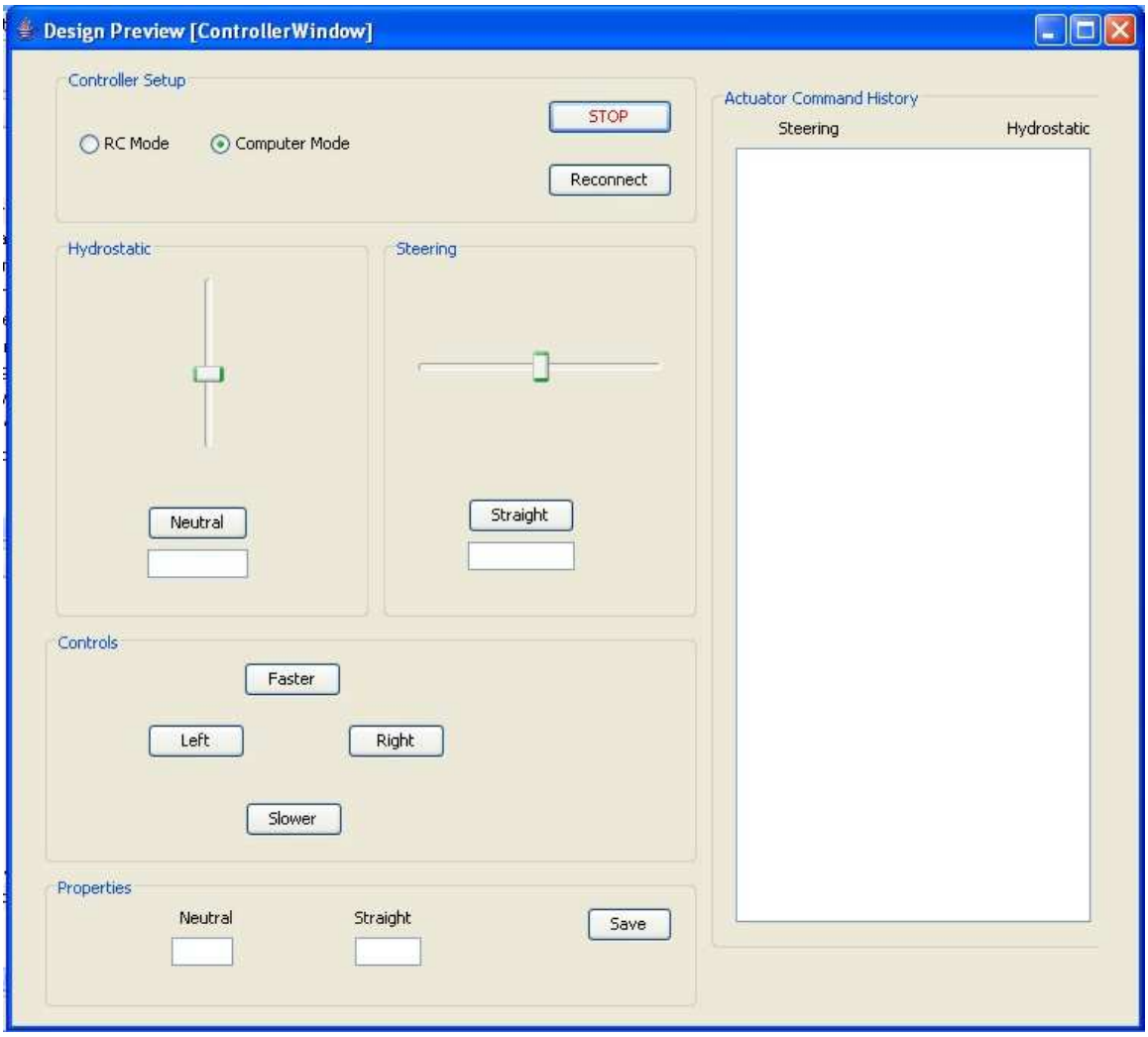

Figure B.3: Controller window used for remotely driving the tractor

B.4 Move Window

The motion window was used for basic testing of the system and controller tuning. The Distance Test panel was used for the first form of control. A distance to travel was entered and when Start was pushed the tractor set the wheels straight and moved the hydrostatic lever to a set position. The other fields on the Distance Test were used for providing feedback on the vehicle's performance.

The Speed Test window was used for developing and tuning the speed controller. A target speed and distance to travel was entered. When Start was pushed the tractor would accelerate to the required speed and hold that speed constant. The wheels were set straight to make the tractor travel in a linear path and the tractor traveled for the distance entered. When the distance was reached the tractor stopped. The other fields on the Speed Test panel were for providing tractor progress to the user.

The Heading Test panel was used for developing and tuning the heading controller. A target heading was entered and when Start was pushed the tractor would start moving. The hydrostatic lever was at a set position and the tractor would turn to try and follow a line that was the same as the line the target heading lied on. This test continued until the Stop button was pressed.

The PID settings window was used for tuning the controllers. Values could be loaded from the properties file and then modified. Once modified a test could be ran. By doing this the controllers were tuned by the trial and error method. Once satisfactory values were found they were saved so that the next time the application was started these satisfactory values would be used by the controllers.

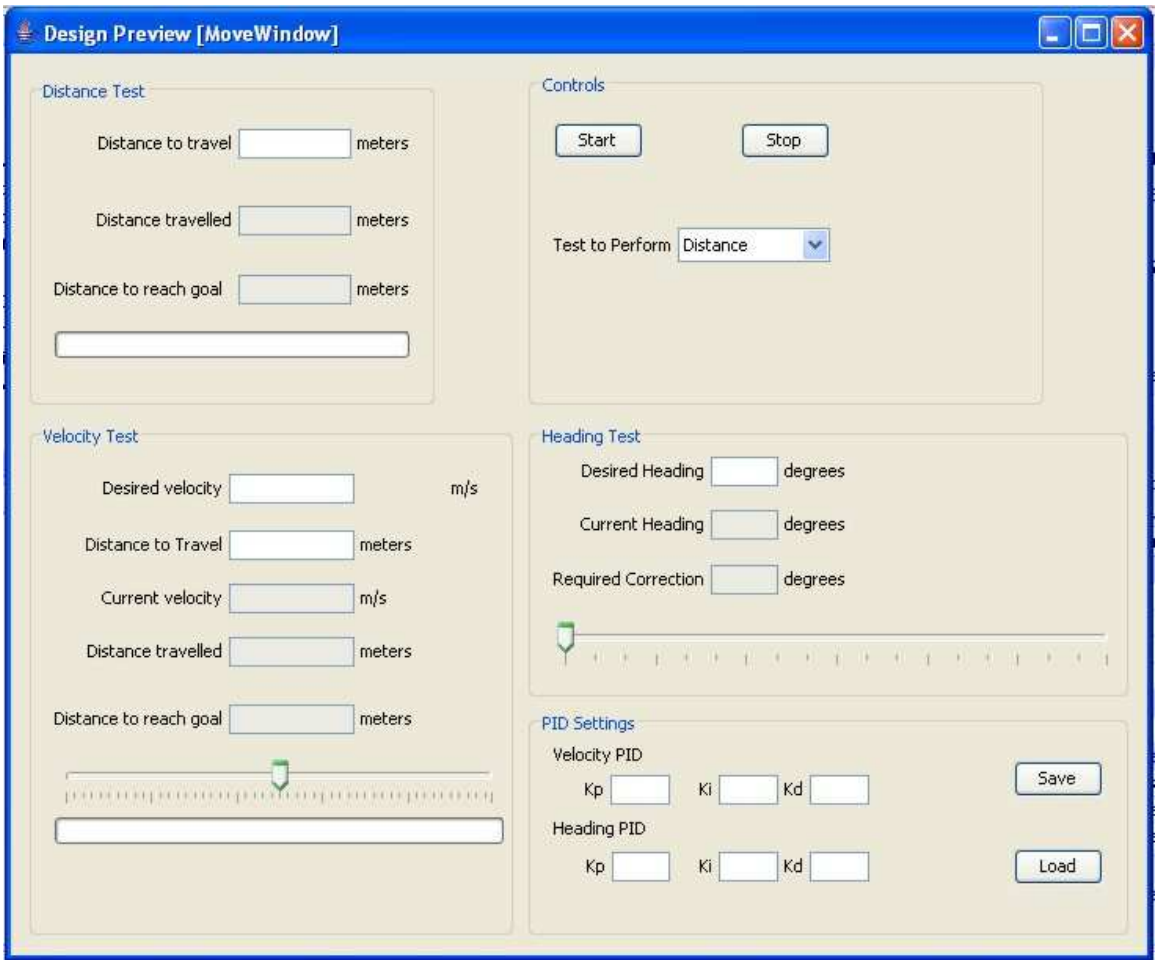

Figure B.4: Move window used for tuning the speed and heading PID controllers

B.5 Dynamic Tests Window

The Dynamic Tests window was used for performing tests of the heading and speed controllers simultaneously or separately. The default was for the test to be performed under complete computer control, but if the Human Steering ON radio button was pressed it gave the user control of the steering. Equivalently if the Human Hydrostatic ON button was pressed it gave the user control of the hydrostatic actuator.

To perform a test an input file containing the commands to be carried out had to be supplied. Filenames to store heading and speed controller data also had to be specified. This test would control the tractor until the end of the input file was reached. While the

test was being performed data was stored in the heading and speed files. When the end of the input file was reached the heading and speed files were closed before ending the test.

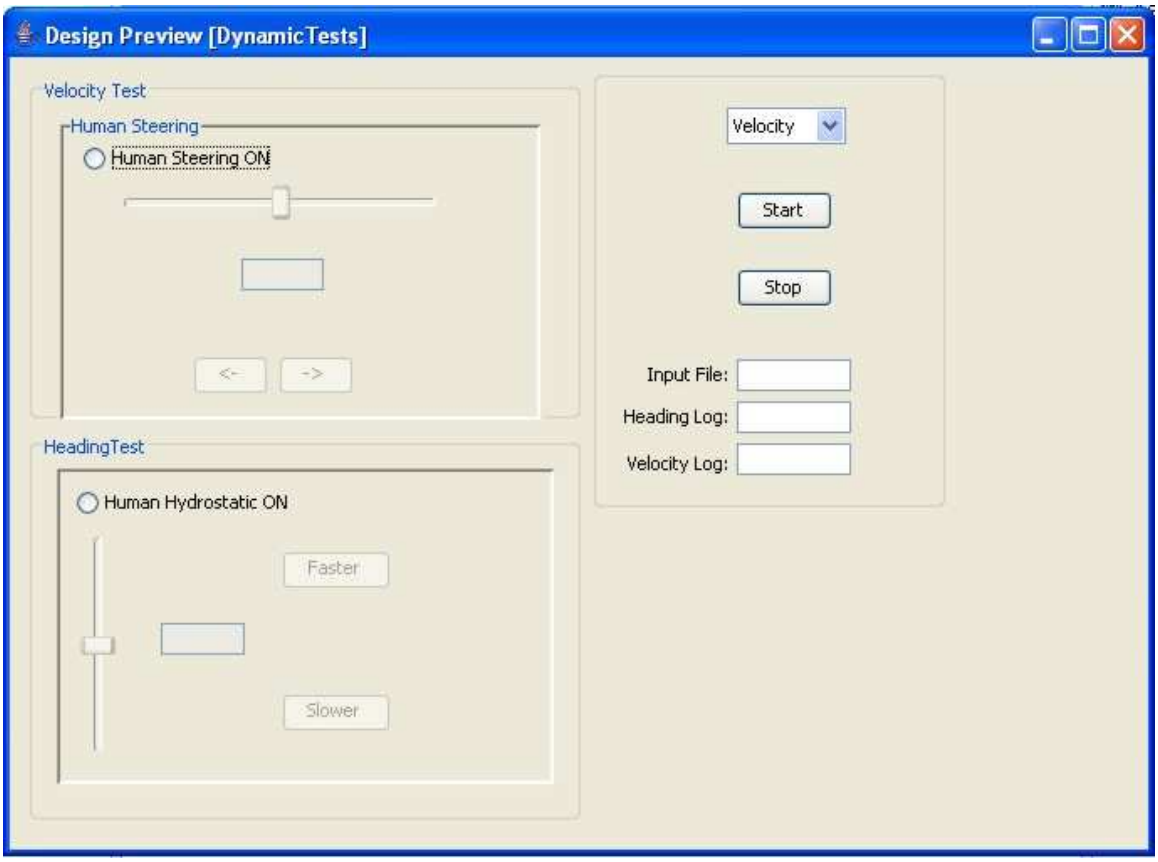

Figure B.5: Dynamic tests window for performing autonomous tests with file input

APPENDIX C: NON-PUBLISHED PAPER ON ALV ANALYZER

Overview of an Analysis Application for Autonomous Vehicle Development

Author: Jason Griffith
Faculty Advisor: Dr. Martin Roberge
Agricultural and Biological Engineering Department
University of Saskatchewan
Saskatoon, Canada
jcg130@mail.usask.ca

Abstract

This paper describes an application, called Analyze, currently being developed for testing, analysis and further development of an autonomous land vehicle. Analyze is primarily written in the Java programming language, with one library being written in C. It provides a wireless remote connection to the vehicle's on board computer through use of Java RMI. Analyze provides a GUI to facilitate remote data viewing, data logging and control of the vehicle while it is in motion. The focus of this paper is the setup of the software and how it is implemented. Safety, reliability and security are also briefly discussed.

1. Introduction

In recent years a considerable amount of effort has been put into development of autonomous vehicles. Removing a human operator is desirable for many reasons, such as military reconnaissance, exploration vehicles and agricultural uses. The concept of an autonomous vehicle is appealing, but operating in an unknown environment introduces many challenges. The largest developmental hurdle is taking a vehicle that is continuous in nature and trying to control it with a computer that operates in discrete space [1]. What this means is the environment around the vehicle is continuous while the computer and sensors are only capable of operating discretely. Computer and sensor response is greatly improving which can lead to the illusion that they are operating in continuously, but they are still effectively discrete [4].

For an autonomous vehicle to be developed, a test application, such as the one discussed within this paper, is utilized. This application is capable of correctly verifying sensor data and vehicle actions while it is operating. To make the application more convenient to use during testing it is best for it to have remote capabilities. The remote capabilities are to allow the software to interact with the vehicle from a computer that is not physically attached to the vehicle.

There are many different methods available for remote interaction. Simple socket connections can be developed or more elaborate solutions such as using middleware software can be used. Because middleware is often designed to ease development of distributed systems it is advantageous to use middleware software. While comparing middleware software the choices were narrowed to Java Remote Method Invocation (RMI) and the Common Object Request Broker Architecture (CORBA). Java RMI is used over a wireless connection to provide the remote link.

2. Vehicle background

The vehicle that is in use is an 860 pound in house custom built vehicle designed specifically for this purpose. The design is a rugged all wheel drive vehicle that allows for operation on a wide range of terrains. The vehicle has a 9 hp Kohler engine that drives a hydrostatic drivetrain with chain drives extending to the front and rear axles. The drivetrain is driven through an electric clutch that can be switched on or off. Having an electric clutch installed provides a method to safely disable the drivetrain and prevent accidents. The vehicle has full suspension and front wheel steering. The vehicle used for testing and development of this software is displayed in Figure 1.

Figure 1: Autonomous Vehicle used for testing and development of Analyze software

The vehicle is equipped with actuators and a collection of sensors. The actuators are used for steering and hydrostatic control, while the sensors are for sensing the environment around the vehicle and recording different vehicular parameters. The sensors in use are a Superstar Flexpack II GPS receiver from Novatel, IMU300CC inertial measurement unit (IMU) manufactured by Crossbow Technology Incorporated and a TCM 2.6 digital compass from PNI Corporation. Electrak E150 actuators produced by Danaher Linear Motion Systems are used to control the steering and hydrostatic. To control the actuators an AX2550 motor controller manufactured by RoboteQ is used. The onboard computer is an IBM Thinkpad with a Debian Sarge operating system.

The vehicle is designed to operate in either remote control (RC) or computer controlled mode. In RC mode an RC receiver receives commands from a remote and transfers them to the motor controller. When in computer control the motor controller receives its input commands from a computer. All motor control commands and data messages from sensors are sent and received by the onboard computer.

3. Software overview

The software that is developed is for testing and analysis of an autonomous land vehicle (ALV). A wireless communication link, using an off the shelf wireless router, is created to allow the transfer of control commands and sensor data. The application allows for remote computer control of the vehicle actuators, which means the vehicle can be remotely operated from a computer. The application is also capable of logging all data that the user requests and saving it to file or simply displaying it on a monitor.

The application utilizes Java RMI for its communication between the server and client. The reason for using Java RMI is that the application is programmed in Java, except the IMU driver which already has a JNI library to allow it to function in Java. One drawback to using Java RMI is that there is a large overhead when sending packages, which causes slower transfer speeds while using wireless networks [3]. These transfer speeds can be improved through different methods, but for this work it is not necessary. CORBA is another suitable middleware, but because only Java programming is used it is not used.

Communication with the majority of sensors and motor controller is carried out over RS232 connections. Each sensor and the motor controller have a properties file that is used for setting of the serial port before connecting to the sensor. This makes it simple to change the serial ports that are in use. The IMU does not have a property file and is always used on the same serial port.

Analyze provides support for a SICK LMS 2 dimensional laser scanner, but the scanner is not currently installed on the vehicle. For this reason some of the laser functionality is included in the software, but it is not complete. If the laser is going to be utilized, setting the laser up would require minimal effort. A laser package is developed, but not included in the software and not discussed in this paper.

4. Software structure

The application consists of two separate applications; server side and client side. The client side application is programmed in Java and the server side is primarily in Java with the IMU driver being programmed in C. The server side is installed on board the ALV and carries out actual interactions with the vehicle. The client side has a remote connection to the server and is used for sending commands to the server as well as receiving data from the server. The remote connection between the server and client is carried out using Java RMI. A display of the application with both server and client is shown in Figure 2. The figure also shows the packages used by the server and client applications. The only package shared between the server and client is the transfer package. This package contains the remote interface definition, remote interface implementation and the stub class.

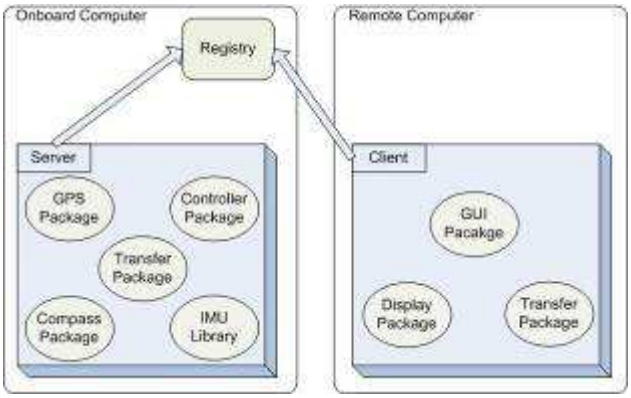

Figure 2: Software overview showing the RMI connection and packages in use

4.1 Server side

The server side application is designed to operate on a Debian Sarge operating system with kernel 2.6.8. The server is for establishing communication with all sensors and the motor controller. It then binds the interface implementation to the RMI registry for allowing remote connections to its methods. The interface

implementation then allows methods for retrieving sensor data or sending motor controller commands to be called remotely.

The server is broken into several packages, as displayed in Figure 3, with each sensor and the motor controller having its own package. The IMU is the only component that does not have a separate package. It has a native library implementation instead. Each of the sensor packages is used for establishing connection with the sensor, sending commands to the sensor, receiving data from the sensor and then formatting data into a usable format. The motor controller package is used for sending commands to the motor controller, as well as receiving analog voltages from the controller. The transfer package is necessary for interacting with each of the packages as well as providing methods that are bound to the registry so they can be called remotely.

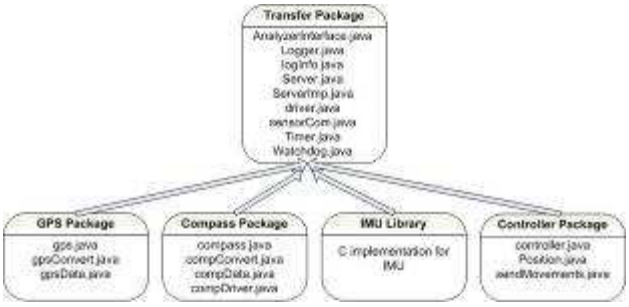

Figure 3: Server side packages with their classes

As required by Java RMI, an interface is developed for allowing remote access to some methods on the server side. This interface only contains methods that are accessible remotely. The methods are used for receiving formatted sensor data, sending movement commands to the motor controller, closing comm. port communication, shutting down the server and reconnecting to the motor controller.

All methods declared in the interface are implemented within the ServerImp.java file. The ServerImp file also carries out all communication with sensor and motor controller packages. When the ServerImp class is initialized, a connection with the GPS, compass and motor controller is created. Once GPS communication is established the starting X and Y coordinates are found from the initial latitude and longitude. The initial X and Y coordinates are used for establishing a zero position for all testing. These coordinates can be updated for different tests. Native methods are also declared to establish communication with the IMU library.

When all communication is established, a thread for sending commands to the motor controller is started. This thread is used for checking an object that stores the target actuator position and sending that position to the motor controller. The thread operates in a continuous loop with a 100 millisecond delay, which means the motor controller commands are checked and updated every 100 milliseconds.

Once the ServerImp initialization is complete the class waits for remote calls to methods. When a call to a method is received the method is carried out and any results returned to the client. The most common method calls are to retrieve data from sensors and return them to the client or to send commands to the motor controller. Data is transferred between the client and server as strings or string arrays so that it can be serialized for transfer. When control commands are sent for the motor controller the ServerImp class uses those commands to reset an object where the motor controller thread checks for changes. Within the next 100 milliseconds the thread will read that change and pass it to the motor controller. The ServerImp class also releases the locks on each serial port that is in use by the sensors when the server is closed. This makes it possible for the connection to be reestablished or for a different application to access the communication port.

The packages used by the ServerImp class are the gps, compass and controller packages. All three of these packages use the RXTX 2.1.7 native library for communication with the serial ports. The RXTX library is a native library that offers serial and parallel port support within Linux because the Java Communication API is only supported in Windows environments.

The gps package has three classes. One class handles all serial port communication. It opens the port and creates a lock on it so other applications can't access the port. An interrupt handler is created to handle all incoming data from the GPS. The GPS continually transmits data at 1Hz and each time new data is available an interrupt is triggered. The interrupt reads all the data in and then sends it to a translator class. The data that is read in from the GPS is retrieved in the National Marine Electronics Association (NMEA) standard. All data is received from the GPS, but the message ID tag is checked and if the message is not navigational data it is discarded. Data is read in as integers and then must be converted to its proper format.

The data received in a navigation message is the date, time, latitude, longitude, altitude, ground speed, track angle, north speed and vertical speed. Other data is received, but it is not used and so it is never converted to usable forms. The data that is converted is received as Unicode characters (Uchar), doubles, Unicode shorts (Ushort) and floats. To make the data usable in Java, the Uchars are converted to integer values and Ushorts are converted to short values. The double and float values must also be converted into Java signed floats and doubles. All data is transmitted with the least significant bit being the first bit received. For this reason the complete message is put into binary format

and from that the proper conversions are obtained. As data is converted it is put into arrays. Each time a new navigation data message is received the old data is discarded and replaced with the newly formatted data.

To interact with the package a driver class is included. All communication with the class is made through the driver class. Examples of tasks that are carried out through this driver is requesting data, opening the serial port and closing the serial port.

The compass package is similar to the gps package. It has three classes; one for communication with the package, one for communication with the serial port and one for translating data. The class that carries out all communication with the package is used for opening the port with all necessary settings, closing the port and requesting data.

The class for communication with the serial port sets all necessary settings and opens the serial port. A lock is put on the port so other applications can not access it. An interrupt handler is setup so that when new messages are received they are automatically read into the program. The interrupt receives data strings from the compass at a rate of 8Hz in ASCII format. Once the message is received it is passed to a conversion class.

The conversion class is used to take raw data in ASCII format and translate it into an array of strings. This breaks the data into separate components so it is more usable for comparisons and corrections with other sensors. The data received from the compass is heading, pitch, roll and temperature. Each section of data is lead by either a 'C', 'P', 'R' or 'T' and followed by their value in ASCII format. This makes breaking the data into separate components very simple. The most important piece of data from these for the Analyze software is the heading. Whenever new data is received the conversion class discards the old data and replaces it with newly formatted data.

The controller package has a similar structure to the gps and compass, except that it does not have a class for communicating with the package. The ServerImp class starts the controller communication class which locks the serial port and sets it up for communication with the motor controller. A thread is then used to check for new motor commands and write those commands to the motor controller every 100 milliseconds. When an analog voltage is requested the request is sent directly from the ServerImp class to the controller serial port communications class instead of through a package communication class. The major difference with the controller package is that it does not have an interrupt handler. A request for analog voltages is the only task that requires input from the motor controller so it is executed using a command that writes the instruction to the serial port followed by a read data from the serial port command.

Data being transferred to the motor controller are inputted as integers that range from -127 to 127. These values are then converted to a hexadecimal string and the string is then converted to an array of characters. The motor controller receives ASCII values as inputs so the array of characters can be outputted to the motor controller, one character at a time. The analog voltages received from the motor controller are received as hexadecimal values and then converted to integers. By converting them to integers it makes it simple to compare the analog voltages to actual commands sent to the actuators because they are both in the range of -127 to 127. When the motor controller is in RC mode it is not possible to receive analog voltages.

The controller also has a thread included in it to act as a safety measure. The thread repeatedly polls the time that the last command was sent. If this time is more than five seconds the controller places the hydrostatic actuator in neutral and straightens the front wheels. This is to prevent the vehicle from operating without control for more than five seconds. The vehicle could go out of control if connection was lost or the client side crashed.

The IMU does not have its own package, instead it uses a library. The IMU driver is written in C programming language so the Java Native Interface is used to communicate with it. Calls to retrieve information from the IMU are made through native function calls in the ServerImp class. Before the Analyze application is started the IMU server must be started to communicate with the IMU. This is necessary because the IMU driver is setup as a server-client application. The IMU server is left as is, but the client is replaced by the ServerImp calls.

The actual server main method is found in a file Server.java. Within this class a static shared library is loaded for the IMU. This library is loaded as static to ensure that only one instance of the library will exist within the Java Virtual Machine (JVM). The server then creates and installs a security manager, which is required for Java 2. The security policy used is to grant all permissions. The server then creates an instance of ServerImp and binds it to the registry.

4.2 Client side

The client is used for providing a remote link to the vehicle in which data can be viewed and saved as well as having motor commands issued. The application's client side is designed to operate on a Windows XP machine. The client connects to the server to send and receive information.

The client only uses three packages; the GUI package, display package and transfer package. These packages and their classes are displayed in Figure 4. The transfer package contains classes necessary for

sending remote method calls, the GUI package displays options and data on the monitor and the display package is used for updating some parameters within the GUI.

Within the GUI package both JFrame and JPanel forms are used. The MainGUI.class is the class that is first initialized. MainGUI creates a tabbed frame that allows for easy switching between different windows for control and analysis. The class immediately creates objects for communicating with the remote interface. One of the objects created is sensorCom.class. This class creates the actual link to the server. Through a dialog box the user inserts an IP for the server they wish to connect with. ServerCom then tries to connect to a server using that IP and when it does connect an instance of that connection is stored. A connection to the server now exists and any remote method calls can be made.

The tabbed windows inside the MainGUI are Main Window, Sensor Window, Controller Window and Move Window. All of these windows are JPanels that add functionality to the application.

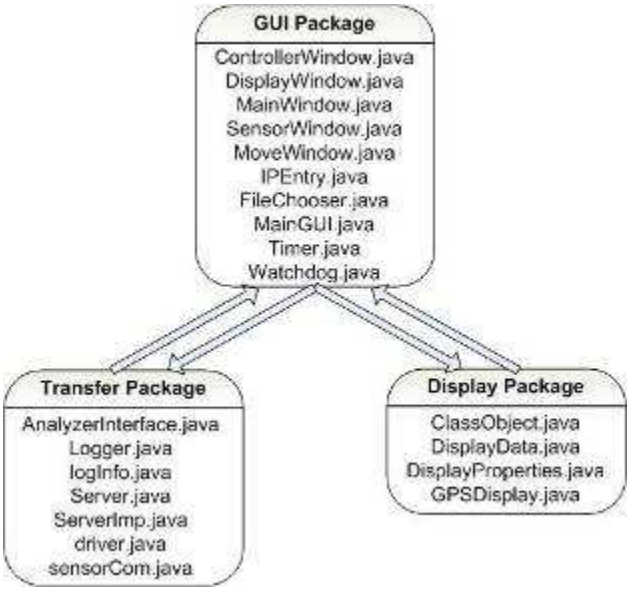

Figure 4: Packages used by the client and all classes included within them

The Main Window is used for connecting to the server, logging data and displaying basic vehicle parameters. The Main Window is the default window that is displayed when the application is started. The only options available to the user at startup are to connect to a server or to exit the program. If the user chooses to connect to a server a dialog box requesting the server IP address is displayed. Once an IP address is entered the IP is passed to the sensorCom class. The sensorCom class uses the IP address to look for an RMI registry at that IP address. The default port is set to public and a user does not have the option of changing this. Once the connection is established it is passed to another class that simply waits for other classes to request remote methods.

The final task for sensorCom is to start logging data if required. Using the Main Window filenames for each of the sensors can be inputted. All files default to the csv extension to allow files to easily be imported into a spreadsheet. If a user checks the box beside the sensor then when the start button is clicked sensorCom will receive all the filenames from the Main Window and passes them to an object that stores the data logger settings. It also passes the sample rate to the data logger object. A thread is then started that receives data from the server and saves them into their appropriate files. The data logging thread continually checks the data logger object to see what the sampling time is and if it should continue logging. If the data logging is continuing then the logger receives the data, writes it to the required files and then sleeps for one sampling time. If the logging is finished then the logger closes the files, but continues looping and sleeping at the required sampling time. If a user supplies more filenames and requests data logging resumes then the logger will start writing data to those new files. This means a thread does not have to be created for each data logging session; instead the old thread just resumes logging with either the same filenames or new files if requested.

Another component in the Main Window is the vehicle navigation information. The navigation information is currently received from the GPS, but as development continues it will be combined with the compass and IMU data to provide more accurate data. When a connection to the server is established a thread is automatically started and created. This GPSDisplay class is a thread that continually loops requesting a heading, speed, X and Y coordinates from the server. On every loop the thread sleeps for one second before retrieving new information.

Another window that is within the tabbed pane is the Sensor Window. This window is used for displaying sensor data. The sensors that can be viewed in this window are GPS, IMU, compass and analog voltages. Only one sensor can be viewed at a time within the window, but the sampling time can be changed to 0.25, 0.5, 1, 2, 5 and 10 seconds. When this window is created a thread is started that reads its settings from a properties object. The properties object stores which sensor should be displayed and what the sampling rate is. This window also has a button to create a new sensor window. A sensor window that is outside the tabbed pane is useful because it allows data to be viewed while viewing other windows such as the controller window. It also allows for multiple sensor windows to be open so multiple sets of data can be viewed simultaneously. The

separate display window is an exact replica of the Sensor Window except that it also includes the navigation display included on the Main Window.

The Controller Window is another window in the tabbed pane. This window is used for sending and displaying motor controller commands. When the window is in computer mode the arrow keys on a keyboard can be used for controlling the actuators which control vehicle motion. A slider is also available for each actuator as well as buttons that can be clicked and there is a text area where positions can be manually entered. A button to return the hydrostatic actuator to neutral and to set the steering actuator to straight is also available. There is also a button that simultaneously returns the hydrostatic and steering actuators to their middle position. The final part of this window is a display to show a list of recently sent motor commands.

The Controller window does not use any threads. The only time it is updated is when there is an event, such as a new motor command being sent to the motor controller. A reconnect button is on the window for instances when connection with the motor controller is lost. This is also useful if the motor controller is changed to remote control mode for some maneuvers and then put back into computer control mode. This button calls a remote method that sends ten new line characters to the motor controller, which places the motor controller back into computer control mode.

4.3 Software evaluation

Evaluation of the software is done in incremental steps to ensure that all individual packages are operating before it is combined into one complete application. Initially sensor packages are tested and as new functionality is added to the software it is tested.

To be able to test the GPS package a simple console based application is developed that interacts with the GPS package. Using this application it is possible to connect to the GPS and request data from it to ensure it is correct. Initially GPS testing is performed inside a building to see that communication is established and then that data is formatted properly. Once data format appears to be correct, further testing of the GPS is performed outdoors where it is possible to receive actual satellite signals. Stationary tests are performed and compared to another GPS or to the same GPS receiver using evaluation software. Once the GPS is receiving correct results in different locations it is tested when in motion. To do this the application is setup to log data for a set amount of time and save it to a file. The data within the file is reviewed after the test to confirm that correct path data is received. The compass and IMU have similar testing procedures where they are tested inside in a stationary position and then moved outdoors to be tested while in motion.

The motor controller requires slightly more testing because it must have both stationary and mobile testing, but first it must be calibrated. To do this the full travel of the actuator is found so that the maximum and minimum values that can be passed to the motor controller are known. The positions that put the hydrostatic in neutral and the steering straight must then be found too. Once these values are known they are hard coded into the application to work as software limits to prevent damage to the actuators and any mechanical parts.

With the packages tested and verified, testing of the Analyze software is started. Each window within the GUI is tested for functionality and to ensure correct data is being received.

The main window is tested first because the first functional component that is necessary is being able to connect to the server. Once it is possible to connect to the server the data logging was setup to be sure the files were being opened and closed properly. Once files are being opened and then closed it is tested to see if data is actually being received and written to the files correctly. The starting and stopping of data logging as well as the sample rate for each data piece is also tested. With the data logging receiving data and writing it to files correctly it can be assumed that the sensor window is also able to receive correct data because they both use the same method calls.

With the main window capable of receiving data it can then be tested to see that the navigation thread is correctly receiving data and displaying it in the vehicle parameter section. Once this is verified the functionality of all remaining buttons such as stopping communications and shutting down the server is verified.

To test the sensor window functionality of the different options is tested. First is to see that when a different sensor is selected for display the title is changed appropriately. Then the start, display and clear button's functionality are verified. To verify these buttons the application can either be setup to receive actual data or it can receive a string just to display that it is displaying input. Even with useless input it is possible to test the functionality of each of the buttons. Using the method of having either real or fake data can then be used to verify that the sampling rate does change when a new rate is selected. Once full functionality is verified actual data is fed into the sensor window and functionality is checked once again. The sensor window is then tested outdoors with a moving vehicle to verify correct data is being displayed. The last button on the sensor window that is tested is the New Window button. This is tested last because the new window that is displayed is basically a copy of the sensor window with

one new area from the main window added on. If the main window and sensor window are both operating correctly then the new window is most likely also operating correctly. This is quickly verified by testing it outdoors with a moving vehicle.

The next window that is tested is the controller window. The functionality of the window is tested by testing each button and ensuring that the slider moves to the appropriate location and the proper value is displayed in the value area, as well as in the actuator display area. With proper values being received from the user the motor controller and actuators are connected. The controller window is then used to test that the actuators do move to the positions they are told to move to. It is also checked to ensure that the actuators are connected with proper polarity so that they don't operate opposite to the controls. Once thorough stationary testing has been performed the vehicle is tested outdoors. To do this the vehicle's engine and transmission is engaged and the vehicle is then driven by the computer.

Evaluation of the software is an ongoing process. As vehicle parameters change and the software is updated it must be further evaluated to ensure correct results. Some examples of this are if the steering is reset so that the actuator's travel changes. In this case the steering actuator would have to be recalibrated and the software limits would most likely have to be changed. Another example is if an autonomy window is added into the Analyze application to make the vehicle carry out some basic maneuvers. This makes it necessary to test all data being received and all commands being sent. This testing expands the functionality of the software, but more importantly it incrementally provides safe testing measures for the vehicle. With this testing it also provides a method of logging the data being received and commands being sent.

5. Safety

With a software application such as this, safety is an important concern because the software is being used to control a large vehicle. If the vehicle goes out of control, it is necessary to have a strategy in place to ensure it will not do any serious damage.

One software safety mechanism that is incorporated is a watchdog timer. The watchdog is used for checking how long it has been since a request from the client has been received by the server. If the server has not received a request from the client within the past five seconds the hydrostatic actuator is placed into neutral position and the steering actuator is placed into the straight position. This is important for a circumstance in which communication with the client has been lost. Without a watchdog the actuators will not reset themselves, instead they will stay in the command sent position. If this last position is such that the vehicle is moving forward then it will continue traveling forward and will not stop until something or someone interferes with it.

To avoid poor communication between the computer and motor controller it is setup so it is not possible to automatically recover from errors. With the compass and GPS if communication is momentarily interrupted, once the connection is reestablished the software automatically resumes receiving data. With the motor controller if connection is lost the application does not attempt to reconnect. The connection is lost and the user must manually request a software reconnection. This is achieved by having the motor controller set to default into RC mode. To establish communication with the motor controller ten carriage returns must consecutively be sent to the motor controller. A button to allow a user to reconnect to the motor controller is placed in the controller window. This button simply transmits the ten required carriage returns to force the motor controller from RC to computer control. Once the motor controller is returned to computer control it responds normally.

Related to the inability to reconnect is an RC override switch. The hydrostatic actuator input to the motor controller is always connected to both the computer and RC. This allows the RC remote to be used to alter the vehicle speed or even stop the vehicle. This is possible even if the motor controller is operating in RC mode. The steering actuator is either computer controlled or RC controlled, but not both at the same time. When the RC override switch is set to RC, only RC commands are received by the motor controller. When the switch is set to computer mode the steering only receives computer commands, but the hydrostatic can receive commands from both the RC and computer.

For additional safety concerns the vehicle is also outfitted with a few hardware switches. One switch will disable all electronics if pushed. This switch is a large, easily identifiable push button switch placed on the top of the ALV. The main safety switch is a remotely operated clutch switch. This switch enables and disables an electronic clutch that engages or disengages the drivetrain. When the clutch in engaged the drivetrain is operating, making it possible for the vehicle to move. When the clutch is disengaged the vehicle is not able to move at all.

6. Security

With this application security is not a major consideration. The ALV is generally operating in open areas away from the internet. This means that for any computer to connect they must be within range of the

local area network. This is however somewhat possible because it is a wireless network that could easily be connected to. The wireless network used for this application is security enabled with a WEP key and could easily be setup to do MAC address blocking. To further increase security the wireless network could be replaced with cables. This would mean the vehicle would be tethered and extra care would have to be taken when testing it.

Another method of making the application more secure is to modify the RMI security file. Currently it is set to allow all connections and does not restrict what methods can be accessed. This security file could be adjusted to be quite restrictive if necessary. If target the RMI registry could operate on a port other than 1099, which makes it more difficult for unwanted access because other applications would have to know what port it is operating on.

Within this application Java RMI is used with a very open security policy, but because RMI is used it does provide some protection. For a client to invoke a method on the server they must first have a reference to the object. A reference to an object can be retrieved by having the stub class and all other classes locally on their computer [2]. The application is not setup for dynamic stub loading so the only way for somebody to connect is to have a copy of the stub file. Without this stub file and a copy of the required classes it is not possible to connect to the server.

7. Recommendations

The software developed thus far is excellent for basic testing, but further development is recommended. The current vehicle parameter data, which consists of X and Y coordinates, heading and speed, is not as accurate as it can be. It is necessary to combine the GPS data with compass and IMU data. This allows compensation for errors in each of the sensors as well as providing data even if the GPS is experiencing a signal outage. Each sensor is not completely accurate, but by combining them together the result is a more accurate estimation of the vehicle parameters.

Some further development required is a window to perform basic testing of the vehicle while in autonomous mode. This window is currently under development for testing shortly. The window is to allow for testing of basic maneuvers, such as straight line tests, distance tests and basic heading tests. This window allows for verification of autonomous control while maintaining a partially controlled environment. Another development is to create a window that allows for simulated instructions from path planning software. These instructions are in the form of target speed and heading. Using this window the target parameters can be inputted and the vehicle then carries out the required processes to reach the target heading and speed.

8. Conclusion

The application that is developed is capable of receiving ALV data and sending it motor controller commands. The application includes a server and client that operate over a wireless network. The server side application is used for interacting with the vehicle, while the client is used for sending and receiving requests to and from the server. The server formats all data received from sensors prior to transmitting it to the client. The server also converts all motor controller commands before sending it to the motor controller. The client includes a GUI that allows users to easily navigate the software as well as display data, save data and send motor commands. The application is tested to ensure that its functionality is operating correctly and reliably. Further development and testing of the application is going to continue as vehicle development continues.

9. Acknowledgements

This project was done in collaboration with Auto 21. More specific researchers from the Université de Sherbrooke include Dr. Francois Michaud and Patrick Frenette

10. References

[1] E. Frazzoli, M.A. Dahleh, and E. Feron, "Robust Hybrid Control for Autonomous Vehicles Motion Planning." *Technical report, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology*, Cambridge, MA, 1999.

[2] A. Wollrath, R. Riggs, and J. Waldo, "A Distributed Object Model for the Java System." *Second USENIX Conference on Object-Oriented Technologies (COOTS)*, Portland, Oregon, 1997.

[3] S. Campadello, O. Koskimies, K. Raatikainen, and H. Helin, "Wireless Java RMI," p. 114, *Fourth International Enterprise Distributed Object Computing Conference (EDOC'00)*, 2000.

[4] D. Payton, "An Architecture for Reflexive Autonomous Vehicle Control," vol 3: p. 1838-1845, *IEEE International Conference on Robotics and Automation. Proceedings*, 1986.

[5] E. Eberbach, C. Duarte, C. Buzzel, and G. Martel, "A Portable Language for Control of Multiple Autonomous Vehicles and Distributed Problem Solving," *Proeedings. Of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems CIRAS'03*, 2003

APPENDIX D: SUPPORTING DATA

D.1 Stationary Tests

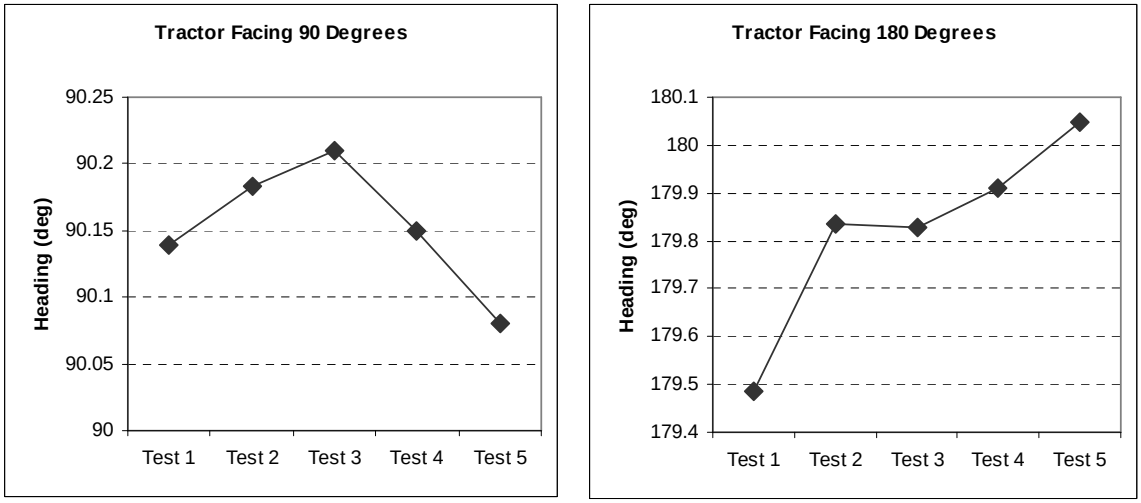

Figure D.1: Average compass heading for each stationary test

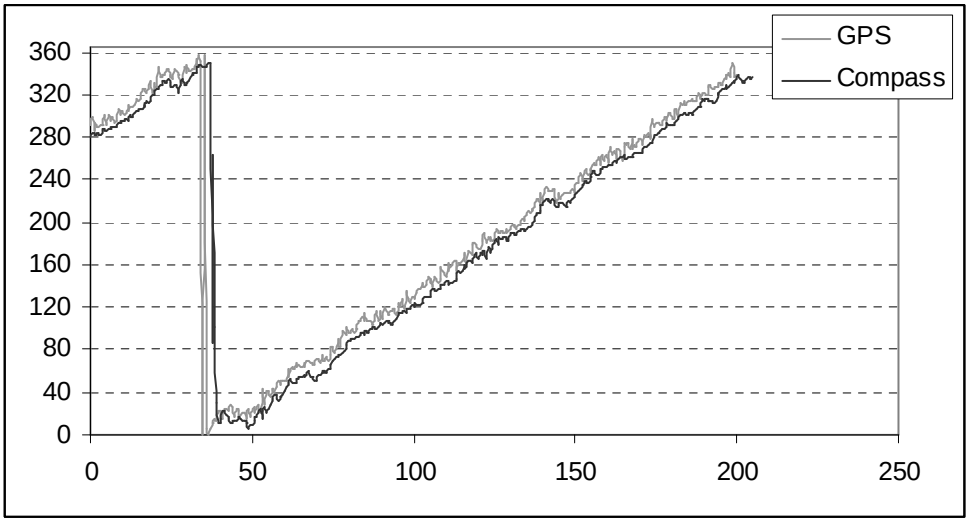

Figure D.2: Circle test showing compass and heading collect data at all headings

D.2 Human Driven Linear Tests

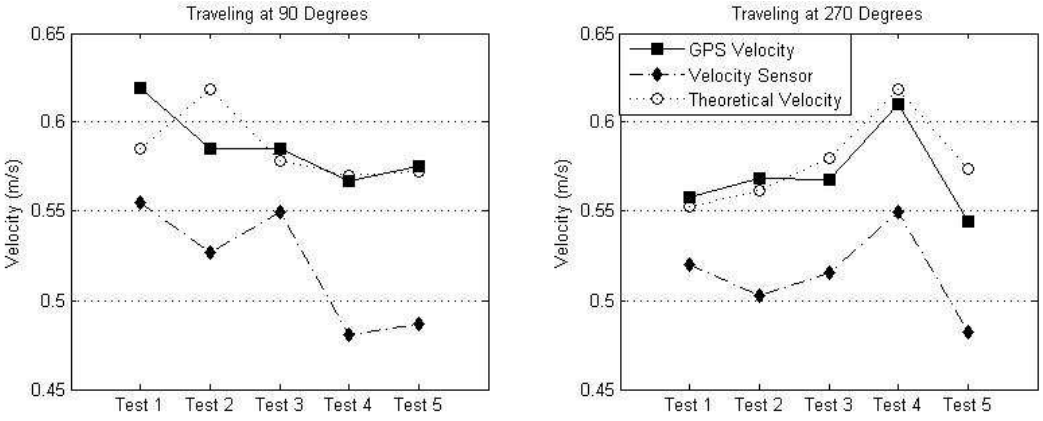

Figure D.3: Average speed when a human was driving the tractor at 0.5m/s

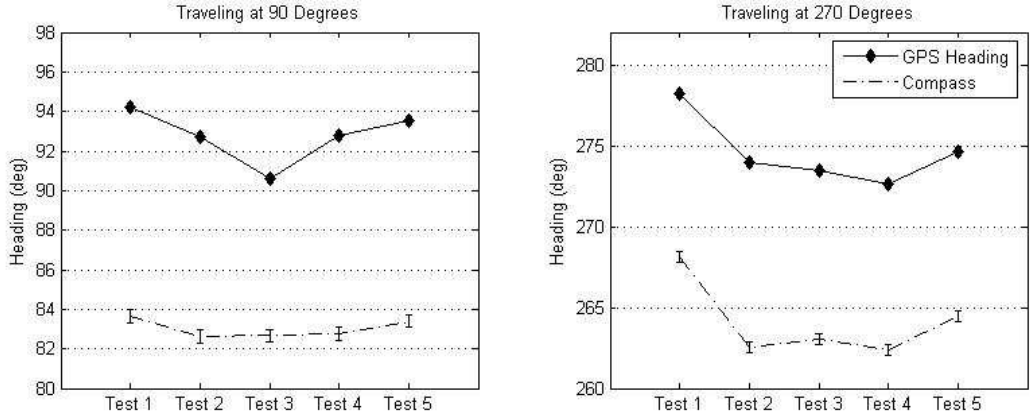

Figure D.4: Average heading for each test when traveling at 0.5m/s being driven by a human

Table D-1: Summary of values from the 0.5m/s human driven linear tests

	Average of all Trials (90˚)	Average of all Trials (270˚)	Standard Deviation of all Trials (90˚)	Standard Deviation of all Trials (270˚)
Compass	83.02	264.11	3.31	3.48
GPS Heading	92.76	274.57	5.36	5.41
Velocity Sensor	0.52	0.51	0.12	0.07
GPS Velocity	0.59	0.57	0.04	0.04
Theoretical Velocity	0.59	0.58		

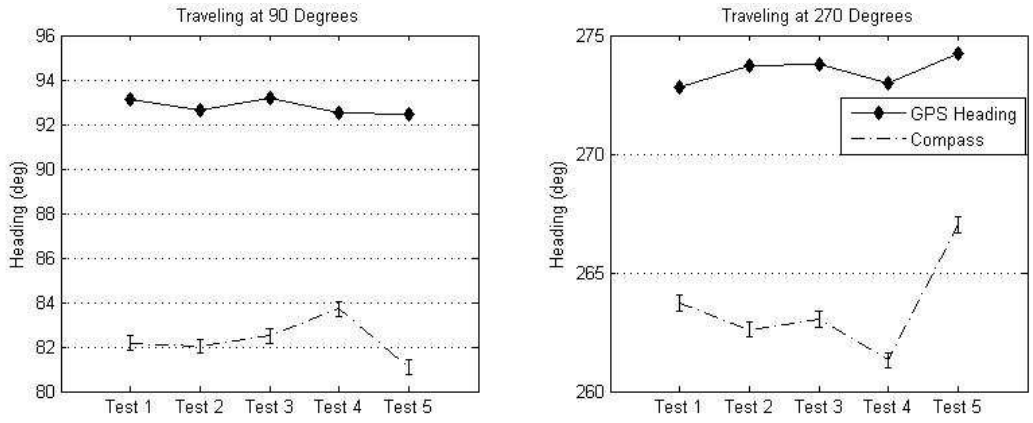

Figure D.5: Average heading when a human was driving the tractor at 1m/s

Table D-2: Summary of values from the 1m/s human driven linear tests

	Average of all Trials (90˚)	Average of all Trials (270˚)	Standard Deviation of all Trials (90˚)	Standard Deviation of all Trials (270˚)
Compass	82.29	263.54	5.06	7.18
GPS Heading	92.78	273.51	4.64	4.76
Velocity Sensor	0.94	0.95	0.13	0.12
GPS Velocity	1.22	1.25	0.05	0.07
Theoretical Velocity	1.21	1.26		

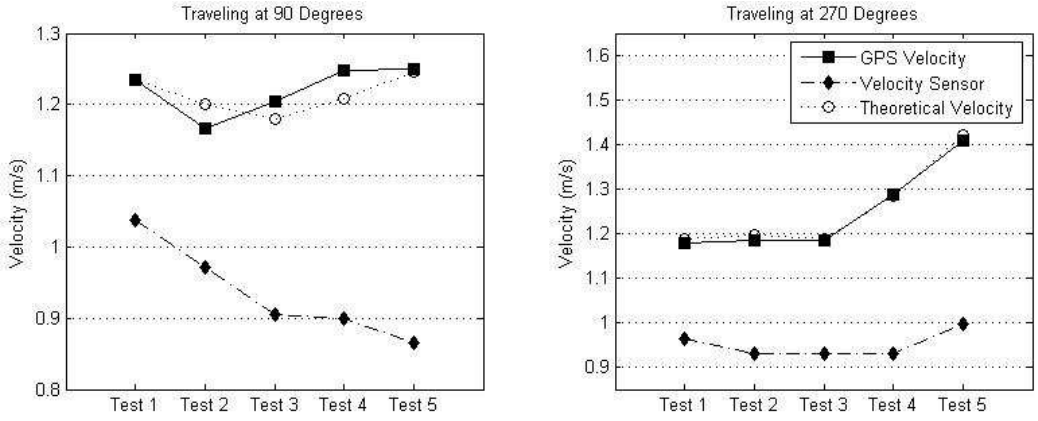

Figure D.6: Average speed when a human is driving the tractor at 1m/s

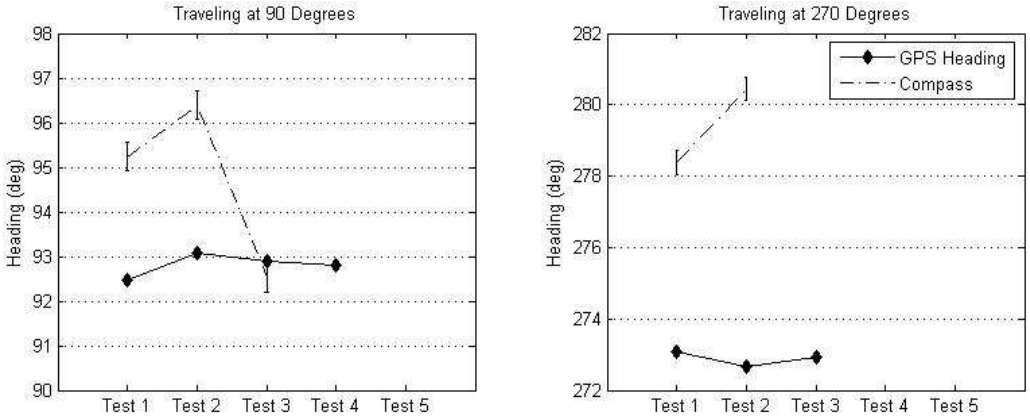

Figure D.7: Average heading when a human was driving the tractor at 1.5 m/s

Table D-3: Summary of values from the 1m/s human driven linear tests

	Average of all Trials (90˚)	Average of all Trials (270˚)	Standard Deviation of all Trials (90˚)	Standard Deviation of all Trials (270˚)
Compass	94.72	264.11	8.76	11.46
GPS Heading	92.81	274.57	4.60	4.72
Velocity Sensor	0.94	0.51	0.21	0.26
GPS Velocity	1.54	0.57	0.11	0.05
Theoretical Velocity	1.66	0.58		

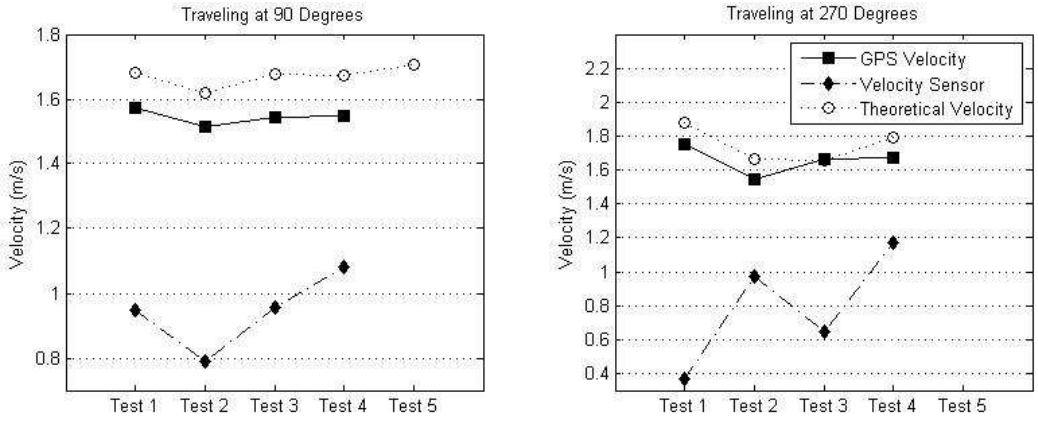

Figure D.8: Average speed when a human is driving the tractor at 1.5 m/s

D.3 Computer Driven Linear Tests

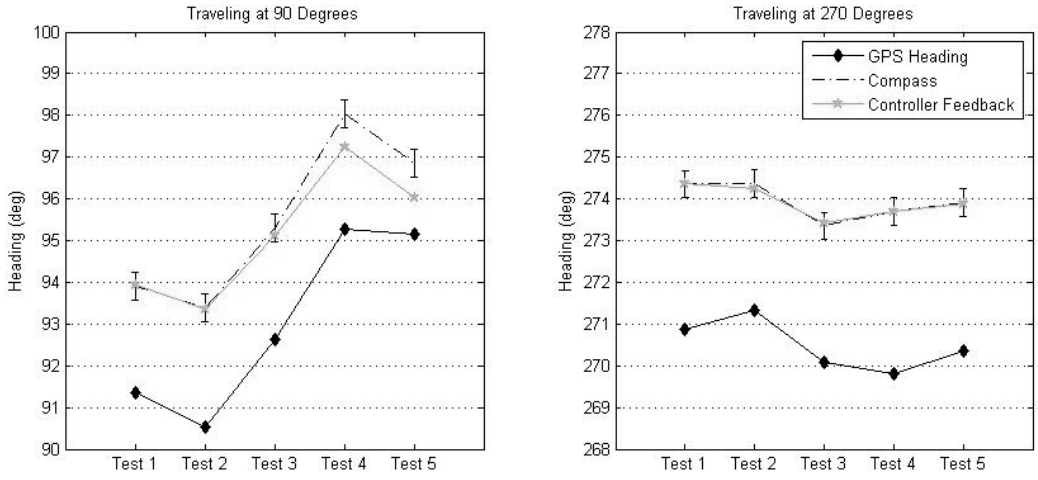

Figure D.9: Average heading when the computer is driving the tractor at 0.5m/s

Table D-4: Summary of data collected during computer driven linear test at 0.5m/s

	Average of all Trials (90˚)	Average of all Trials (270˚)	Standard Deviation of all Trials (90˚)	Standard Deviation of all Trials (270˚)
Compass	95.50	273.93	5.66	3.88
GPS Heading	92.99	270.49	6.11	4.74
Heading Feedback	95.14	273.93	5.37	3.88
Velocity Sensor	0.40	0.42	0.09	0.06
GPS Velocity	0.51	0.51	0.09	0.05
Theoretical Velocity	0.51	0.51		
Velocity Feedback	0.50	0.51	0.10	0.08

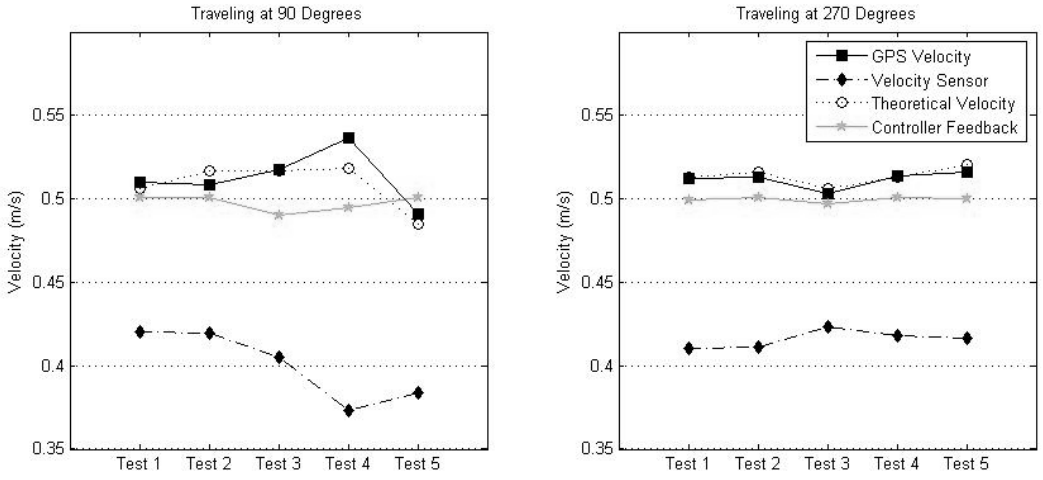

Figure D.10: Average speed when the tractor is driving at 0.5m/s

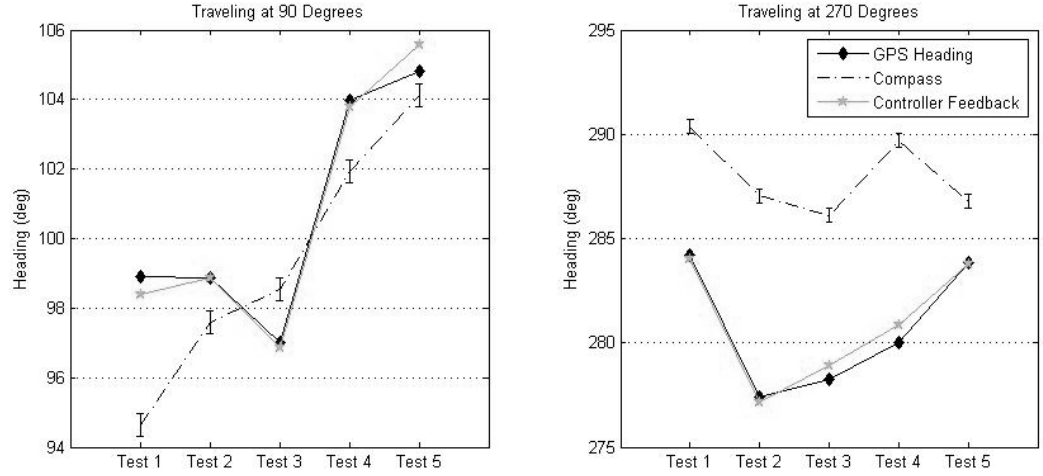

Figure D.11: Average heading when the computer is driving the tractor at 1m/s

Table D-5: Summary of data when the computer is controlling the tractor at 1m/s

	Average of all Trials (90˚)	Average of all Trials (270˚)	Standard Deviation of all Trials (90˚)	Standard Deviation of all Trials (270˚)
Compass	99.37	288.00	9.34	14.24
GPS Heading	100.94	280.73	7.15	8.09
Heading Feedback	101.16	280.95	7.74	8.29
Velocity Sensor	0.42	0.50	0.11	0.06
GPS Velocity	1.03	1.04	0.07	0.07
Theoretical Velocity	1.03	1.04		
Velocity Feedback	0.99	1.00	0.10	0.09

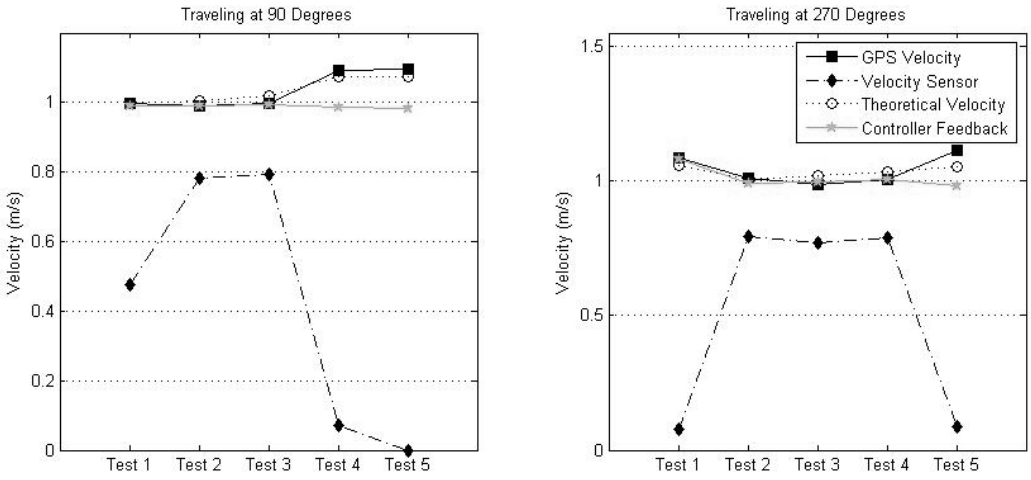

Figure D.12: Average speed when the computer is controlling the tractor at 1m/s

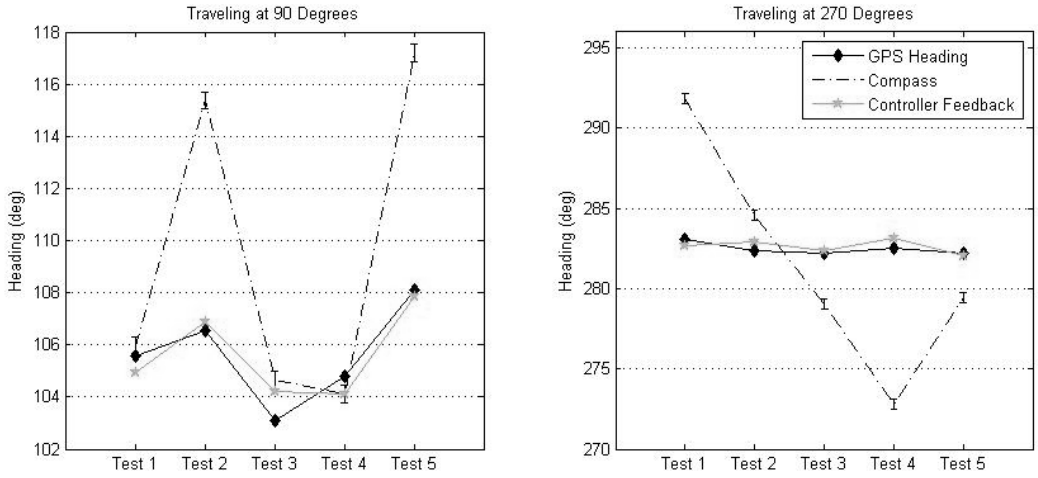

Figure D.13: Average heading when the computer is controlling the tractor at 1.5m/s

Table D-6: Summary of data collected when computer is controlling the tractor at 1.5m/s

	Average of all Trials (90˚)	Average of all Trials (270˚)	Standard Deviation of all Trials (90˚)	Standard Deviation of all Trials (270˚)
Compass	109.46	281.55	22.31	23.42
GPS Heading	105.61	282.47	7.28	7.00
Heading Feedback	105.60	282.63	7.76	6.82
Velocity Sensor	0.13	0.12	0.09	0.06
GPS Velocity	1.62	1.68	0.09	0.07
Theoretical Velocity	1.62	1.66		

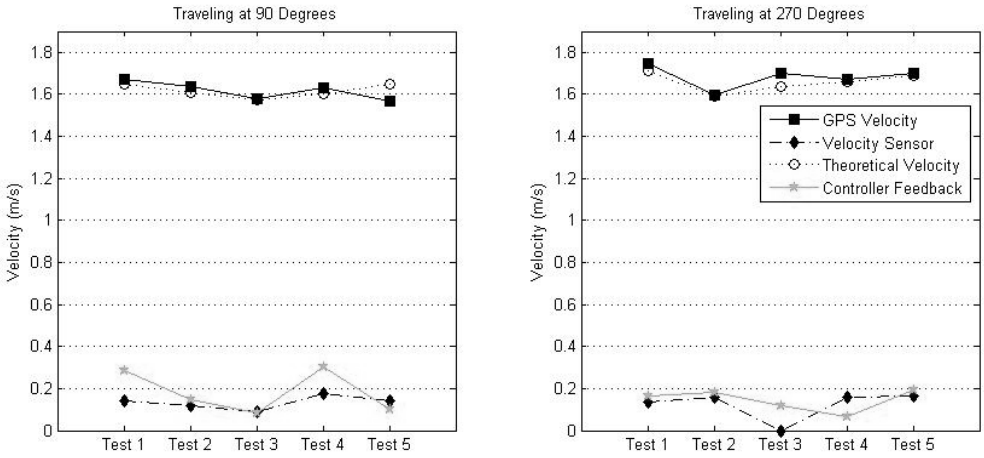

Figure D.14: Average speed when the computer is controlling the tractor at 1.5m/s