CONTROL AND WAYPOINT NAVIGATION OF AN AUTONOMOUS GROUND VEHICLE

A Thesis

by

JAMES PATRICK MASSEY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2006

Major Subject: Mechanical Engineering

CONTROL AND WAYPOINT NAVIGATION OF AN AUTONOMOUS GROUND VEHICLE

A Thesis

by

JAMES PATRICK MASSEY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Make McDermott
Committee Members,	Darbha Swaroop
	Glen Williams
Head of Department,	Dennis O'Neal

May 2006

Major Subject: Mechanical Engineering

ABSTRACT

Control and Waypoint Navigation of an
Autonomous Ground Vehicle. (May 2006)
James Patrick Massey, B.S., Texas A&M University
Chair of Advisory Committee: Dr. Make McDermott

This thesis describes the initial development of the Texas A&M Autonomous Ground Vehicle test platform and waypoint following software, including the associated controller design. The original goal of the team responsible for the development of the vehicle was to enter the DARPA Grand Challenge in October 2005. A 2004 Ford F150 4x4 pickup was chosen as the vehicle platform and was modified with a 6" suspension lift and 35" tires, as well as a commercial drive-by-wire system. The waypoint following software, the design of which is described in this thesis, is written in C and successfully drives the vehicle on a course defined by GPS waypoints at speeds up to 50 mph. It uses various heuristics to determine desired speeds and headings and uses control feedback to guide the vehicle towards these desired states. A vehicle dynamics simulator was also developed for software testing. Ultimately, this software will accept commands from advanced obstacle avoidance software so that the vehicle can navigate in true off-road terrain.

DEDICATION

"Do not let anyone look down on you because you are young, but set an example for the believers in speech, in life, in love, in faith, and in purity." (1 Timothy 4:12)

All thanks to God for His continuing influence in my personal and professional life.

To Bethany

ACKNOWLEDGMENTS

I would like to thank Dr. Make McDermott, Dr. Glen Williams, Dr. Fred Fisher, Aaron Palermo, Craig Odom, Justin Bozalina, and Caleb Wells for help on this project and thesis.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

LIST OF TABLES

INTRODUCTION

There has been an increase in interest in the field of autonomous vehicles during the last few years. Autonomous vehicles eliminate the possibility of losing a conventional vehicle's most precious cargo: its human operator. Specifically, autonomous ground vehicles have direct application in Iraq, where roadside bombs have killed hundreds of troops. In specific instances, such as cargo trucks, the driver can be easily removed providing the technology exists to mimic human reactions.

In response to a Congressional mandate, the DARPA Grand Challenge is a "field test intended to accelerate research and development in autonomous ground vehicles that will help save American lives on the battlefield". [1] More specifically, the Grand Challenge is a grueling 132 mile race across the Californian desert. Once started, the vehicle can only accept GPS location information. All path planning and obstacle avoidance decisions are made by on-board computers. Obstacles include: difficult off-road terrain, natural obstacles such as water, sand, and ditches, and man made obstacles such as tank-traps, concrete posts, and tunnels.

This paper discusses the initial development of a vehicle intended to complete such a course. It begins with an overview of the purchase of a vehicle platform and drive-by-wire actuators and then describes the design of the control and waypoint following software. This software is self contained and guides the vehicle along a set of pre-determined waypoints at speeds up to 50 mph. It is also designed to accept overriding inputs from future obstacle avoidance software, which will give the vehicle a capability to maneuver around a variety of objects. A simulator was also developed, based upon a three-degree of freedom model of the vehicle dynamics. The results of extensive testing of the system are presented and are compared with the predicted path from the simulator.

This thesis follows the style of the Journal of SAE Papers.

VEHICLE SELECTION

The vehicle is the test platform on which all software and hardware are tested. It is also one of the most expensive and important individual purchases required, thus an extensive need analysis was completed. The requirements for the vehicle are based almost solely on the rules and requirements of winning the DARPA Grand Challenge.

VEHICLE FUNCTIONAL REQUIREMENTS

I. Accommodate other components in a "safe" environment
   A. Temperature 40-90 degrees Fahrenheit
   B. Humidity – Non-condensing
   C. Provide power to onboard components
   D. Provide a stable platform for sensors
   E. Provide a "clean" environment
      a. Separate from sand, water and other contaminants
      b. Maintain sensor's external viewing ability

II. Traverse Terrain
   A. Traverse side slope of 30 degrees while maintaining steering control
   B. Traverse ascending or descending slope while maintaining steering control
      a. Hard packed dirt – 40 degrees (approach and departure angles must also meet criteria)
      b. Soft sand – 30 degrees
      c. Mud (less than 1 foot deep) – 30 degrees
   C. Ground Clearance
      a. 16 inches under axle (assuming a solid axle vehicle)
      b. 18-20 inches under frame at mid-point of vehicle
      c. Approach angle of 60 degrees
      d. Departure angle of 40 degrees
      e. Breakover angle of 35 degrees (assuming a 4 wheeled vehicle)
   D. Articulation – One front wheel must be able to rise 12 inches while maintaining ground contact on three other tires

E. Flotation
    a. Minimize ground contact pressure to avoid sinking – Approximately 12 psi
    b. Tread width of at least 10 inches
    c. Tread must not dig in soft sand but be aggressive enough for mud
F. Ford water depth of 24 inches
G. Climb 12 inch step from standstill (no forward momentum)
H. Speed
    a. Maximum speed of at least 45 mph on dirt roads and smooth fields
    b. Average speed of 18 mph
I. Size
    a. 6.5 feet in width (DARPA rules set a 10 foot limit)
    b. 7 feet in height (DARPA rules set a 9 foot limit)
J. Front Impact
    a. Maintain drivability and sensor survival upon contact with a non-movable object at 20 mph

III. Availability
A. Available locally for purchase
B. Maintenance
    a. Local car shops must be able to repair the vehicle
    b. Parts must be available locally

VEHICLE CHOICE

In response to the vehicle functional requirements, it was decided to purchase a new 2004 Ford F150 single cab 4x4 truck. This vehicle can be seen in Figure 1. Partial sponsorship of the project was provided by the College of Engineering with funds from a gift by Ford Motor Company. The truck is a base model except for a few options that were defined as requirements. These include air conditioning, automatic transmission, cruise control, and a V-8 engine.

Figure 1: 2004 Ford F150 single cab 4x4 truck purchased as a platform for the AGV

VEHICLE MODIFICATIONS

While a stock 4x4 truck is very capable and DARPA states that its course can be traversed by such a truck with a human operator, major modifications to the truck were performed to increase its off-road robustness. There is a trade off between the truck's ability to "run over" obstacles and its ability to detect obstacles. Smaller obstacles are more difficult for sensors to detect, thus a more physically robust truck has a greater chance of completing the course.

To increase ground clearance, a 6" Skyjacker brand suspension lift kit was installed on the truck. This kit contains new rear leaf springs, front steering knuckles, an anti-roll bar lowering bracket, front coil-over shocks, and a front skid plate. The contents of the kit are shown in Figure 2. The Skyjacker kit lowers the front A-arm mounting points and replaces the rear leaf spring with a larger one. These suspension modifications allow clearance for 35" Mickey Thompson brand off-road tires, while maintaining factory suspension geometry and ride. [2] The tires are mounted on double-beadlock

wheels that physically clamp the tire to the rim. This permits very low air pressure to be run in the tires without fear of the tire coming off the bead. Lower air pressure allows the tires to conform to obstacles better, providing more traction, as well as damping out high frequency bumps. The wheel is a two-piece design which bolts together in the middle. Figure 3 shows the wheel and tire during assembly before all of the bolts were installed.

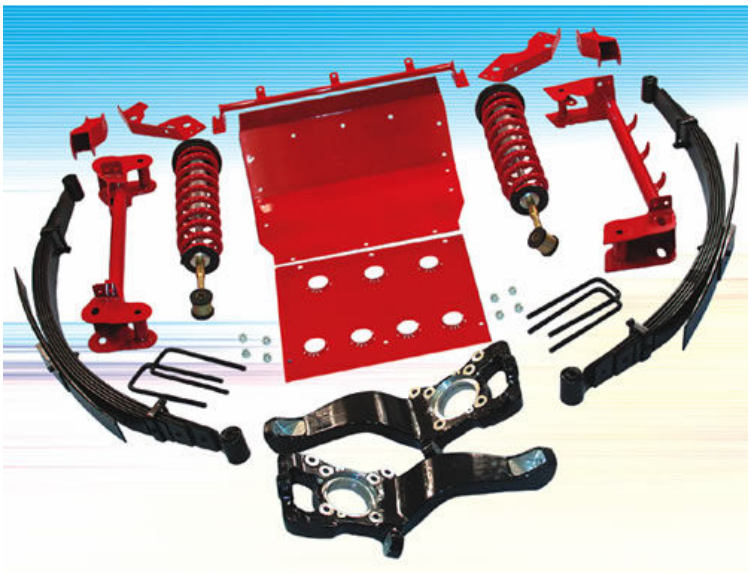

Figure 2: Suspension lift components [2]

Figure 3: Close up photo showing two pieces of the double beadlock wheel

Table 1 quantifies the measurable differences between the stock vehicle and the vehicle after modifications, the result of which can be seen in Figure 4. The stock vehicle clearances were found on the Ford truck website. [3]

Dimension	Before Modification	After Modification
Approach Angle	24 degrees	32 degrees
Departure Angle	26 degrees	31 degrees
Break over Angle	22 degrees	33 degrees
Rear Differential Clearance	8.7 inches	10.5 inches
Front Frame Clearance	7.75 inches	9.75 inches

Table 1: Comparison of suspension clearances between a stock and a modified 2004 Ford 4x4 truck

Figure 4: Ford truck after suspension modifications

DRIVE-BY-WIRE MODIFICATIONS

The next step in creating a test bed for the system required for autonomous operation is to modify the vehicle with servos and actuators that can accept commands from the control software. Primary controls (steering, throttle, and brake) must be commanded electronically. Secondary controls (starting, gear selection, and air conditioning) must also be commanded electronically. Functional requirements were developed to assure that the design or purchase of the modifications would properly serve present as well as future purposes.

DRIVE-BY-WIRE MODIFICATIONS FUNCTIONAL REQUIREMENTS

A. Robustness
    a. Stand alone system, does not depend on development computers used for autonomous functions
    b. Withstand offroad driving conditions
B. Minimal impact on vehicle
    a. Allow manual driving for public roads
    b. Quickly convertible from electronic to manual modes
    c. Preserve manual brake for emergency stops while traveling autonomously
C. Respond to steering control
    a. Interface: 2.5v = straight ahead
       5v = full right lock
       0v = full left lock
    b. Move front wheels to commanded angle at an average steering wheel rate of 2 sec/rev
        i. Constraint: capable of maximum rates of at least 1 sec/rev
        ii. Constraint: over-shoot of less than 0.5 degree
D. Respond to throttle commands
    a. Interface: 2.5v = throttle closed (idle)
       5v = wide open throttle

b. Move throttle to commanded position within 0.5 seconds

E. Respond to brake commands
   a. Interface: 2.5v = no brake application
      0v = max brake pedal force
   b. Apply force to brake pedal to achieve commanded pedal position within 0.5 seconds

F. Respond to shifter commands
   a. Interface: digital signal with distinct values corresponding to PRND21
   b. Move shifter to commanded gear within 3 seconds

G. Respond to key on/off and start commands

EMC PACKAGE OVERVIEW

To satisfy the functional requirements it was decided to purchase the vehicle modification from Electronic Mobility Corporation (EMC). [4] This saved the time that would have been required to design and test our own system. The EMC package provides a robust system that accepts commands from a data acquisition (DAQ) card controlled by the development computer (which runs the waypoint following software) installed in the truck. The EMC computer, mounted in the center of the cab between the two front seats, accepts signals from the development computer and provides primary control using heavy-duty servos connected to the steering column and accelerator/brake. This computer can be seen in Figure 5.

Figure 5: EMC in-vehicle computer

The servo drive connected to the steering column is easily and quickly disconnected by pulling out the safety pin and then pushing in the large yellow knob. The steering wheel can then be used for normal driving. This system is shown in Figure 6. The accelerator and brake are controlled by a single servo as the need will never arise to apply both the brake and accelerator at the same time. The pedals remain intact and can always be used to control vehicle speed manually while sitting in the driver's seat. This servo can be seen in Figure 7.

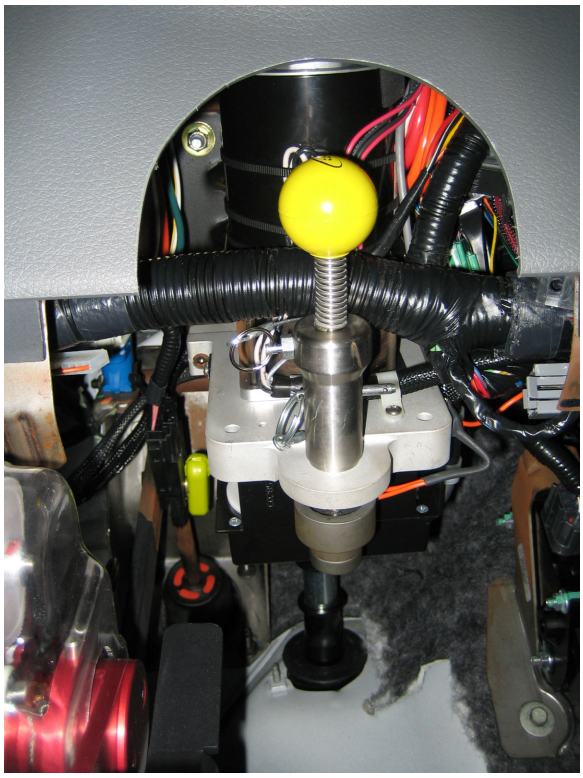

Figure 6: Bottom view of EMC steering servo and disconnect mechanism

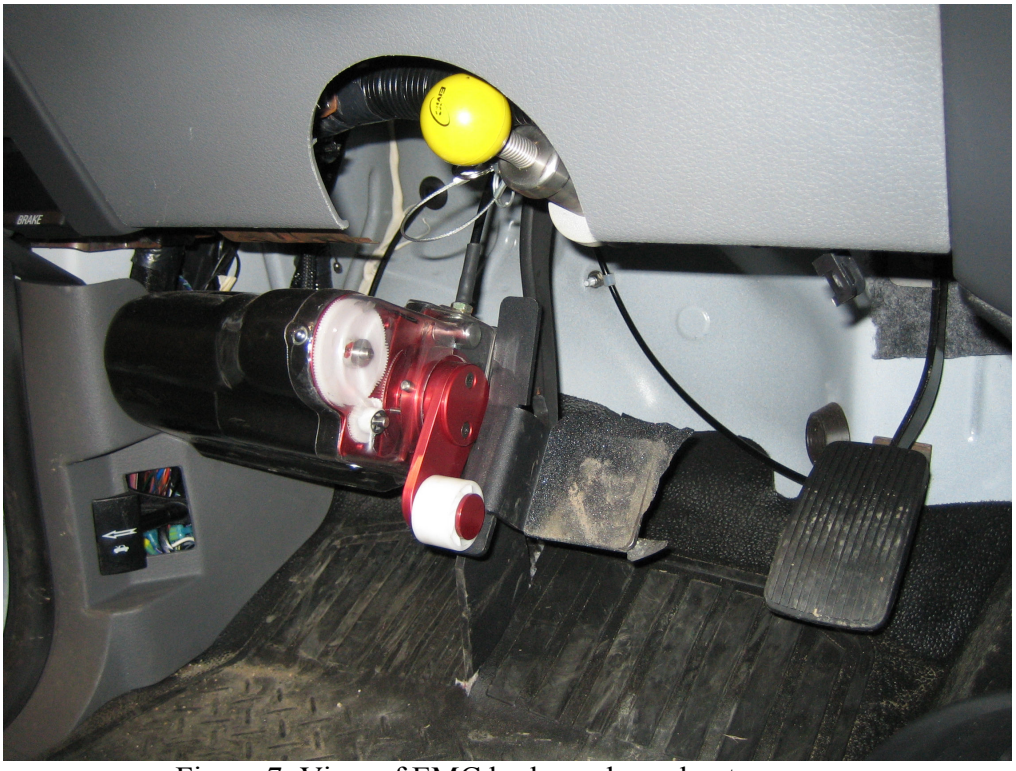

Figure 7: View of EMC brake and accelerator servo

While the steering servo is connected, the analog input into the EMC computer from the development computer can be immediately overridden at any time by using the evaluator hand controls. The evaluator controls consists of a small steering wheel and a joystick for throttle/brake control mounted on a box that sits on the lap of the driver. When the override button is depressed, the driver (evaluator) takes control the vehicle. This proved to be extremely valuable during testing as the vehicle can be easily moved into the start position. Also, if the vehicle is heading in an incorrect path due to a software error, control of the vehicle can easily and safely be regained. One downside to this setup is the fact that the desired steering position sensor is a rotary potentiometer. This means that if the potentiometer is turned to a high steering angle, the vehicle will immediately steer to this angle once the evaluator is activated. Care must be taken to center the evaluator steering wheel prior to testing so that if the evaluator controls are activated the initial steering command will be straight ahead. This eliminates the possibility for dangerous high speed turns upon evaluator activation. The evaluator controls can be seen in Figure 8.

Figure 8: Photo of EMC's evaluator

The secondary functions of the vehicle can also be controlled via ground pulses into the EMC computer. The engine can be started and stopped, the transmission can be shifted into any gear (as long as the brake is depressed), the lights can be turned on, and the windshield wipers can be activated. These functions are also controllable on the EMC control panel that is mounted in reach of the driver, and can be seen in Figure 9.

Figure 9: EMC control panel for secondary functions

VEHICLE DYNAMICS SIMULATION

In order to test the control software, an appropriate model of the truck must be developed to simulate its responses. This model also allows the use of classical control design tools using the associated transfer functions to determine control logic and gains.

If travel on a flat plane is assumed with negligible pitch and roll, the major states of the vehicle can be described by yaw (heading), lateral velocity, and longitudinal velocity. The model was developed in body fixed coordinate using these three degrees of freedom. The body coordinates can then be transformed into earth fixed Cartesian coordinates. The longitudinal equations are assumed to be independent of lateral motion and are only affected by engine and brake forces on the tires.

Integration of the accelerations developed in this section were performed using a 4^{th} order Runge-Kutta method programmed in C. [5] This method was chosen for the accuracy it provides, while not requiring a large amount of calculations.

COORDINATE SYSTEMS

The earth fixed coordinate system follows SAE recommended practice J670e. [6] This system is denoted by OXYZ and is a right-hand orthogonal axis system fixed in the earth. SAE J670e requires that the X and Y axes are horizontal and the Z-axis is pointed down. The origin, O, is at the first waypoint. X is positive to the north, Y is positive to the east, and Z is positive down.

The vehicle fixed coordinate system also follows SAE recommended practice J670e, and is a right handed system denoted by oxyz. The origin o is the current location of the truck in the OXYZ coordinate system. The x-axis is substantially horizontal, points forward, and is in the longitudinal plane of symmetry. The y-axis is nominally horizontal and points to the driver's right. The z-axis is positive down to complete a right hand system.

Heading (yaw) is defined as the angle from the earth fixed X axis to the vehicle fixed x axis, and it represented with ψ. It is zero when the vehicle is pointing due north

and increases clockwise about the Z axis. These coordinate systems are shown in Figure 10.

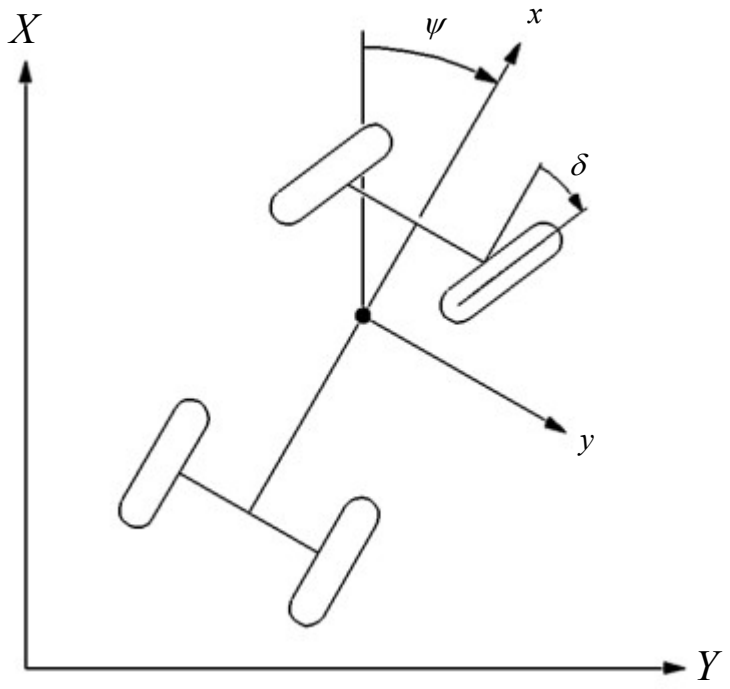

Figure 10: Vehicle coordinate systems [7]

LONGITUDINAL DYNAMICS

To begin the derivation of the longitudinal equation of motion, a free body diagram of the vehicle is shown in Figure 11. This derivation is adapted directly from Gillespie. [7]

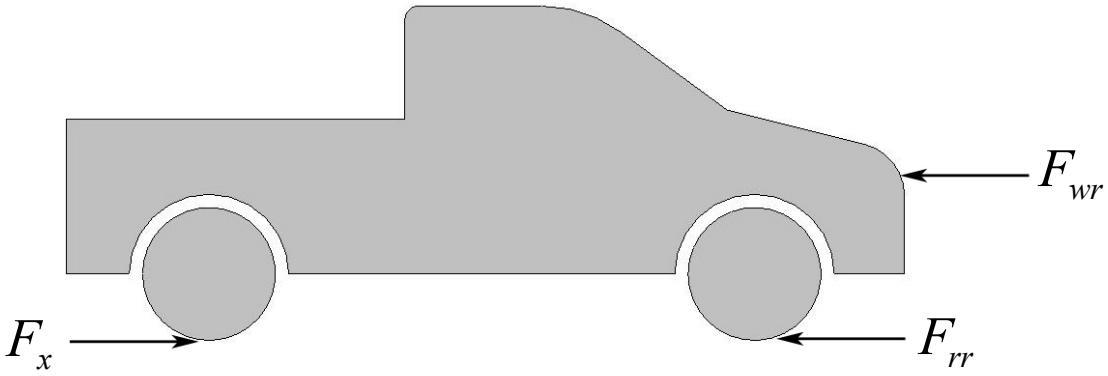

Figure 11: Free body diagram of truck longitudinal forces

F_x = Tractive force at tires (N)

F_{wr} = Force due to wind resistance (N)

F_{rr} = Force due to rolling resistance (N)

Summing forces:

$$ma_x = F_x - F_{wr} - F_{rr} \qquad (1)$$

Determining the engine force at the tires involves simulating the truck's drive-train, as it adds to the equivalent mass of the vehicle and multiplies the engine torque. A simple representative drawing is found in Figure 12.

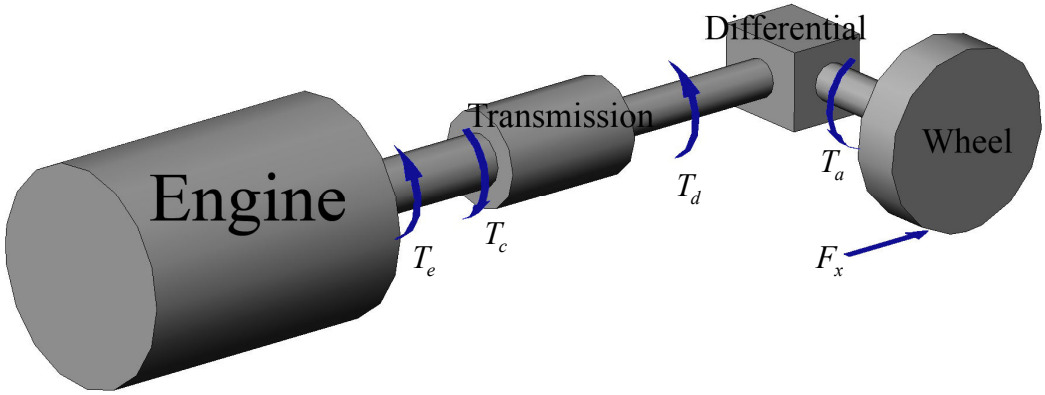

Figure 12: Diagram showing engine drive-train forces

The torque the engine produces is directly applied to the clutch (or torque converter). An actual engine's output torque varies with RPM. As an approximation, the engine torque is based upon the throttle percentage, with maximum throttle corresponding to maximum torque. The engine angular acceleration is governed by the following equation:

$$I_e \alpha_e = T_e - T_c \qquad (2)$$

I_e = Inertia of engine $\left(kg * m^2\right)$

T_e = Torque produced by engine $\left(N * m\right)$

T_c = Torque at clutch (or torque converter) $\left(N * m\right)$

The transmission angular acceleration is governed by a similar equation, with the output reaction torque divided by the transmission gear ratio.

$$I_t \alpha_t = T_c - \frac{T_d}{N_t} \qquad (3)$$

I_t = Inertia of transmission (referred to input shaft) $\left(kg*m^2\right)$

T_d = Transmission output torque $\left(N*m\right)$

N_t = Transmission gear ratio

The driveshaft angular acceleration is determined by the torque at the output of the transmission and the reaction torque at the differential.

$$I_d\alpha_d = T_d - \frac{T_a}{N_f} \tag{4}$$

I_d = Inertia of driveshaft $\left(kg*m^2\right)$

T_a = Torque at axle $\left(N*m\right)$

N_f = Gear ratio of final drive $\left(N*m\right)$

In the last dynamics equation needed to describe the system, the angular acceleration of the wheels is determined by the difference between the axle torque and the torque caused by the tractive force between the tire and the ground.

$$I_w\alpha_w = T_a - F_x r \tag{5}$$

I_w = Inertia of wheel $\left(kg*m^2\right)$

r = Radius of tire $\left(m\right)$

There is assumed to be no slip in the drivetrain and between the tires and the ground. This is true for the gears in the transmission and final drive, but is an approximation for the torque converter and tire-ground interaction.

The resulting kinematic equations are:

$$\begin{aligned}\alpha_t &= \alpha_e \\ \alpha_d &= N_f \alpha_w \\ \alpha_e &= N_t N_f \alpha_w \\ \alpha_w &= \frac{a_x}{r}\end{aligned} \tag{6}$$

Combining equations (1)-(6) yield the following differential equation relating engine torque (T_e) as the input to the tractive force (F_x) as the output. An overall mechanical efficiency of the transmission and final drive (η_{tf}) is added to the equation.

$$F_x = \frac{T_e N_t N_f \eta_{tf}}{r} - \left(\left(I_e + I_t\right)N_t^2 N_f^2 + I_d N_f^2 + I_w\right)\frac{a_x}{r^2} \tag{7}$$

When the brake is applied in the simulation, the tractive force (F_x) of the tires is replaced with the percentage of max braking force corresponding to the desired brake input. Brake ranges from 0 to -1, and thus the force is opposite the tractive force produced by the engine.

Referring to the free body diagram in Figure 11, the force due to rolling resistance is approximated as a constant rolling resistance coefficient multiplied by the weight of the vehicle. Note that the rolling resistance coefficient will vary over a large range (approximately 0.01 to 0.20) depending on the surface upon which the vehicle is operating.

$$F_{rr} = f_r mg \tag{8}$$

Where f_r = rolling resistance coefficient

Force due to air drag is a function of velocity, and it is given by:

$$F_{WR} = \frac{1}{2}\rho V^2 C_d A \tag{9}$$

ρ = density of air $\left(\frac{kg}{m^3}\right)$

V = truck forward velocity $\left(\frac{m}{s}\right)$

C_d = coefficient of drag

A = truck frontal area $\left(m^2\right)$

For anticipated speeds, the aerodynamic drag will be small compared to the maximum available tractive force.

Combining equations (1), (7), (8), and (9), and solving for a_x results in a final equation for longitudinal acceleration, which is shown in equation (10).

$$a_x = \frac{-f_r mg - \frac{1}{2}\rho V^2 C_D A + \frac{T_e N_t N_f \eta_{tf}}{r}}{m + m_I} \tag{10}$$

Where m_I is defined as the equivalent rotating mass of the drivetrain:

$$m_I = \frac{(I_e + I_t)N_t^2 N_f^2 + I_d N_f^2 + I_w}{r^2} \tag{11}$$

LATERAL AND YAW DYNAMICS

The other two degrees of freedom are lateral (vehicle fixed y) and yaw (heading) movement. The equations of motion are derived by assuming the four-wheel vehicle can be described by a bicycle model, which is only affected by the slip angle at the tires. The lateral tire forces at the two axles rotate the vehicle as well as push it sideways. A display of the velocities at the tires is shown in Figure 13. This derivation is adapted directly from Milliken. [8]

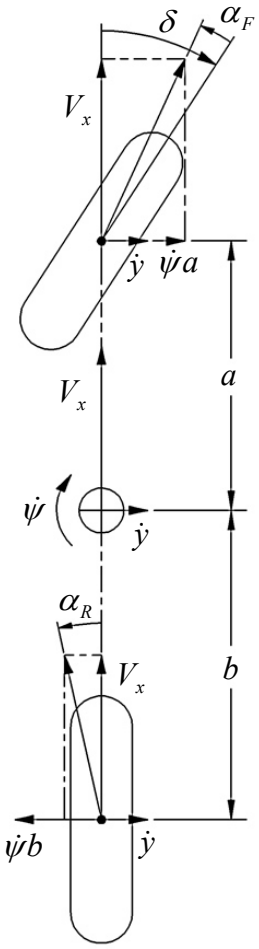

Figure 13: Top view of vehicle bicycle model showing velocities at the tires for a right turn with negative slip angles [8]

The important variables are defined below:

δ = Steering angle (positive CW in top view)

V_x = Forward (longitudinal) speed

α_F = Front tire slip angle

α_R = Rear tire slip angle

a = Distance from CG to front axle

b = Distance from CG to rear axle

$\dot{y}$ = Lateral speed (positive to right)

$\dot{\psi}$ = Yawing speed (positive CW in top view)

The slip angle for an individual tire is defined as the angle from the direction that the wheel is pointing to the direction that the center of the wheel is moving. Slip angles conform to the sign conventions defined for the body fixed coordinate system; clockwise rotation is defined at positive. A positive steering angle produces a right turn, but the slip angles are negative. From the figure, the front and rear slip angles are given in the following equation. Note that the small angle approximation is used.

$$\alpha_F = \tan^{-1}\left(\frac{\dot{y}+\dot{\psi}a}{V_x}\right)-\delta \cong \frac{\dot{y}}{V_x}+\frac{\dot{\psi}a}{V_x}-\delta$$
$$\alpha_R = \tan^{-1}\left(\frac{\dot{y}-\dot{\psi}b}{V_x}\right) \cong \frac{\dot{y}}{V_x}-\frac{\dot{\psi}b}{V_x} \qquad (12)$$

A linear constitutive equation is used for the tires to calculate the lateral force generated by the tires as a function of slip angle. The corresponding forces are the tire cornering stiffnesses multiplied by the slip angle. Notice that the slip angle becomes indeterminant at zero forward speed, resulting in a singularity in the equation. This shows that this model will not be stable at low speeds, and thus can only be used at speeds above 0.5 m/s.

$$F_{yF} = -C_{\alpha F}\alpha_F = -C_{\alpha F}\frac{\dot{y}}{V_x}-C_{\alpha F}\frac{\dot{\psi}a}{V_x}+C_{\alpha F}\delta$$
$$F_{yR} = -C_{\alpha R}\alpha_R = -C_{\alpha R}\frac{\dot{y}}{V_x}+C_{\alpha R}\frac{\dot{\psi}b}{V_x} \qquad (13)$$

Where $C_{\alpha F}$ = Front tire cornering stiffness (always positive)

$C_{\alpha R}$ = Rear tire cornering stiffness (always positive)

The total lateral acceleration is the centripetal acceleration plus the direct lateral acceleration $(\ddot{y})$, as shown below:

$$a_y = \ddot{y} + V_x\dot{\psi} \tag{14}$$

Note that expressing the centripetal acceleration as $V_x\dot{\psi}$ produces the correct sign (direction) for the centripetal acceleration. That is, $\dot{\psi} > 0$ corresponds to a right turn for which the centripetal acceleration is to the right $\left(V_x\dot{\psi} > 0\right)$.

Applying Newton's 2nd law yields:

$$\left(\ddot{y} + V_x\dot{\psi}\right)m = F_F + F_R = -C_F\frac{\dot{y}}{V_x} - C_F\frac{\dot{\psi}a}{V_x} + C_F\delta - C_R\frac{\dot{y}}{V_x} + C_R\frac{\dot{\psi}b}{V_x} \tag{15}$$

The tire forces also produce a moment acting on the vehicle and application of the angular momentum principle yields:

$$I_z\ddot{\psi} = F_Fa - F_Rb = -C_Fa\frac{\dot{y}}{V_x} - C_F\frac{\dot{\psi}a^2}{V_x} + C_Fa\delta + C_Rb\frac{\dot{y}}{V_x} - C_R\frac{\dot{\psi}b^2}{V_x} \tag{16}$$

Rearranging equations (15) and (16) result in the final dynamics equations governing lateral and yaw motion:

$$\ddot{y} = -\dot{y}\left(\frac{C_F + C_R}{mV_x}\right) - \dot{\psi}\left(\frac{C_Fa - C_Rb}{mV_x} - V_x\right) + \frac{C_F}{m}\delta \tag{17}$$

$$\ddot{\psi} = -\dot{y}\left(\frac{C_Fa - C_Rb}{I_ZV_x}\right) - \dot{\psi}\left(\frac{C_Fa^2 + C_Rb^2}{I_zV_x}\right) + \frac{C_Fa}{I_z}\delta \tag{18}$$

Equations (17) and (18) are analogous to a mass/damper system. Thus, the coefficients of $\dot{y}$ and $\dot{\psi}$ are damping coefficients. The presence of V_x in the denominator of these coefficients result in decreased damping with increased speed. The response properties of this system are discussed in Chapter 6 of Milliken [8]. The effect of increasing speed on lateral control performance will be quantified in the following sections of this thesis.

Integration of the accelerations in equations (17) and (18) yield velocities in the body fixed coordinate system. Since the body fixed z axis and the earth fixed Z axis are collinear, the yaw rate $\dot{\psi}$ is the same in both coordinates systems. Integrating $\dot{\psi}$ yields heading(ψ). Using this heading angle, the linear velocities are transformed into the earth fixed coordinate system. Recall that positive X is pointing north and the positive Y is pointing east. The kinematic coordinate transformations are:

$$\begin{aligned} \dot{X}_{earth} &= V_x \cos(\psi) - \dot{y} \sin(\psi) \\ \dot{Y}_{earth} &= V_x \sin(\psi) + \dot{y} \cos(\psi) \end{aligned} \tag{19}$$

Integration of equations (19) and $\dot{\psi}$ yield the position and heading of the vehicle.

PARAMETER IDENTIFICATION

The above differential equations are generic in the sense that they can be used to model any vehicle with the appropriate choice of parameters. The values of the parameters characterizing the AGV must be determined to have an accurate model. Some parameters can be directly measured or calculated from measured values, but most had to be estimated using known information. The accuracy of such parameters is most likely low. The ultimate test will be how well the simulation reproduces the path of the actual truck.

The rotating inertias of the drivetrain and the cornering stiffnesses of the tires were the most difficult to evaluate. The inertias of the drivetrain parts can be estimated by estimating the radius of gyration and measuring or estimating the mass of the part. The radius of gyration can be estimated from the geometry of the item. The cornering stiffnesses of the tires were estimated based on available data for race tires, and then lowered to account for their increase in size. These values were then tuned during the open loop validation until the performance of the vehicle simulation matched the actual vehicle. The parameters and their associated values are listed below in Table 2.

Variable	Description	Value (SI)	Value (ENG)	Source
C_F	Front Tire Cornering Stiffness	55000 $\frac{N}{rad}$	215.8 $\frac{lb}{\deg}$	Estimated
C_R	Rear Tire Cornering Stiffness	40000 $\frac{N}{rad}$	156.9 $\frac{lb}{\deg}$	Estimated
m	Vehicle Mass	2585 kg	5698.9 lb	Ford [3]
r	Tire Radius	0.4445 m	17.5 in	Measured
a	Distance from CG to front axle	1.55 m	5.08 ft	Calculated
b	Distance from CG to rear axle	1.65 m	5.41 ft	Calculated
I_Z	Truck moment of inertia about the z axis	3400 $kg*m^2$	80683 $slug*ft^2$	Estimated using solid model
f_r	Rolling resistance coefficient	0.03	0.03	Estimated
C_D	Air drag coefficient	0.8	0.8	Estimated
A	Frontal Area	5.57 m^2	60 ft^2	Calculated
N_t	Transmission gear ratio	1.9	1.9	Ford [3]
N_f	Differential gear ratio	3.77	3.77	Ford [3]
I_d	Driveshaft inertia	0.15 $kg*m^2$	3.56 $slug*ft^2$	Estimated
I_w	Wheel and axle inertia	16.2 $kg*m^2$	384.43 $slug*ft^2$	Estimated
I_e	Engine inertia	0.56 $kg*m^2$	13.29 $slug*ft^2$	Estimated
I_t	Transmission inertia (referred to input shaft)	0.34 $kg*m^2$	8.06 $slug*ft^2$	Estimated
η_{tf}	Drivetrain efficiency	85 %	85 %	Estimated
$T_{\max}$	Maximum engine torque	397 $N*m$	293 $lb*ft$	Ford [3]
F_{BRAKE}	Maximum tire braking effort	17000 N	3821 lb	Calculated
$\delta_{\max}$	Maximum steering angle	35°	35°	Measured

Table 2: Parameter descriptions and values

MODEL VERIFICATION

To verify that the differential equation model adequately simulates the truck, results from the model must be compared to results from testing the physical truck. At a -30 degree steering angle at a low constant speed, the radius of the circle the truck tracked was measured to be 6.4 m. This matched the kinematic relationship given by the Ackerman steer angle based on low speeds with no slip angle.

$$\sin(\delta) = \frac{L}{R} \tag{20}$$

L = Vehicle wheelbase

R = Circle radius

At a speed of 0.5 m/s, the simulated truck's path is shown in the following figure.

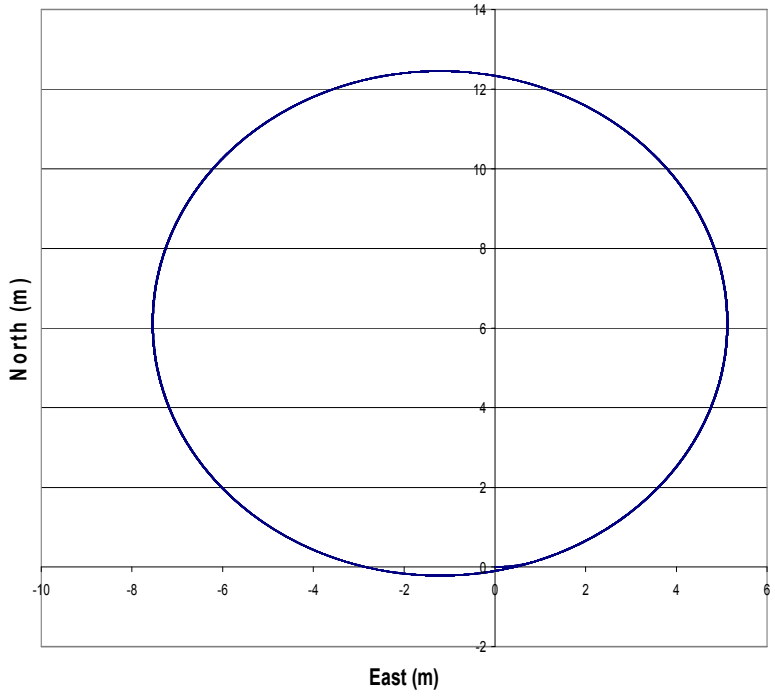

Figure 14: Vehicle simulation using a constant steering angle of -30 degrees

As can be seen, the circle has a radius of 6.3 meters, very close to its kinematically predicted track of 6.4 m. The initial heading of the vehicle is 90 degrees (pointed directly east), but the circle is not perfectly centered on the X (North) axis. This is due to the fact that only longitudinal velocity is given as an initial condition. The vehicle takes a short distance to reach a steady state turning speed and, once it does, it tracks the circle perfectly. Another observation is that higher speeds lead to slightly larger radius circles which demonstrates that the simulation understeers slightly.

To further test the validity of the developed differential equations, the responses of both the model and the actual truck were found for a series of time-based open loop throttle and steering commands. Positive throttle is the percentage of maximum torque the engine can produce. A negative throttle is the percentage of maximum brake force the tires can produce. This is an approximation of the actual torque output of the engine which varies with RPM. Also, the transmission in the simulation does not shift gears. These approximations were proven to be valid by this test. The steering angle is given in degrees, with a positive number representing a right turn. The commands are given in Table 3: **Series of open loop commands for model verification**

Time that command begins (sec)	Throttle/brake position (frac. of max.)	Steering angle (degrees)
0	0.18	0
5	0.11	-30
10	0.11	0
15	0.06	25
20	0.06	0
25	0.08	15
30	0.03	0
35	-0.4	0

Table 3: Series of open loop commands for model verification

The results of commands are displayed together in Figure 15:

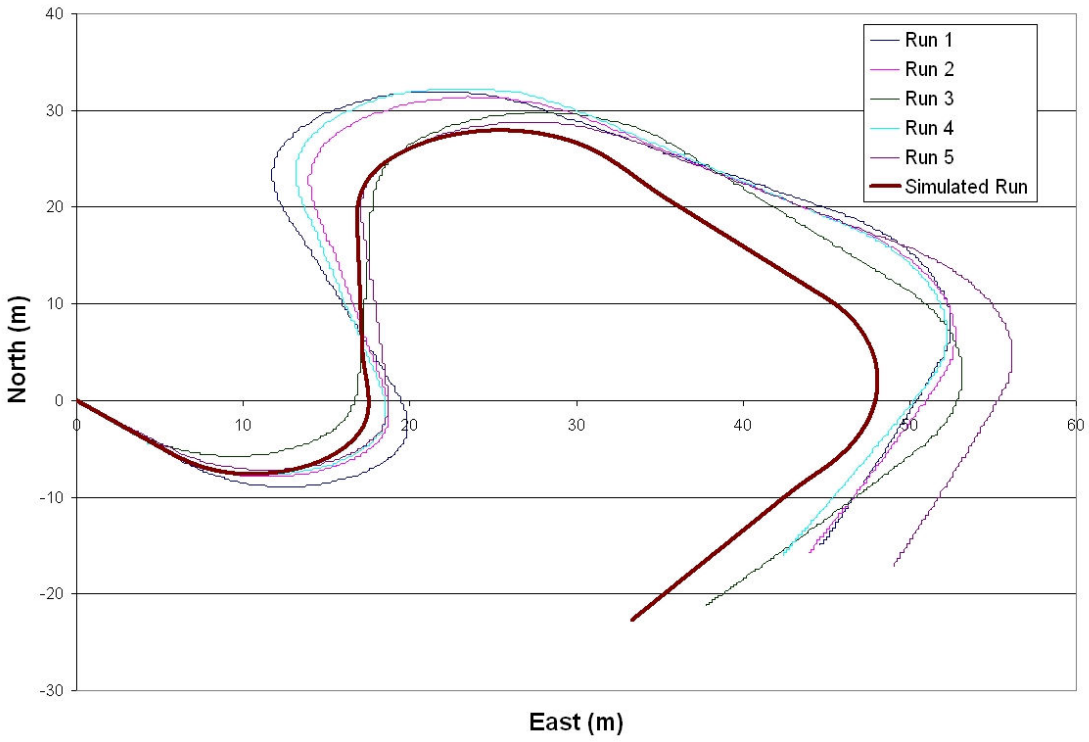

Figure 15: Comparison of the path of the simulated versus the actual path of the vehicle in response to a series of open loop commands

Five different runs were completed in the vehicle, and results from all five runs and the simulation are plotted. As can been seen from Figure 15, the paths of the computer based model and the actual truck are similar; however, small heading errors propagate to large displacement errors over time and the paths deviate further as the vehicle travels. The velocity of the vehicle plays a large part in its path and is displayed in Figure 16. If the simulation is traveling even slightly slower than the actual truck in a turn, the vehicle's heading upon turn exit can be very different. This can be seen in the first turn in Figure 15. Though the end positions differ by several meters, the model is valid for use in closed loop simulations.

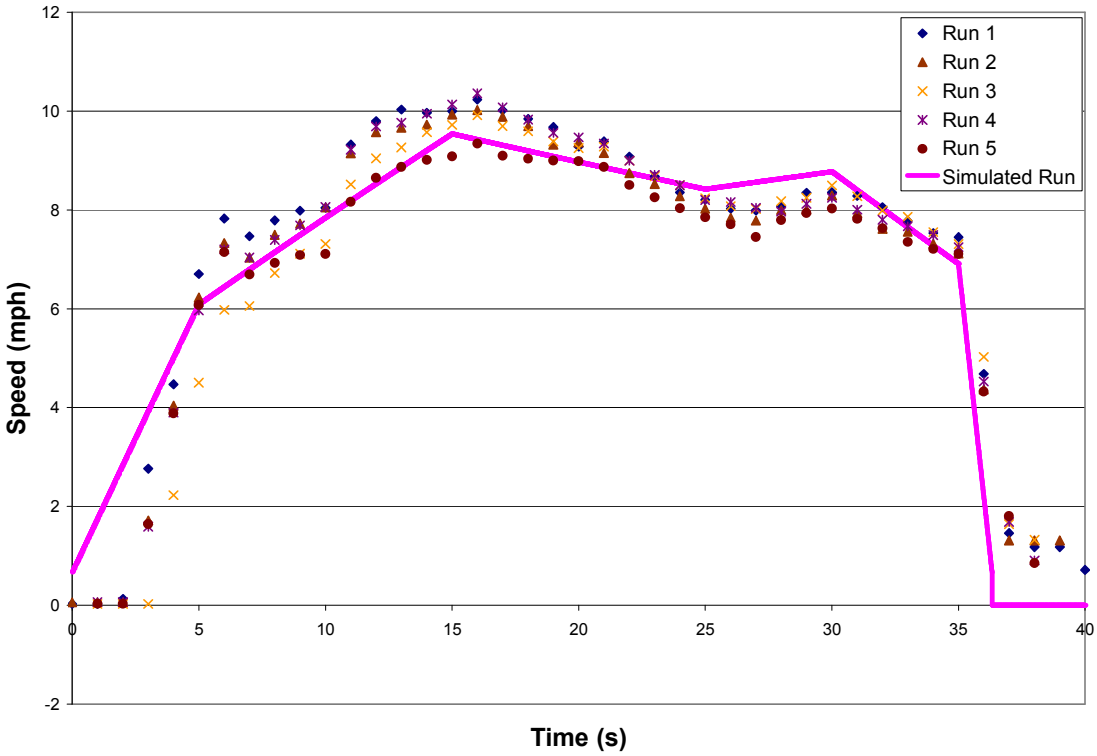

Figure 16: Comparison of the simulated versus actual speed of the vehicle in response to a series of open loop commands

The logging software recorded the vehicle position and speed every second. All points are shown in Figure 16. The apparent delay at the start of the run is due to the fact that the GPS calculates speed and heading by using back differences in position with inherent errors at low speeds. The delay in the servo is less than a second, and is modeled as a first order delay. The model is given an initial velocity of 0.5 m/s to avoid instability. The steering reaction is also not immediate, so it is modeled using a max turn rate of 18 degrees per second, which was determined based on measurements on the truck.

Based on the above results for open loop commands it was decided that the model was adequate for controller design and simulator testing of other vehicle systems, such as obstacle detection and avoidance.

CONTROLLER DESIGN

The main functions of the controller are to direct the truck to a dynamically changing desired heading and desired velocity with emphasis on a well damped response. The 'Waypoint Following' section of this thesis deals with determining these desired states. Three control loops are to be used to control heading error, path error, and velocity error. Path error is defined as the perpendicular displacement of the vehicle from the straight line between waypoints. It is described in depth on page 50. Heading and path error are feedback signals to the steering angle controller and velocity error is the feedback signal to the throttle/brake position controller. A combination of heading and path control is defined in equation (47). To greatly simplify the system the heading and path controller are designed separately. The heading error will be the dominating feedback. The control loops are stable separately and the simulation indicates that they are stable together.

To begin the design of a suitable controller, a transfer function is determined from the differential equations of the vehicle. The lateral and yaw equations are equations (17) and (18), and the longitudinal equation is equation (10).

LATERAL AND YAW DYNAMICS – HEADING CONTROL

For heading control the objective is to move along a desired heading. The control variable is steering and the output variable is heading (yaw), which is controlled to steer toward a waypoint. To simplify the algebra, the group of constants multiplied to each state variable are represented by a single placeholder constant.

$$\begin{aligned} \ddot{y} &= -\dot{y}A - \dot{\psi}B + \delta C \\ \ddot{\psi} &= -\dot{y}D - \dot{\psi}E + \delta F \end{aligned} \tag{21}$$

Where the constants are defined by:

$$A = \frac{C_F + C_R}{mV_x}$$

$$B = \frac{C_F a - C_R b}{mV_x} - V_x$$

$$C = \frac{C_F}{m}$$

$$D = \frac{C_F a - C_R b}{I_Z V_x}$$

$$E = \frac{C_F a^2 + C_R b^2}{I_z V_x}$$

$$F = \frac{C_F a}{I_z}$$

Note that the coefficients A, C, E, and F are always positive. Coefficient D and the first term in B have the same sign as the understeer gradient, which is positive for this understeering vehicle. For a neutral steer vehicle D = 0, and the equations are decoupled.

Taking the Laplace transform and solving for the open loop transfer function from steering angle (δ) to heading angle (ψ) yields:

$$\frac{\psi(s)}{\delta(s)} = \frac{Fs + FA - CD}{s^3 + s^2(A+E) + s(AE - BD)} \tag{22}$$

A PD control logic is employed:

$$\frac{\delta(s)}{e(s)} = K_{PH} + K_{DH}s \tag{23}$$

Where $e(s)$ is the error between the heading (ψ) and the desired heading (ψ_d).

The corresponding closed loop transfer function is:

$$\frac{\psi(s)}{\psi_d(s)} = \frac{FK_{DH}s^2 + (K_{DH}FA - CDK_{DH} + K_{PH}F)s + (K_{PH}FA - K_{PH}CD)}{s^3 + (A + E + K_{DH}F)s^2 + (AE - BD + K_{DH}FA - K_{DH}CD + K_{PH}F)s + K_{PH}FA - K_{PH}CD} \quad (24)$$

Figures 16 and 17 show the regions of the stable gain values of the closed loop system at speeds of 0.5 m/s and 23 m/s, based on Routh's stability criteria. [9] Using Matlab, K_{DH} and K_{PH} were varied independently, and if the system was stable for a K_{DH} - K_{PH} pair, a green dot was placed on the graph at the corresponding point. If the system was unstable, no dot was placed.

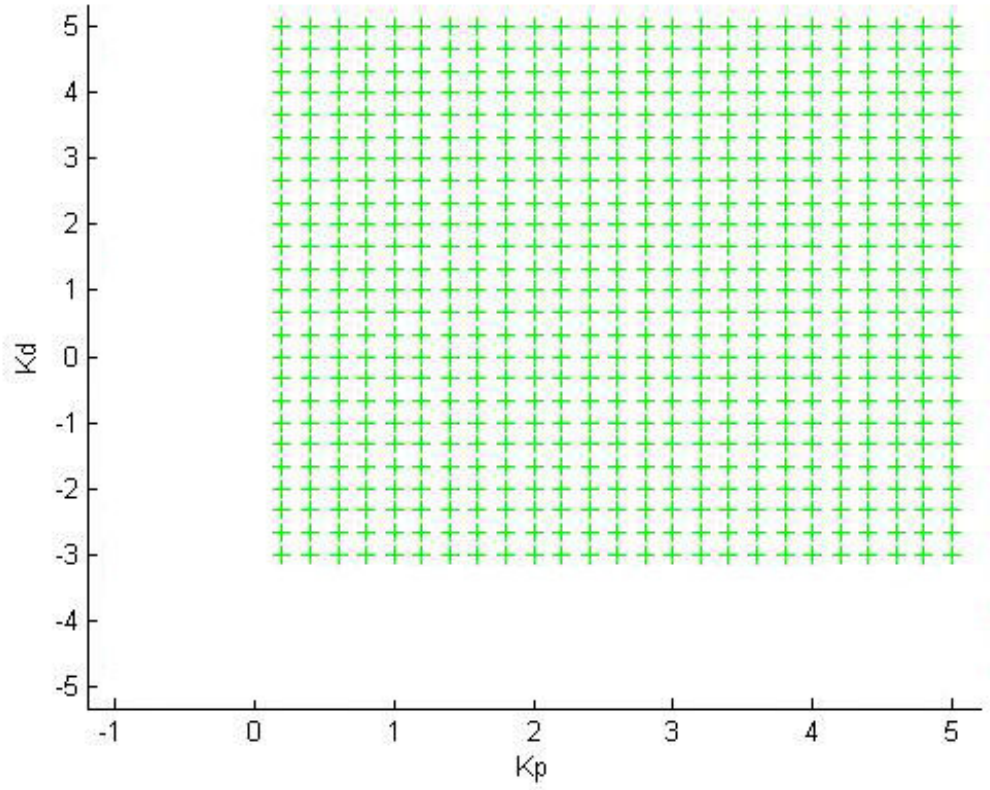

Figure 17: Stable gain values for a control system tracking a constant heading at a velocity of 0.5 m/s

At very low velocities (0.5 m/s), all positive values for both gains are stable, and negative K_{DH} gains greater than -3 are stable. Notice that K_{DH} values of zero are stable

but K_{PH} values of zero are not stable. Figure 18 shows the same stability criteria at a speed of 23 m/s (51.4 mph), which is slightly above the highest DARPA allowed speed.

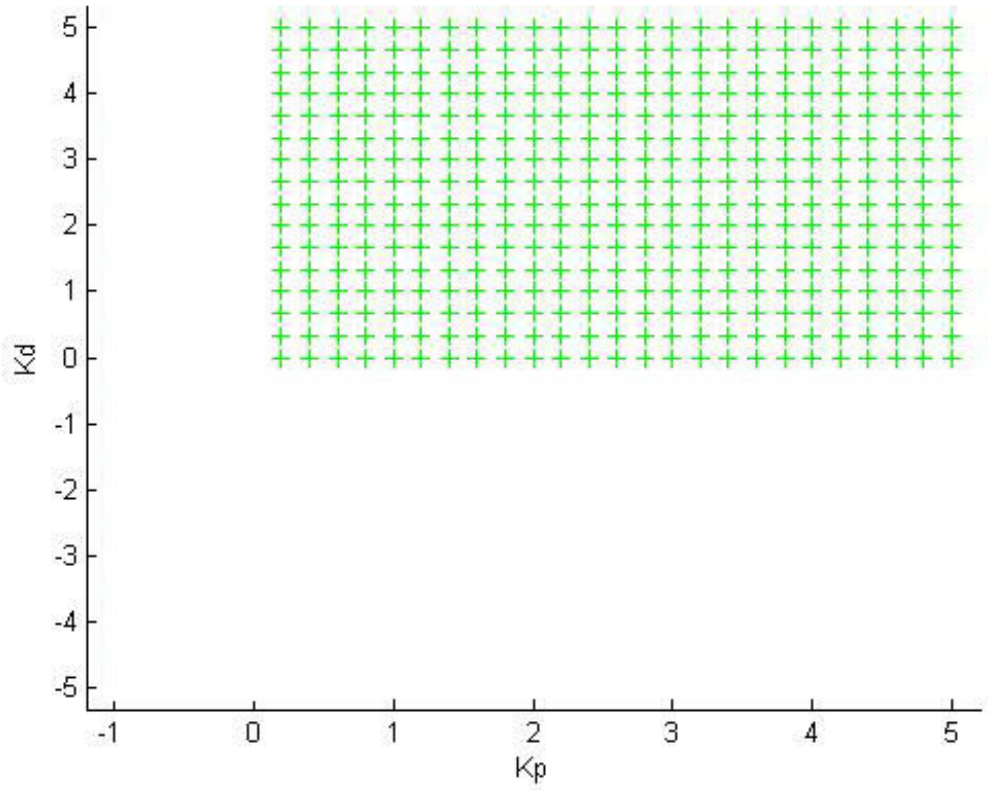

Figure 18: Stable gain values for a control system tracking a constant heading at a velocity of 23 m/s

As expected, the negative K_{DH} gains are now unstable but all positive values are still stable. This leads to the conclusion that stability is not an issue in this system, so other methods must be used to find suitable control gains. Digital control systems provide non-continuous feedback that results from implementing control logic on a digital computer. In this case, the GPS system outputs the states of the vehicle at a rate of 20 Hz, which limits the rate at which the controller can update the feedback. To easily convert the continuous system to discrete, the capabilities of Matlab are employed. Using a bilinear transformation at a sampling period of 0.05 seconds, a velocity of 23 m/s, and

gains of $K_{PH} = 1$ and $K_{DH} = 0$, along with vehicle parameters from Table 2, the resulting closed loop transfer function is:

$$\frac{0.01222z^3 + 0.01304z^2 - 0.01058z - 0.0114}{z^3 - 2.796z^2 + 2.604z - 0.8084} \tag{25}$$

The corresponding digital root locus, where the proportional gain is varied and the derivative gain is zero, is shown in the following figure.

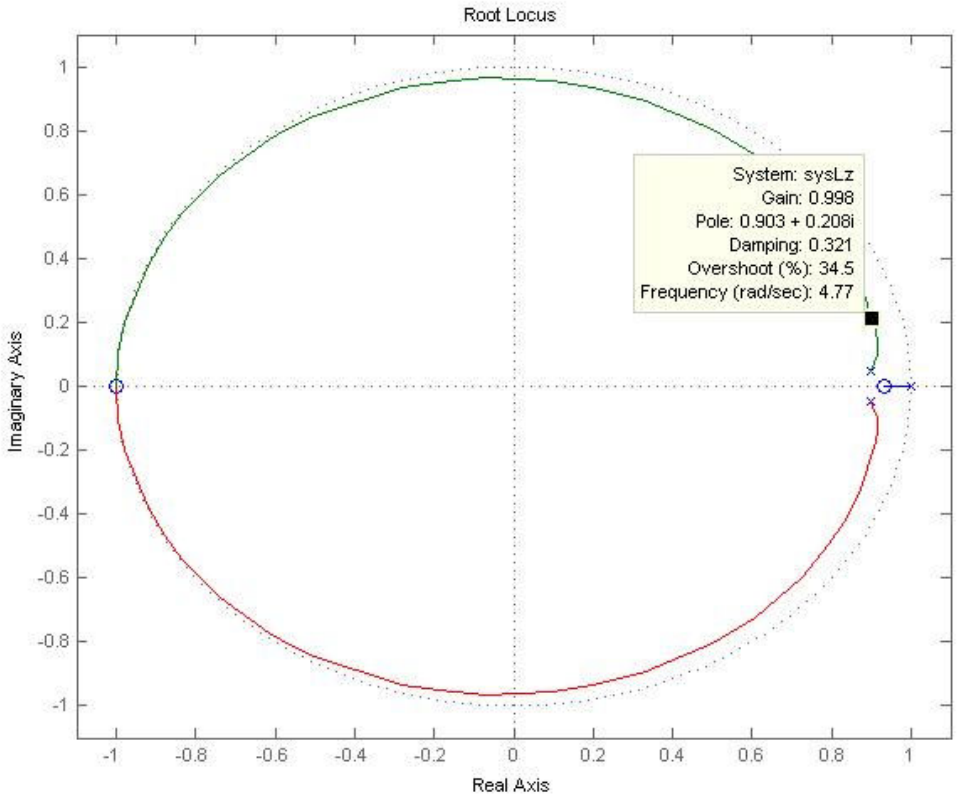

Figure 19: Digital root locus for $K_{DH} = 0$ and a velocity of 23 m/s

As expected, there is no proportional gain that will drive the system unstable. The root locus is contained entirely within the unit circle. The overshoot, however, increases dramatically with the gain. The unity gain results in an overshoot of 35%, and a gain of 5

results in a 65% overshoot. Obviously, this overshoot also varies with speed. The following figure shows five root loci created for decreasing speeds.

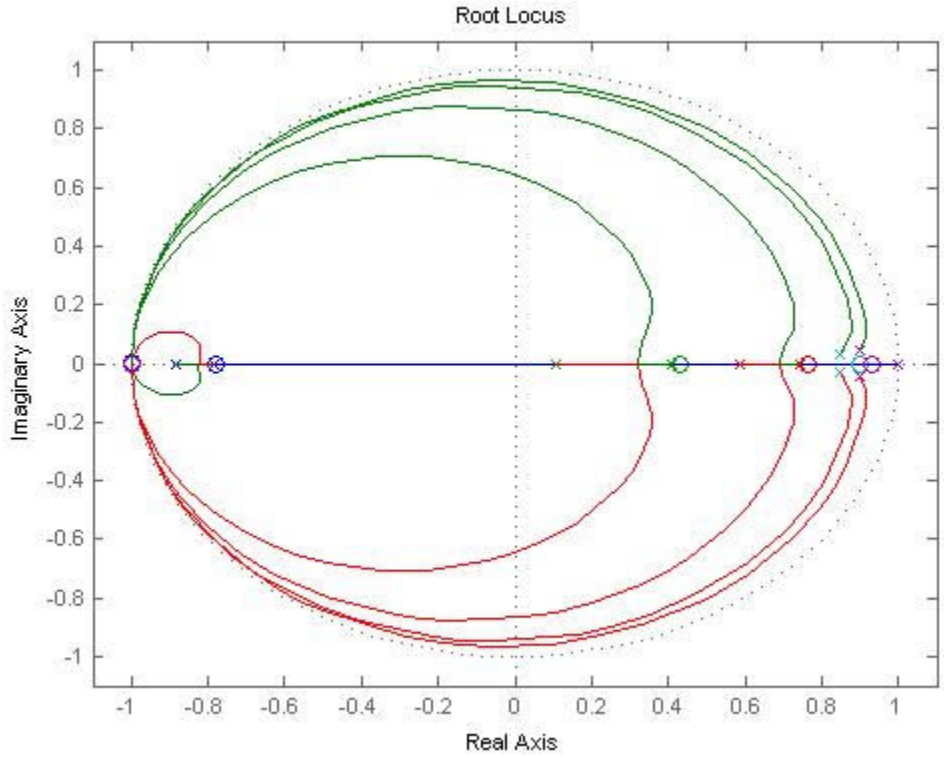

Figure 20: Digital root loci for $K_{DH} = 0$ and velocities of 23 m/s, 15 m/s, 6 m/s, 2 m/s and 0.5 m/s, using proportional gains only

In this graph, the circles enlarge with increasing velocity. Damping increases substantially, overshoot decreases, and natural frequency increases as speed decreases. All these serve to improve the performance at lower speeds. Much higher gains can be used at lower speeds to further increase performance.

Adding a derivative gain to the system serves to reduce overshoot for higher speeds, while not affecting the natural frequency very much. Figure 21 shows how the root's path is 'pulled' inward. Here a ratio of $K_{DH} / K_{PH} = 0.5$ is used, and the overall gain is varied to produce the root locus. Notice that for the lower velocities (<5 m/s), the root locus is not affected much by the addition of the derivative gain. After some

experimentation, was found that a constant K_{DH} gain of 0.5 reduces the overshoot at all velocities and is the value chosen for the preliminary design.

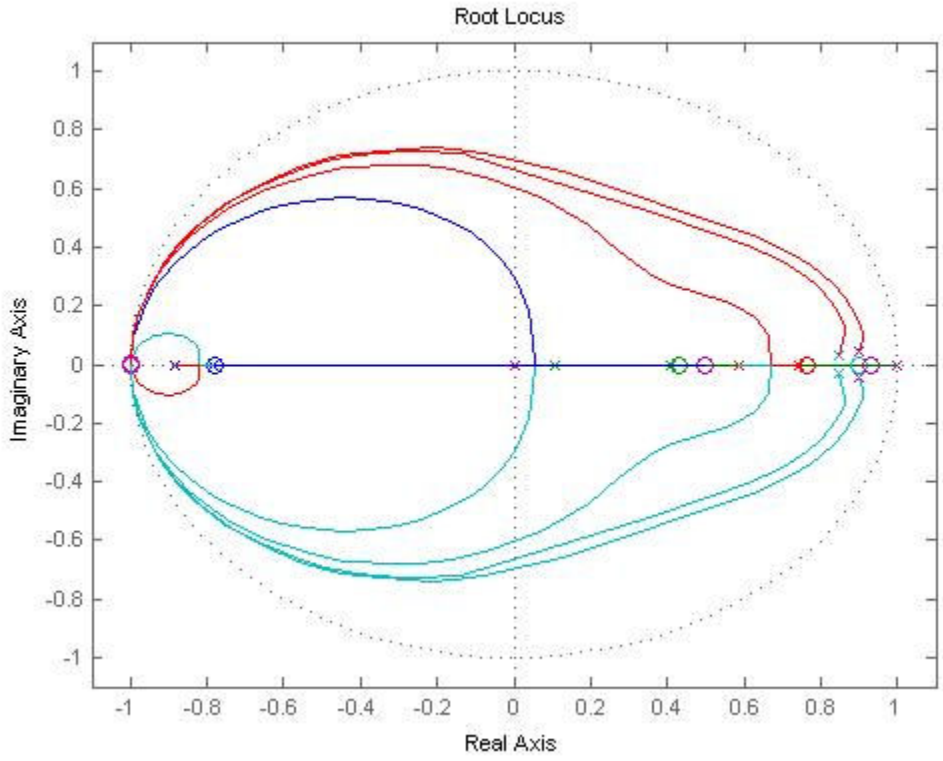

Figure 21: Digital root loci for velocities of 23 m/s, 15 m/s, 6 m/s, 2 m/s and 0.5 m/s at a feedback rate of 20 Hz, using proportional and derivative gains

To take advantage of the increased stability at lower velocities, the proportional heading gain has been programmed as a function of velocity. Values were chosen to keep the overshoots close to zero, while maintaining a fast rise time. An algebraic function was fit to the values of gain vs. speed, so that a gain for any speed could easily be found. The extreme values of the function are limited as they approach infinity and zero, at low and high speeds respectively. This curve fit is:

$$K_{PH} = 5 * V_x^{-0.8} \qquad 0.4 < K_{PH} < 5 \qquad (26)$$

This equation is shown in graphical form in Figure 22, which illustrates the increase in the gain as velocity decreases.

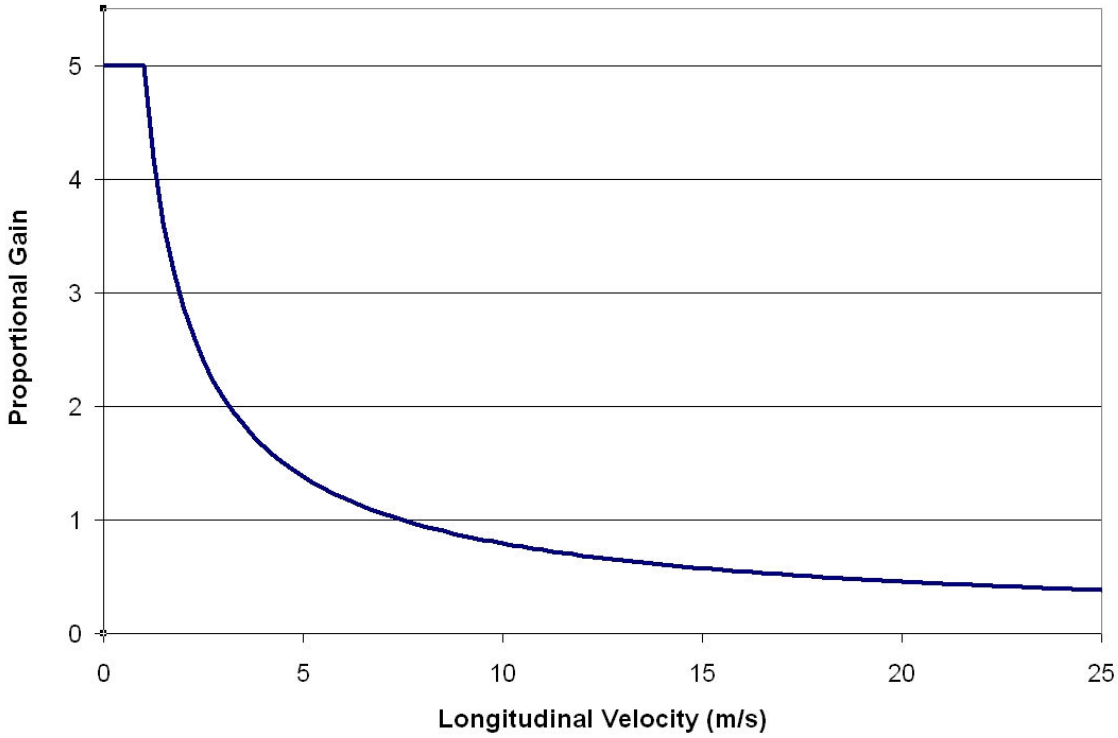

Figure 22: Relationship between the longitudinal proportional gain and velocity

LATERAL AND YAW DYNAMICS – PATH CONTROL

Another control approach that is useful is path control; i.e., to minimize the lateral displacement of the vehicle from the straight line path between two waypoints. The control algorithm uses lateral displacement from the desired path to guide the vehicle back to the path, where the terrain is assumed to be the most desirable. The calculation of the feedback is described more in depth in the 'Waypoint Following' section.

To make this setup a normal LTI system, a few changes must be made. The straight path between waypoints has a global heading angle. The path is rotated by this angle, so that it is parallel with the X (north) earth fixed axis. It is then translated so that the previous waypoint is at the origin. This places the path directly on the earth fixed X axis, so that the earth fixed Y displacement is the path error. To linearize the system, the

heading angle is assumed to be small. Starting with the coordinate transformation in equation (19) to solve for the Y velocity (perpendicular to the X earth fixed axis):

$$\dot{Y}_{earth} = V_x \sin(\psi) + \dot{y}\cos(\psi) \tag{27}$$

If ψ is assumed to be small, then the equation (27) reduces to the following equation, where $\dot{y}$ is the lateral velocity and V_x is the longitudinal velocity:

$$\dot{Y}_{earth} = V_x\psi + \dot{y} = \dot{e} \tag{28}$$

The open loop transfer function that is needed uses steering angle as an input and perpendicular displacement in the Y direction (e) as an output. Thus a relationship between lateral displacement (y) and steering angle (δ) and between yaw angle (ψ) and steering angle (δ) is needed to make equation (28) into the required transfer function. The transfer function between yaw angle and steering angle was found in equation (22), and is rewritten here using the same placeholder variable used before:

$$\psi(s) = \frac{Fs + FA - CD}{s^3 + s^2(A+E) + s(AE - BD)}\delta(s) \tag{29}$$

Similarly, the transfer function between lateral displacement and steering angle is found by solving equations (21) for the lateral displacement (y) instead of yaw angle, and is:

$$y(s) = \frac{Cs + CE - BFD}{s^3 + s^2(A+E) + s(EA - BD)}\delta(s) \tag{30}$$

The Laplace transform of equation (28) contains both $\psi(s)$ and $y(s)$:

$$sY(s)_{earth} = V_x\psi(s) + sy(s) \tag{31}$$

Solving equations (29-31) for the transfer function $\frac{Y_{earth}}{\delta}$ yields:

$$\frac{Y_{earth}}{\delta} = \frac{s^2C + s\left(V_xF + CE - BFD\right) + V_x\left(FA - CD\right)}{s^4 + s^3\left(A + E\right) + s^2\left(AE - BD\right)} \tag{32}$$

As before, the closed loop transfer function when combined with a PD controller is found, and is given below:

$$\frac{s^3CK_{DP} + s^2\left(\left(V_xF + CE - BFD\right)K_{DP} + CK_{PP}\right) + s\left(\left(FA - CD\right)V_xK_{DP} + \left(V_xF + CE - BFD\right)K_{PP}\right) + \left(FA - CD\right)V_xK_{PP}}{s^4 + s^3\left(A + E + CK_{DP}\right) + s^2\left(AE - BD + \left(CE + V_xF - BFD\right)K_{DP} + CK_{PP}\right) + s\left(\left(CE - BFZ + V_xF\right)K_{PP} + \left(FA - CD\right)V_xK_{DP}\right) + \left(FA - CD\right)V_xK_{PP}} \tag{33}$$

The characteristic equation from this transfer function is used in Matlab to map the stable values of K_{PP} and K_{DP}. For a velocity of 23 m/s, the stable proportional and derivative gains are shown in Figure 23.

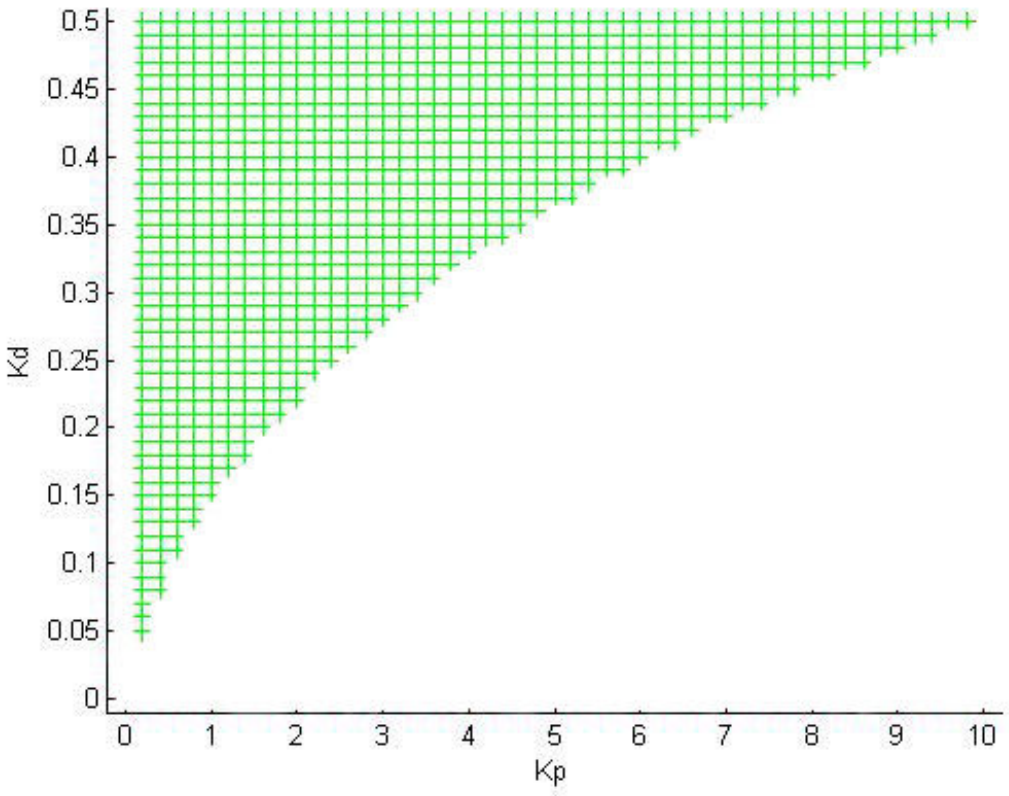

Figure 23: Graph showing stable gain values for a control system tracking a path at a velocity of 23 m/s

As can be seen from the graph, guiding a vehicle along a path is inherently more unstable than tracking a constant heading. At this speed, proportional gain is not enough; a small derivative gain is required to stabilize the system. As expected, at slower speeds the stable region is much larger. Figure 24 shows the stable gains at a speed of 10 m/s.

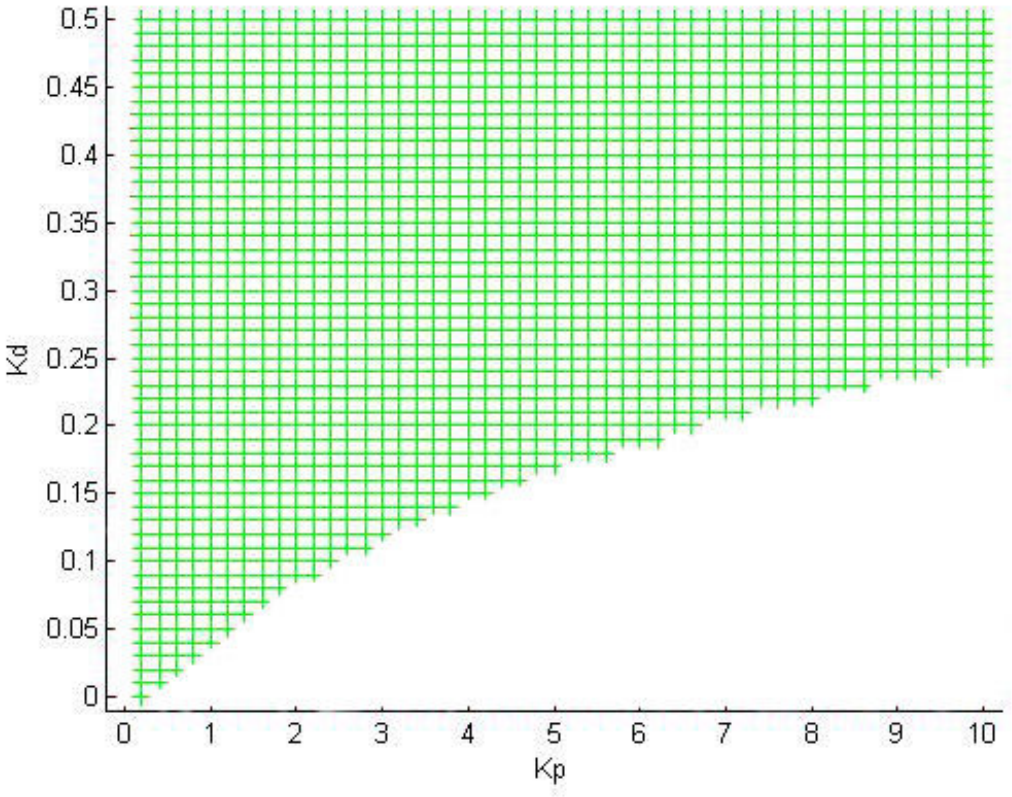

Figure 24: Graph showing stable gain values for a control system tracking a path at a velocity of 10 m/s

Here, small proportional gains are stable without derivative feedback. Also, for higher proportional gains, the required derivative gain is lower than at 23 m/s. As the vehicle slows down, at approximately 5 m/s the required derivative gain for stability suddenly drops off. As shown in Figure 25, all positive K_{PP} are stable for speeds up to 4.5 m/s.

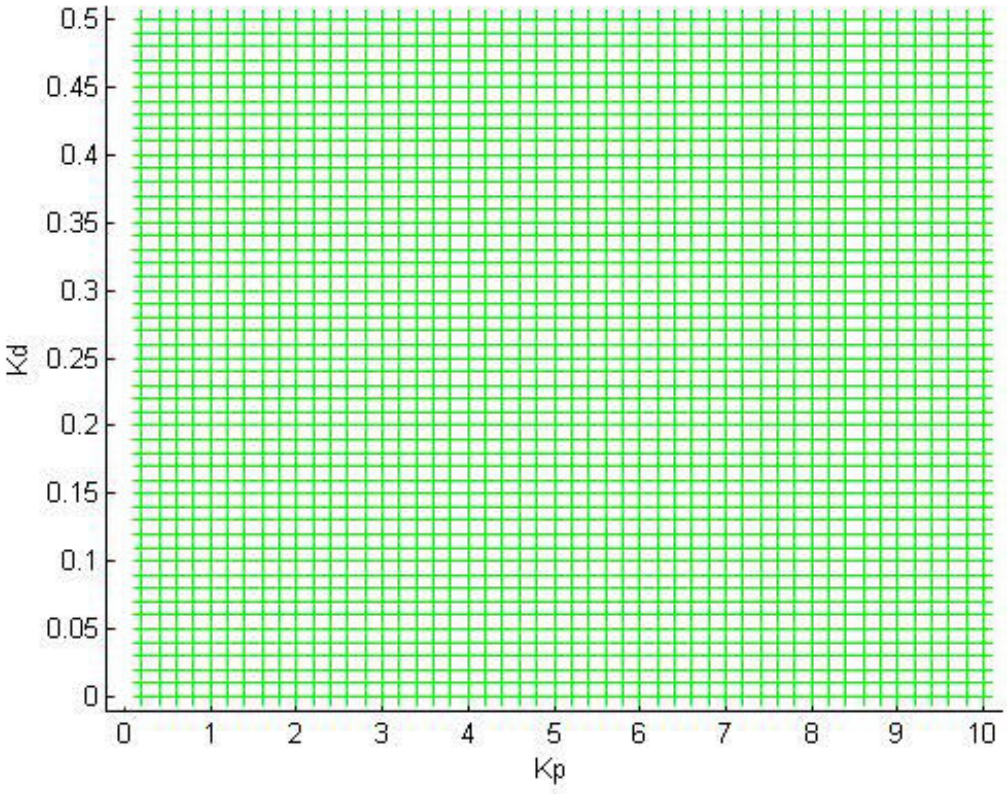

Figure 25: Graph showing stable gain values for a control system tracking a path at a velocity of 4.5 m/s

For further insight into choosing gains to achieve acceptable performance, the digital root locus is used. As before, the transfer function is transformed to digital form using a bilinear transformation at a sampling rate of 20 Hz. Matlab is used to draw the graph, and it is shown in Figure 26 for a velocity of 23 m/s using a derivative gain that is 1/4 of the proportional gain. This relationship of gains satisfies the stability criteria for all speeds.

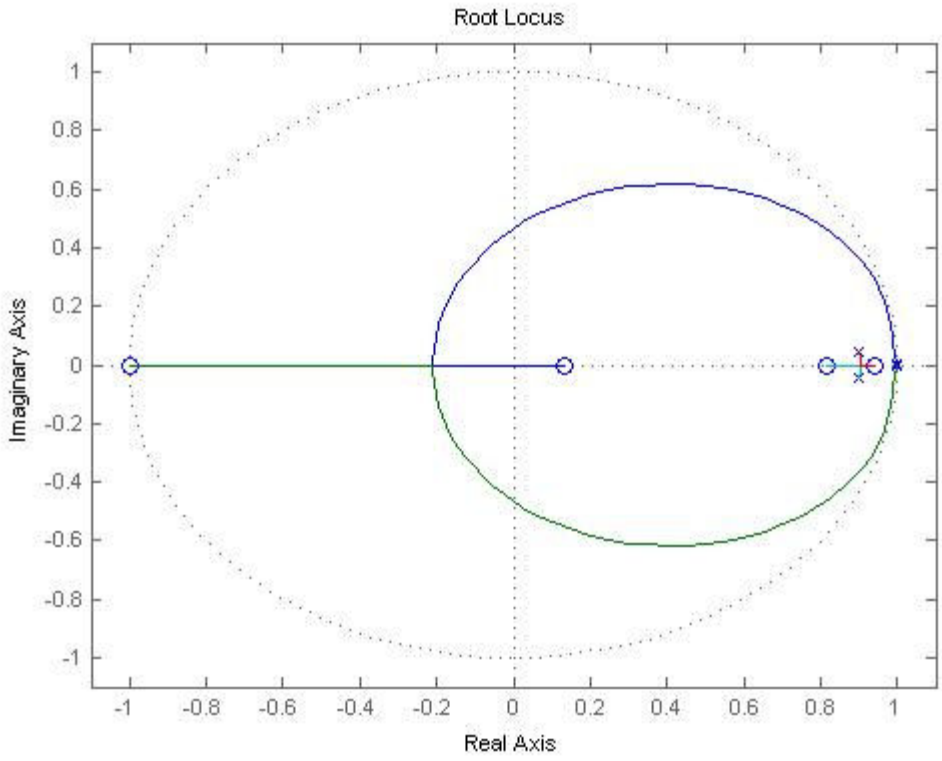

Figure 26: Digital root loci for tracking a path at a velocity of 23 m/s, using proportional and derivative gains

Here, only low and very high gains allow for a response that has low overshoot. Using high gains for the path control would also overpower the heading control, as both feedback errors contribute to the steering angle, as shown in equation (47). Heading control needs to be the dominant feedback term, so low path gains are used to simply bias the vehicle back to the path.

As expected, the digital root locus at a speed of 6 m/s has a similar shape, but displays better overshoot performance with lower rise time, as can be seen in Figure 27.

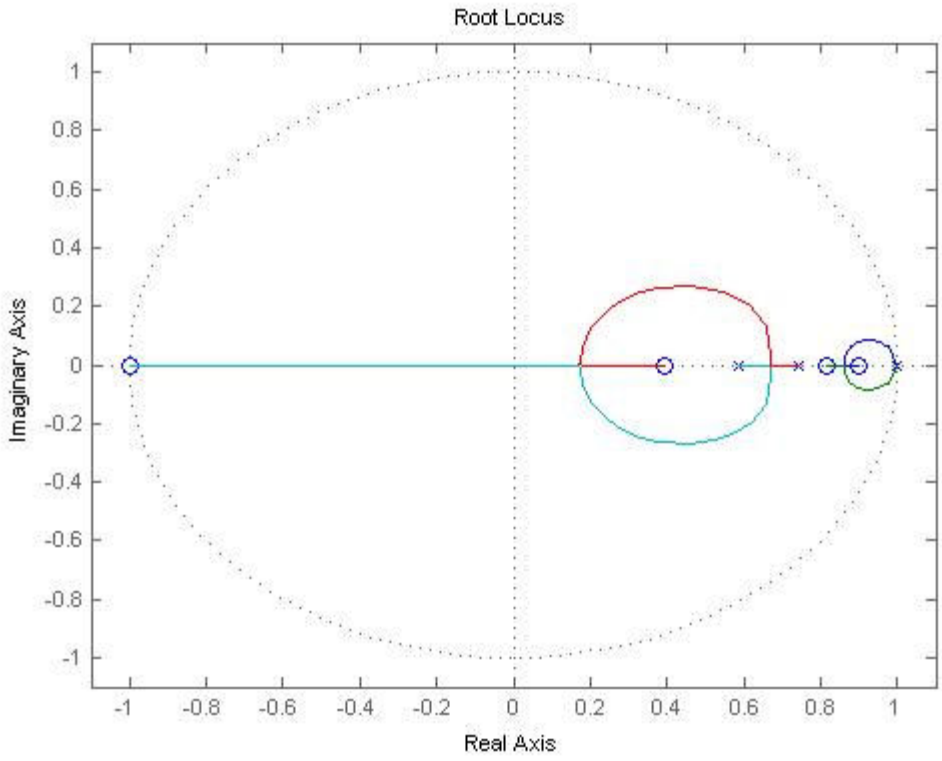

Figure 27: Digital root loci for tracking a path at a velocity of 6 m/s, using proportional and derivative gains

Furthermore, as with the gains associated with heading control, the path gains could be calculated as a function of velocity. This is unnecessary here, as path feedback is secondary in the control logic. As the velocity of the vehicle increases, the heading feedback proportional gains will decrease, actually putting more emphasis on the path gains than at lower speeds. This is desirable if it is assumed that the vehicle is traveling at a high speed because the target waypoint is further away, and the vehicle has more time to move toward the path centerline. After experimenting with the complete system using a software-in-the-loop simulation and choosing gains primarily for higher speeds, a K_{PP} value of 0.05 and a K_{DP} value of 0.0125 were decided upon for the preliminary design.

LONGITUDINAL DYNAMICS – SPEED CONTROL

The control system must also modulate the throttle and brake to achieve a desired speed. In this case, overshoot is more important than rise time, because the vehicle cannot be permitted to overshoot a speed limit. This could possibly cause disqualification by DARPA race officials for violating the speed limit by a significant margin. Also, the system is saturated by the engine's lack of power, and takes several seconds to reach the desired speed. However, the brakes can absorb much more power than the engine can produce, so the system can achieve a much higher acceleration, and thus is generally more unstable. Thus, the control will be designed more towards smooth braking, while verifying that the throttle gains are not set too low. If the system is given more power to slow down than to speed up it would be highly nonlinear, so this is avoided.

Before the differential equations can be transformed to determine a transfer function, they must be linearized due to the squared term associated with aerodynamic drag and the constant rolling resistance term. As in the heading dynamics, the constants in the longitudinal differential equations are grouped together and replaced with a placeholder for simplification.

$$\ddot{x} = -\frac{H}{G} - \frac{I}{G}\dot{x}^2 + T_e\frac{J}{G} \tag{34}$$

Where the placeholder constant are defined as:

$$H = f_r mg$$

$$I = \frac{1}{2}\rho C_D A$$

$$J = \frac{N_t N_f \eta_{tf}}{r}$$

$$G = m + \frac{(I_e + I_t)N_t^2 N_f^2 + I_d N_f^2 + I_w}{r^2}$$

An operating speed of 10 m/s is chosen as a point to linearize about. This speed requires a torque for equilibrium, and the result is:

$$\dot{x}_0 = \sqrt{\left(\frac{H}{G} - \frac{T_{e0}J}{G}\right)\frac{-G}{I}} \tag{35}$$

Where T_{e0} = 75 N*m and the operating point ($\dot{x}_0$) is 10 m/s.

The linearized system is now given by:

$$\ddot{x}_* = \left[\frac{\partial \ddot{x}}{\partial \dot{x}}\right]_{\dot{x}_0} \dot{x}_* + \left[\frac{\partial \ddot{x}}{\partial T_e}\right] T_{e*} \tag{36}$$

Solving this leads to:

$$\ddot{x}_* = -2\frac{I}{G}\dot{x}_0\dot{x}_* + \frac{J}{G}T_{e*} \tag{37}$$

Where $\dot{x}_*$ is the change from the operating point, $\dot{x} = \dot{x}_0 + \dot{x}_*$

To further simply the equation, more placeholders are used for the remaining constants.

$$\ddot{x}_* = -K\dot{x}_* + LT_{e*} \tag{38}$$

Taking the Laplace transform, and dropping the asterisks reveals the linearized transfer function. This is simply a first order system with time constant 1/K.

$$\frac{\dot{x}}{T_e} = \frac{L}{s + K} \tag{39}$$

For speed control, integral feedback was added to eliminate steady state error inherent in the system, as well as un-modeled distances such as grade. Using negative feedback with PID control, the closed loop transfer function becomes:

$$\frac{\dot{x}}{\dot{x}_d} = \frac{K_{DV}Ls^2 + K_{PV}Ls + LK_{IV}}{(K_{DV}L+1)s^2 + (K_{PV}L+K)s + LK_{IV}} \tag{40}$$

Using Routh's stability criteria, all coefficients of the characteristic equation must remain positive for the system to be stable. The coefficients are:

$$\begin{aligned} a_1 &= \frac{K_{PV}L + K}{K_{PV}L + 1} \\ a_2 &= \frac{K_{IV}L}{K_{DV}L + 1} \end{aligned} \tag{41}$$

For all positive speeds, both K and L are positive, which means that no positive values of any of the gains can drive the system unstable. As with heading control, a digital root locus is used to gain more insight into how different gains affect the performance of the closed loop system. Transforming the loop transfer function into its digital form using a feedback rate of 20 Hz, an equivalent brake force of 450 Nm for the engine torque, and using model parameters defined in Table 2 leads to:

$$\frac{0.101z + 0.101}{z - 0.996} \tag{42}$$

Using this transfer function with a proportional feedback (i.e., $K_{DV} = K_{IV} = 0$), the digital root locus is shown in Figure 28:

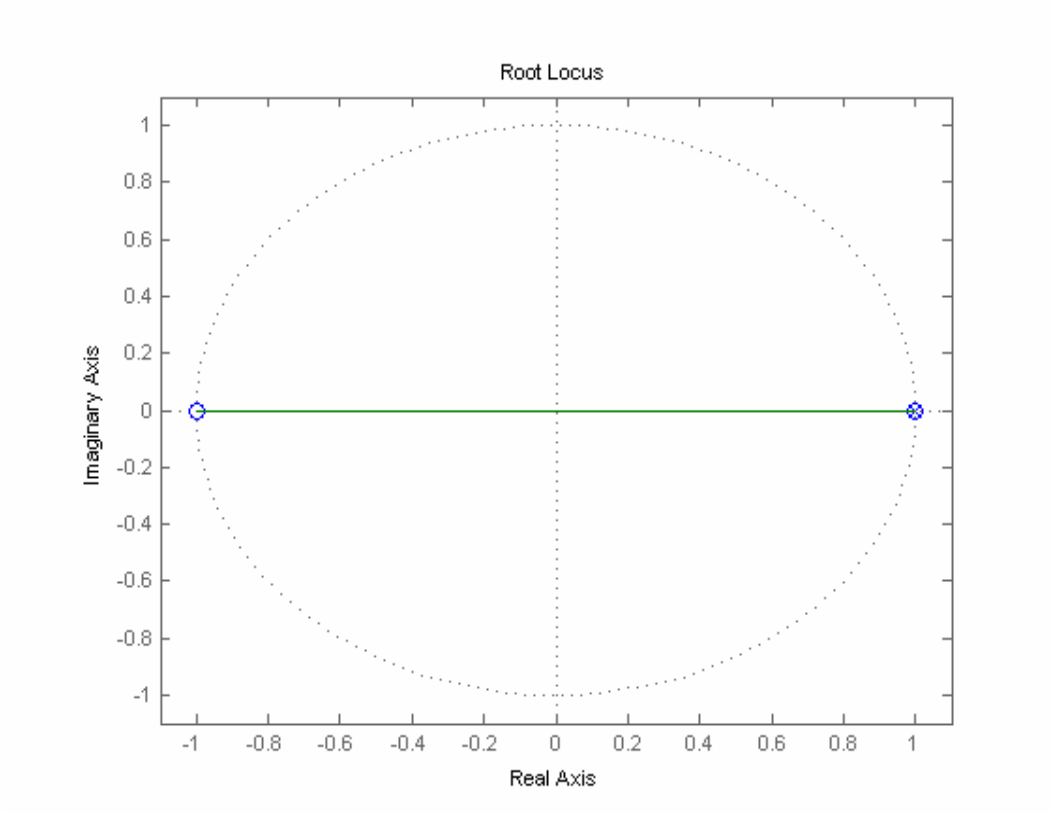

Figure 28: Digital root locus for a purely proportional feedback for longitudinal dynamics

At low gain there is no overshoot, and as the gain raises, the rise times decrease. At a gain of 9.8, the root passes the imaginary axis and the overshoot begins to increase. At a gain of infinity, the overshoot is 100%, but the system is still stable, as the root locus is completely within the circle. At a gain of 9.8 the system has a natural frequency of 119 rad/s (19 Hz).

In an attempt to improve the performance of the system, a non-zero derivative gain is used. Because a derivative gain primarily decreases the overshoot of a system, a higher proportional gain can be used to increase rise time. Using K_{DV} as the parameter of

interest, with a constant $K_{PV} = 9.0$, another root locus is generated, and is shown in Figure 29.

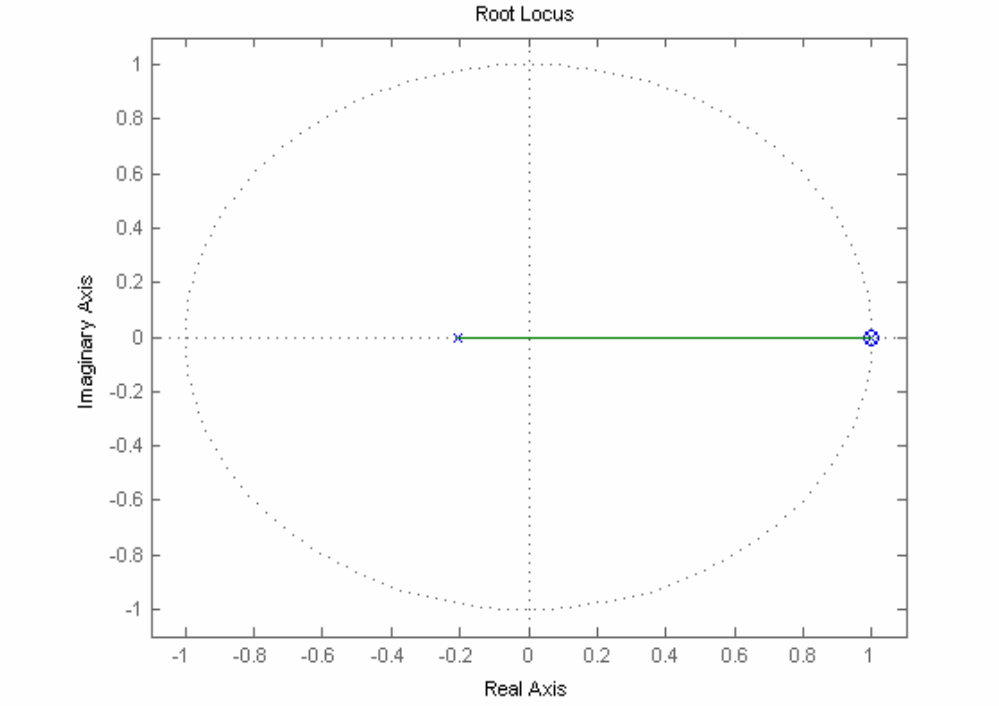

Figure 29: Digital root locus varying the derivative gain with a constant proportional gain of 9.0, for longitudinal dynamics

After a few iterations, a value of $K_{DV} = 1.0$ is chosen here to minimize the rise time of the system while allowing no overshoot.

Adding an integrator to the system induces oscillations and decreases the rise time. The root locus in the Figure 30 varies the integrator gain with $K_{PV} = 9.0$ and $K_{DV} = 1.0$, and shows that higher values of K_{IV} decrease the damping in the system and causes overshoot. Values can be chosen to minimize this, and an integrator could be useful when encountering obstacles, which will require additional throttle to maintain movement. Hills can be sensed by the Inertial Navigation System (INS), so a feed forward term could be added to account for the extra load in the system. However, deep sand or mud would require the use of an integrator.

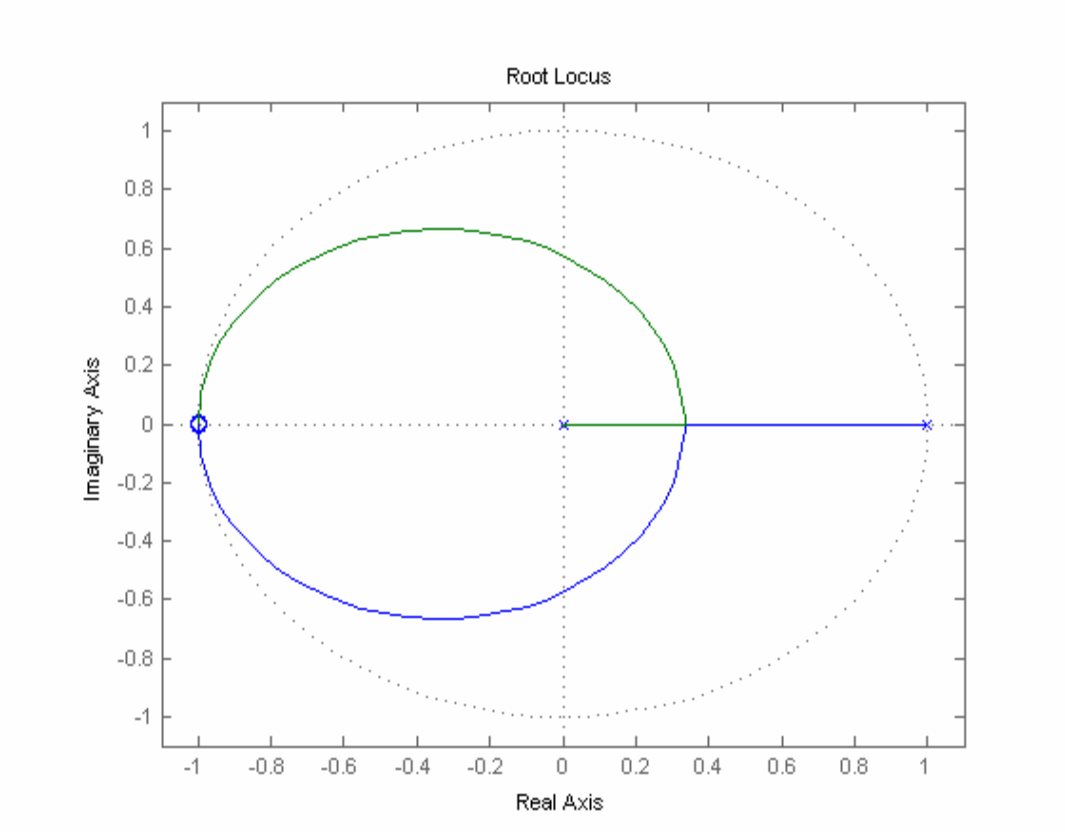

Figure 30: Digital root locus varying the integrator gain with a constant proportional gain of 9.0 and a constant derivative gain of 1.0, for longitudinal dynamics

To finish the design, a conservative integrator gain of $K_{IV} = 0.5$, together with a $K_{PV} = 9.0$ and $K_{DV} = 1.0$ is chosen. While the linearized differential equations used to choose these gains closely matched the responses of the actual vehicle, the actual system is nonlinear. Linearizing these equations for use in the classical control design approach causes increased error when compared to the response of the actual vehicle. After testing the vehicle, the final gains used were much lower, and will be discussed in 'Waypoint Following Using Physical Vehicle' section.

WAYPOINT FOLLOWING

Now that a controller has been designed to direct the vehicle to a desired speed along both a desired heading and path, it is necessary to design the remainder of the software to determine these desired states. A GPS waypoint is an absolute location on the earth defined by a latitude and longitude in degrees which, when combined with elevation, specify the spherical coordinates of the waypoint. To simplify matters, the waypoints are converted to a flat Cartesian system, which will be discussed in a subsequent section. The final goal of the waypoint following algorithms is to determine both desired heading and desired speed.

DESIRED HEADING HEURISTICS

The simplest (and most valuable) approach is to calculate a heading which points directly toward the waypoint. Because the coordinate system is Cartesian, the desired heading is simply:

$$\text{Desired heading} = \psi_d = \tan^{-1}\left(\frac{Y_{WP} - Y_{TRUCK}}{X_{WP} - X_{TRUCK}}\right) \tag{43}$$

The heading error is defined as $e_\psi = \psi_d - \psi$. The magnitude of the heading error can, in certain circumstances, be larger than 180 degrees. This means that the vehicle will turn the "long way" toward the desired waypoint which is behind it. To avoid this, 360 degrees is subtracted from the heading error if it is greater than 180 degrees, and 360 degrees is added to the heading error if it is less than -180 degrees. A negative heading error requires a left turn to compensate, and a positive heading error requires a right turn to compensate.

This approach works well when the vehicle is far away from a waypoint. However, if the vehicle approaches the waypoint at a slight offset, the desired heading

can quickly change from 1-2 degrees to 80+ degrees. This causes the vehicle to lurch to the side before reaching the waypoint. This is due primarily to error in the GPS heading and the coordinate transformation, as well as steady state error in the controller. A simple and robust approach to correct this problem involves defining a circle of specified radius around the waypoint. As soon as the vehicle enters the circle, the waypoint is considered to be achieved. Determined through experimentation, the radius of this circle was set to 3 meters.

Because the vehicle must follow a list of consecutive desired waypoints, it is assumed that the terrain along the straight-line path between the waypoints is the most desirable. Thus the second half of the heading feedback defined in equation (47) is to direct the vehicle to follow the straight line path between the two waypoints. To calculate the error directly, the distance the vehicle is off the path is determined, the geometry of which is shown in Figure 31.

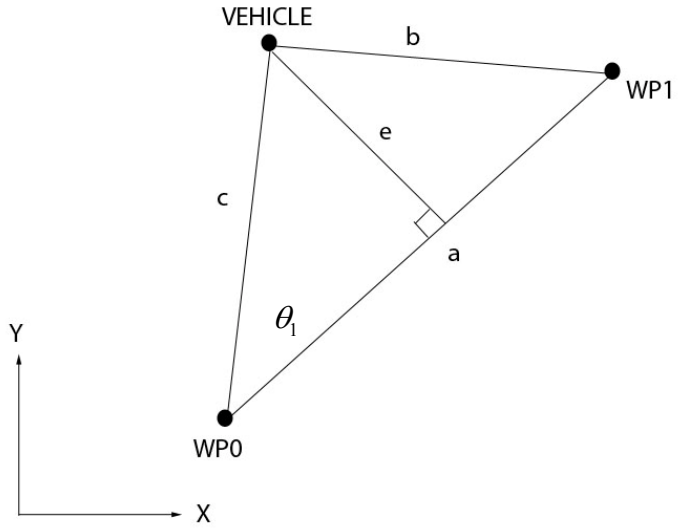

Figure 31: Geometry of path error calculations

The distances a, b, and c are easily calculated, and the law of cosines is used to calculate θ_1.

$$\theta_1 = \cos^{-1}\left(\frac{c^2 + a^2 - b^2}{2ca}\right) \tag{44}$$

Using θ_1, e (path error) can be calculated.

$$e = c\sin(\theta_1) \tag{45}$$

This gives the offset distance magnitude, but no direction information. If the cross product of the vector from WP0 to WP1 and the vector from WP0 to the vehicle is calculated, the sign of the result will give the direction. The cross product is:

$$((x_{WP1} - x_{WP0})(y_{TRUCK} - y_{WP0}) - (x_{TRUCK} - x_{WP0})(y_{WP1} - y_{WP0}))\hat{k} \tag{46}$$

If the result is positive, the truck is to the left of the line, and the path error is designated as positive, so the vehicle turns right. If this path feedback alone determined the heading of the vehicle, it would never be guaranteed to reach the desired waypoint. It would simply oscillate in figure 8's around the path. Only in combination with the heading error to the desired waypoint does it have value. Because the heading error in the C programming is in radians, its max value is 3.14. The path error value is in meters, and is typically much higher, thus it requires much lower gains to scale properly.

The path and heading errors and their derivatives are multiplied by their respective gains and added together to determine the steering angle of the vehicle.

$$\begin{aligned} Steerangle = {} & headingerror * K_{PH} + patherror * K_{PP} + \\ & \frac{d}{dt} headingerror * K_{DH} + \frac{d}{dt} patherror * K_{DP} \end{aligned} \tag{47}$$

This control scheme works well, as long as the heading error is less than 90 degrees. If the truck is pointed away from the desired waypoint, the heading error can be of the opposite sign than the path error. If the errors are similar in magnitude, they will cancel, and the truck will drive straight. This is precisely what happens if the vehicle reaches a waypoint with the next desired waypoint almost directly behind it. To correct

this situation, the path feedback is multiplied by a scaling factor to limit its effect when the heading error magnitude is large. The scaling factor is specified in equation (48).

$$\begin{aligned} pathmultiplier &= 0.71619 * headingerror - 1.125 \quad && 10° < headingerror < 80° \\ pathmultiplier &= 0 && headingerror > 80° \\ pathmultiplier &= 1 && headingerror < 10° \end{aligned} \tag{48}$$

The path multiplier is a continuous function that is linear between heading errors of 10° and 80°. Below and above it is set to one and zero, respectively. The alternative is simply using an 'if' statement that sets path error to zero when the heading error is above a certain value. However, as the truck turns, and the heading error falls below the criterion, the truck then gets an abrupt steering input toward the centerline. Having the multiplier slowly increase prevents this from happening. This equation is presented in graphical form in Figure 32.

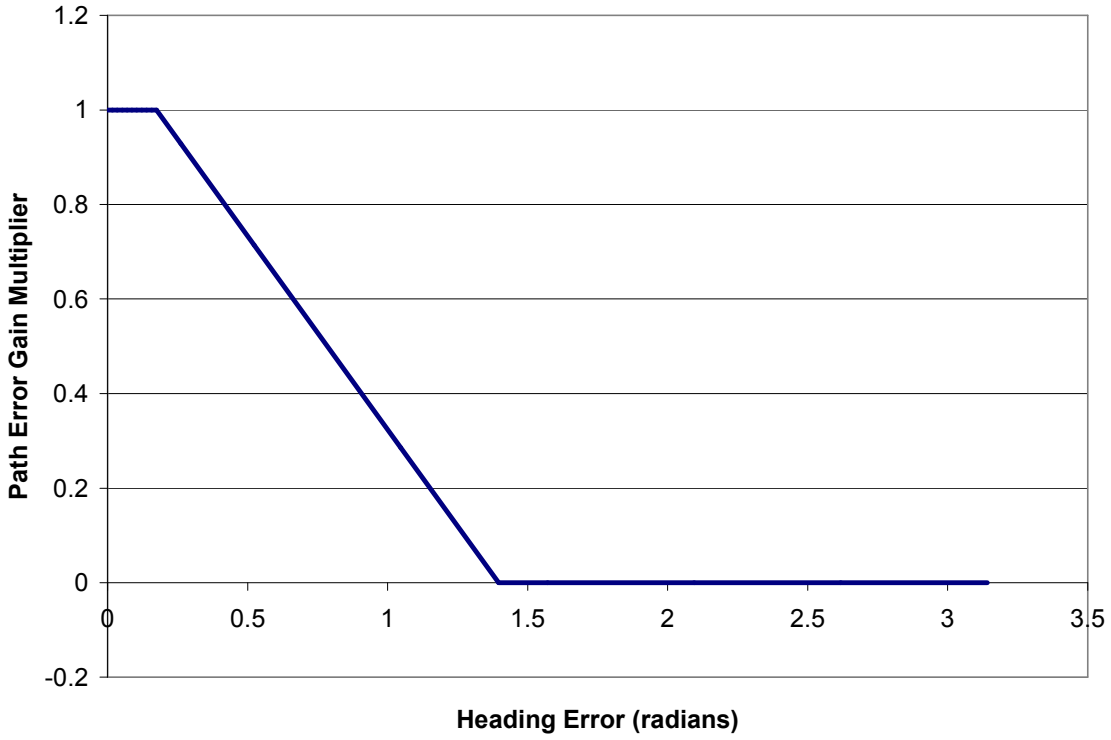

Figure 32: Relationship between heading error and the path error multiplier

The goal of this controller is to turn the vehicle toward the desired waypoint as fast as possible. However, large steering angles will result in unsafe conditions at high speeds. Thus a steering saturation as a function of velocity is needed to maintain a safe lateral acceleration. Driving tests were conducted at many speeds, and a maximum lateral acceleration of 0.58 g's was determined for an experienced human operator. However, all autonomous testing was done with lateral acceleration limited to 0.37 g's. This allows for aggressive turning, but provides for a margin of safety to account for uneven terrain and slopes. The following logic is used to determine the maximum steering angle as a function of speed to limit the lateral acceleration.

From kinematics the lateral acceleration is approximated by:

$$a = \frac{V^2}{R} \tag{49}$$

R = Radius of circle

V = Velocity

For very small slip angles the vehicle's steering angle is approximated by:

$$\delta = \frac{L}{R} \tag{50}$$

L = Vehicle wheelbase

Combining these equations leads to:

$$\delta_{\max} = \frac{a_{\max} L}{V^2} \tag{51}$$

DESIRED SPEED HEURISTICS

Several different algorithms determine the desired heading the controller will receive. All algorithms are independent; each calculates a safe speed based on certain situations. The desired speed is simply the lowest of all the algorithms.

Slowing for Known Turns

In the actual competition, DARPA will supply the maximum speed between waypoints and the vehicle attempts to travel at this max speed. Since the change in heading required after reaching a waypoint may be large and the steering angle is limited based on speed, the steering will be saturated after the vehicle reaches a waypoint and turns toward the next one. This will cause the vehicle to swing out wide after passing through the waypoint, and could cause it to leave the road. The way that a human driver solves this problem is to slow the vehicle down when approaching a waypoint requiring a large heading change. Figure 33 shows a graph of the maximum speed of the vehicle as a function of change in heading for a turn. This graph is based on the author's driving experience as well as maximum steering angles at certain speeds. A simple exponential curve is fit to the data to simplify its transition into computer code.

$$TV = 4.761 * TA^{-0.576} \tag{52}$$

TV =Turning Velocity

TA =Turn Angle

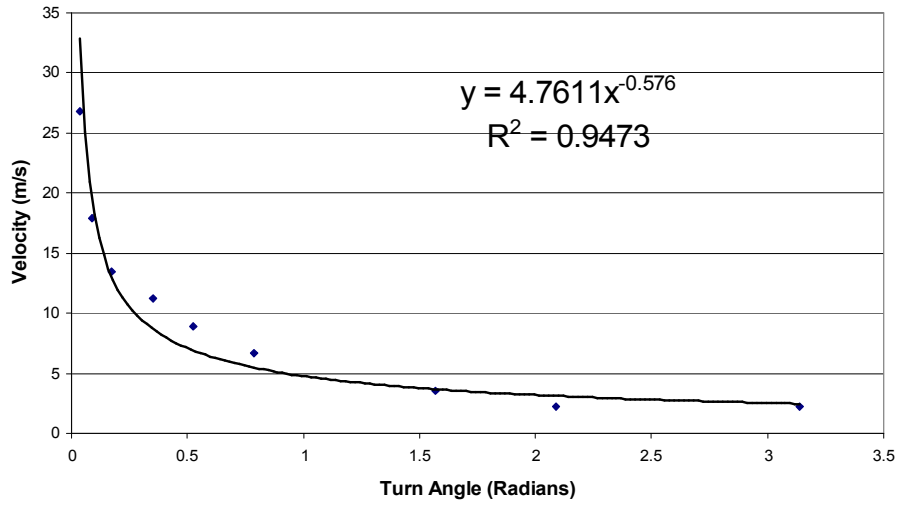

Figure 33: Graph of turn angle vs. max desired speed

The turning angle is calculated using the law of cosines. In Figure 34 the geometry is shown graphically. a ,b, and c are calculated from the X and Y values of the

vehicle position and the two waypoints. WP1 is the next desired waypoint, and WP2 is the subsequent desired waypoint. This can be seen in Figure 34.

$$A = \cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2bc}\right) \tag{53}$$

$$TA = \pi - A = \text{Turning Angle} \tag{54}$$

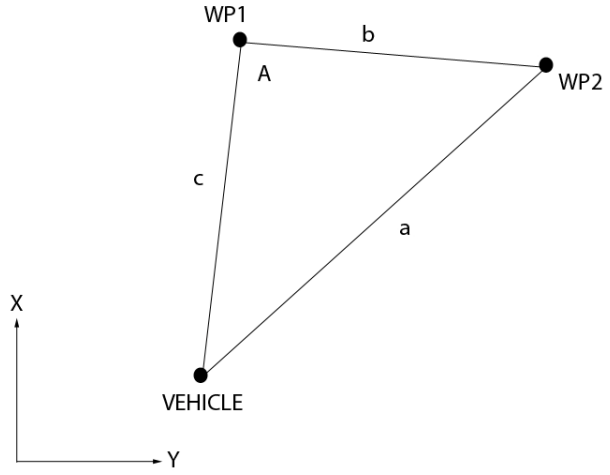

Figure 34: Geometry of turn angle calculations

The turning speed cannot be given as a step function to the controller at the waypoint; the desired speed must be gradually decreased to safely slow the truck. Thus, the desired speed must be a function of the distance to the waypoint as well as the turning speed. To quantify this statement, 50% of the braking power will be used to reach a steady state turning speed within 5 meters of the waypoint. This is reflected in the following equation:

$$V_1 = TS + (Dist - D_0)\beta \tag{55}$$

D_0 = Distance from waypoint at which desired speed is achieved

β = Scaling Constant

When $Dist - D_0 = 0$ the second term in equation (55) disappears, and $V_1 = TS$. Because $(Dist - D_0)\beta$ is added to the turning speed in this equation, β is a constant that relates distance from the waypoint to desired speed. It is based on a linear approximation of speed as a function of distance at a specified deceleration. This linear approximation is conservative since it commands the vehicle to decelerate more at distances further away from the waypoint. The speeds will not be large in the courses the vehicle will follow. This approximation is simple and has been proven to work well in many different speeds and turn angle situations.

Equation (55) is rewritten as equation (56) with the numerical values of D_0 and β plugged in. D_0 is chosen as 5 m, and Beta is chosen as 0.18 (m/s)/m, which was determined by testing.

$$V_1 = TS + (Dist - 5) * 0.18 \quad \text{for } Dist \geq 5 \qquad (56)$$
$$V_1 = TS \quad \text{for } Dist < 5$$

Slowing for Speed Limit in the Following Segment

The maximum speeds supplied by DARPA between the waypoints can be between 5 mph and 45 mph. Thus, when the vehicle passes a waypoint it may be required to slow quickly to reach the maximum speed in the next segment. However, this may mean that the vehicle may exceed the specified maximum speed for a short distance. Therefore, if the maximum speed for the next segment is lower than the next speed for the current segment the desired speed must be lowered before the waypoint is reached to achieve the maximum speed for the next segment before entering that section. This simply requires using the speed limit of the next segment in place of TS (turning speed) found in equation (56).

$$V_2 = SpeedLimit + (Dist - 5) * 0.18 \quad \text{for } Dist \geq 5 \qquad (57)$$
$$V_2 = SpeedLimit \quad \text{for } Dist < 5$$

Slowing for High Steering Angle

When the heading error is high, a sharp turn is desired. If the vehicle is traveling fast, the steering angle will be limited for safety, and the turn will have a large radius. To shorten this radius the vehicle must be commanded to slow down. Based upon the maximum safe lateral acceleration the vehicle can achieve and empirical testing, a relationship was determined to slow the vehicle as a function of steering angle. The steering angle used here is the result of the steering control algorithm, which will precede this algorithm in the control loop. A graph of this relationship can be seen in Figure 35.

$$V_3 = 3.0 * \delta^{-0.4911} \tag{58}$$

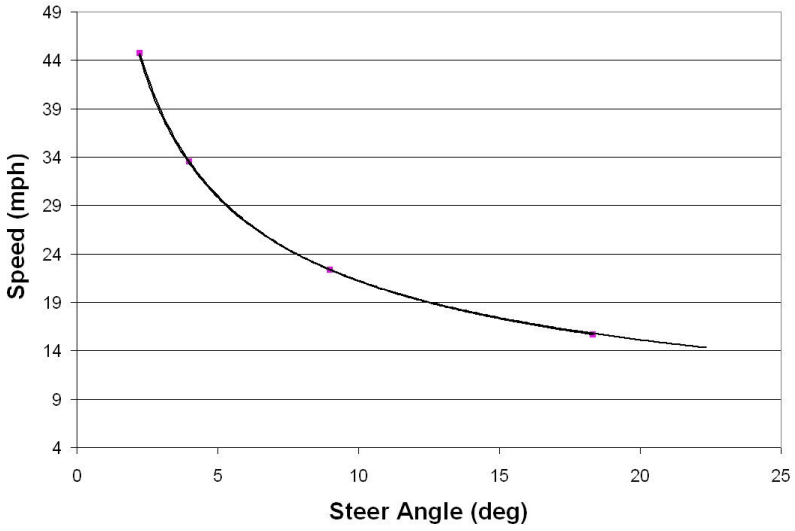

Figure 35: Relationship between steering angle and velocity

This also serves to smooth the turn of the vehicle after it reaches a waypoint with a high required heading change. Before this algorithm was implemented the vehicle would quickly accelerate while turning sharply, resulting in an uncomfortable ride. As the velocity increased, the steering angle would become saturated, and the radius of turn would increase.

Combination of all Speed Algorithms

The final desired speed is simply the smallest of V_1 - equation (56), V_2 - equation (57), V_3 - equation (58), and the speed limit supplied by DARPA (V_4). These speeds are defined as:

V_1 = Maximum allowed speed in response to an upcoming known turn angle

V_2 = Maximum allowed speed in response to a decrease in the DARPA allowed speed limit in the next waypoint segment

V_3 = Speed as a function of steering angle

V_4 = Maximum speed allowed by DARPA

WAYPOINT FOLLOWING SIMULATION

The algorithms described in the waypoint following section were programmed in C, and tested in a closed loop simulation with the vehicle's dynamic model described in the vehicle simulation chapter. This allows the control as well as the heuristic algorithms to be tested and refined before any physical testing on the truck is performed. Initially, the simulation was programmed in Matlab, and then transferred into Labview. Matlab was easier to program for the initial testing, but was difficult for communicate with the GPS and other processes such as the collision avoidance software. Labview's strength is serial communications, but it required 100% of the computer's processor while running real-time. Labview also proved unreliable; it was prone to crashing. The final product, which this thesis concentrates on, was written completely in C. The output is displayed on the screen using Open GL and all data is written to a text file for post processing in Microsoft Excel. The simulation time step is set to 0.05 seconds to mimic the GPS feedback rate in the vehicle. The differential equations are stable at this time step at speeds above 0.5 m/s. If lower speeds are needed a kinematics vehicle model based on no tire slip can be used. The kinematics model accurately models the truck at slow speeds, and it is not unstable. To use the dynamics model, the control section of the loop can be fed sampled and held state values to allow the use of smaller step sizes as needed for stability.

For initial debugging, the simulator was given a waypoint list defining a square spiral. Having the waypoints line up vertically and horizontally allows overshoot and rise times to be easily seen. The plot of the path in such a test is shown in Figure 36, and speed versus time can be seen in Figure 37. The thin red line in Figure 37 shows the desired speed, and the thick blue line shows the simulated vehicle speed.

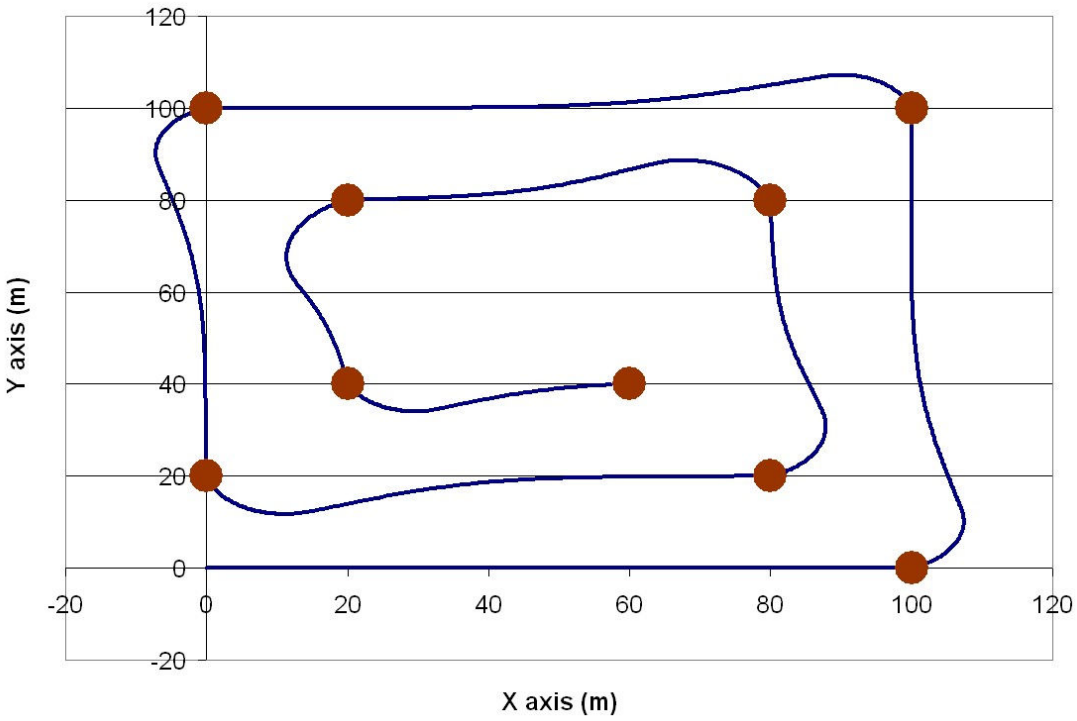

Figure 36: Simulated path of the vehicle around a test square spiral

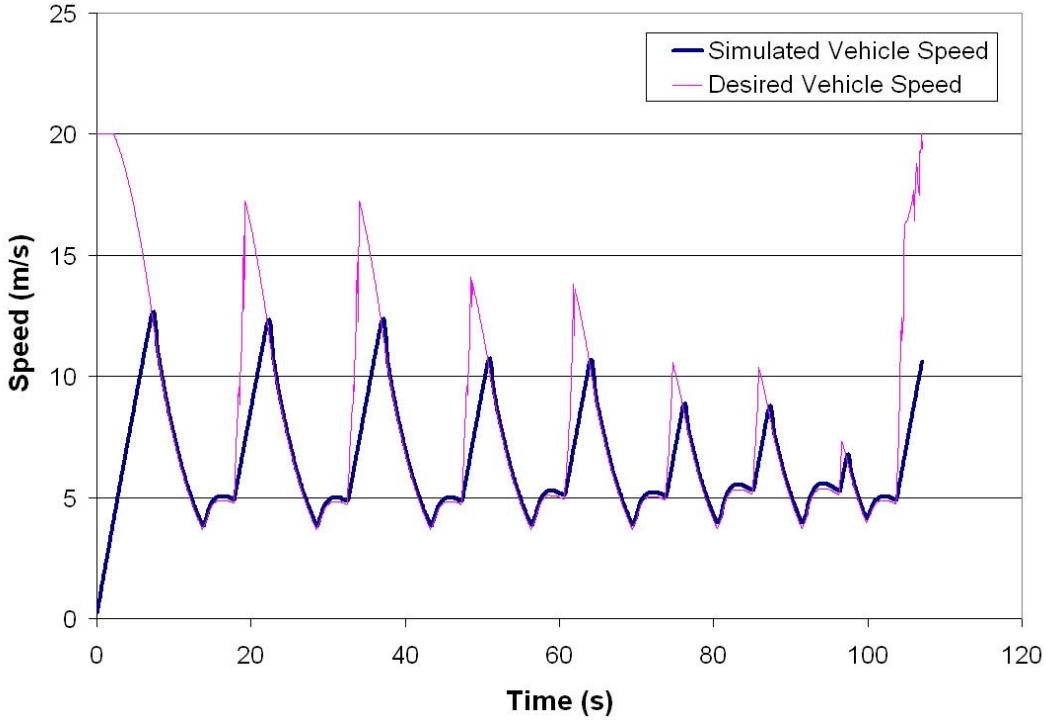

Figure 37: Graph showing the desired vehicle speed and the simulated speed around a squire spiral

The large brown circles in Figure 36 show the size of the waypoints with a 3m allowed error. The gains used are those designed in the 'Controller Design' section. The presence of the path error in the heading control is easily seen in Figure 36, as the vehicle turns back toward the path as it progresses towards the desired waypoint. Also notice the results of the desired velocity heuristics in Figure 37. The resulting 'saw tooth' is exactly how a real driver would drive, slowing down for the turns, and accelerating in the straight portions. The speed oscillations get smaller as the time progresses, as the distance between waypoints gets smaller. There is no turn at the last waypoint so the desired speed is only lowered by the steering angle of the vehicle. The screen output of the path is shown in Figure 38.

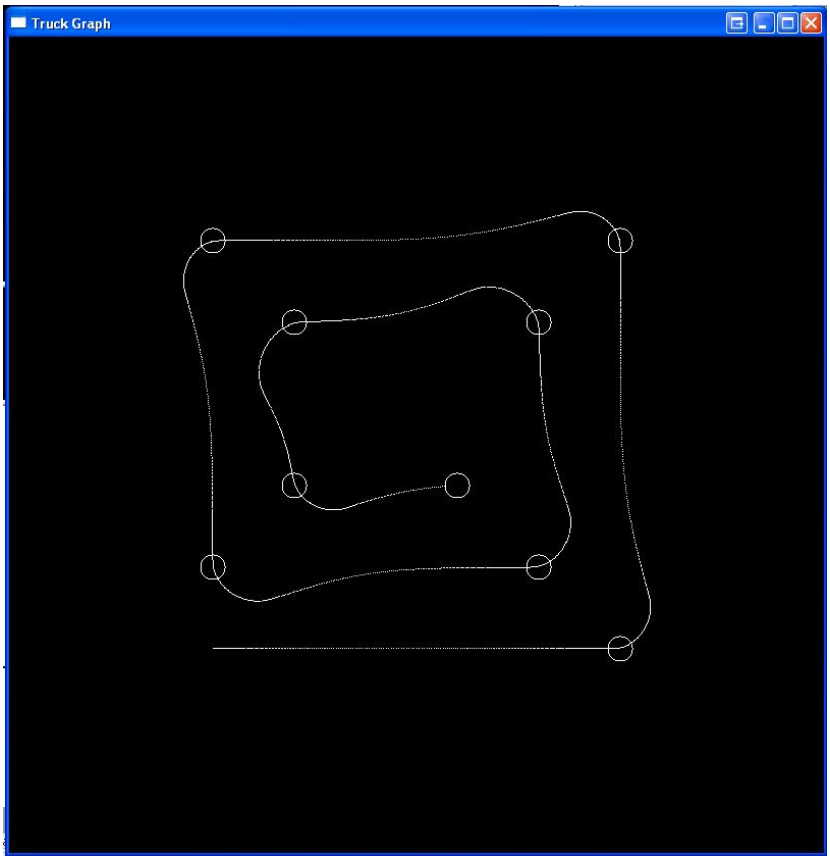

Figure 38: Screenshot of Open GL output of the simulated path of the vehicle around a test square spiral

HARDWARE IMPLEMENTATION AND SOFTWARE SETUP

The simulator discussed in the previous sections was a stand alone program developed to debug and test the algorithms and heuristics used to guide the simulated vehicle around the course. It proved to be difficult to maintain different versions of the software for simulation and to actually control the vehicle hardware. To solve this problem, the control and waypoint section of the program was completely separated from the vehicle simulation. The controller was programmed to communicate with the simulated vehicle using COM ports and a DAQ card exactly as it would on the actual vehicle. The simulator now accepts analog voltages, which determine the steering angle, throttle, and brake, and outputs the position of the vehicle in the NMEA format that the real GPS provides. This allows the exact same controller software to be used in simulation and on the actual truck. However, this is not without problems. The ease of a stand alone program that runs instantly is replaced with two computers, two DAQ cards, and the wires to connect them together. The programs must wait on each other to loop, but they do run approximately twice as fast as real time. The advantage is clear: the program is simply loaded on the truck and run. The program asks upon startup whether it is running the actual vehicle or the simulation. The majority of the communication programming was done by computer science members of the AGV team. The flow diagram of the actual vehicle setup is shown in Figure 40, and the simulation setup is shown in Figure 39. The “path controller” block is described in this thesis.

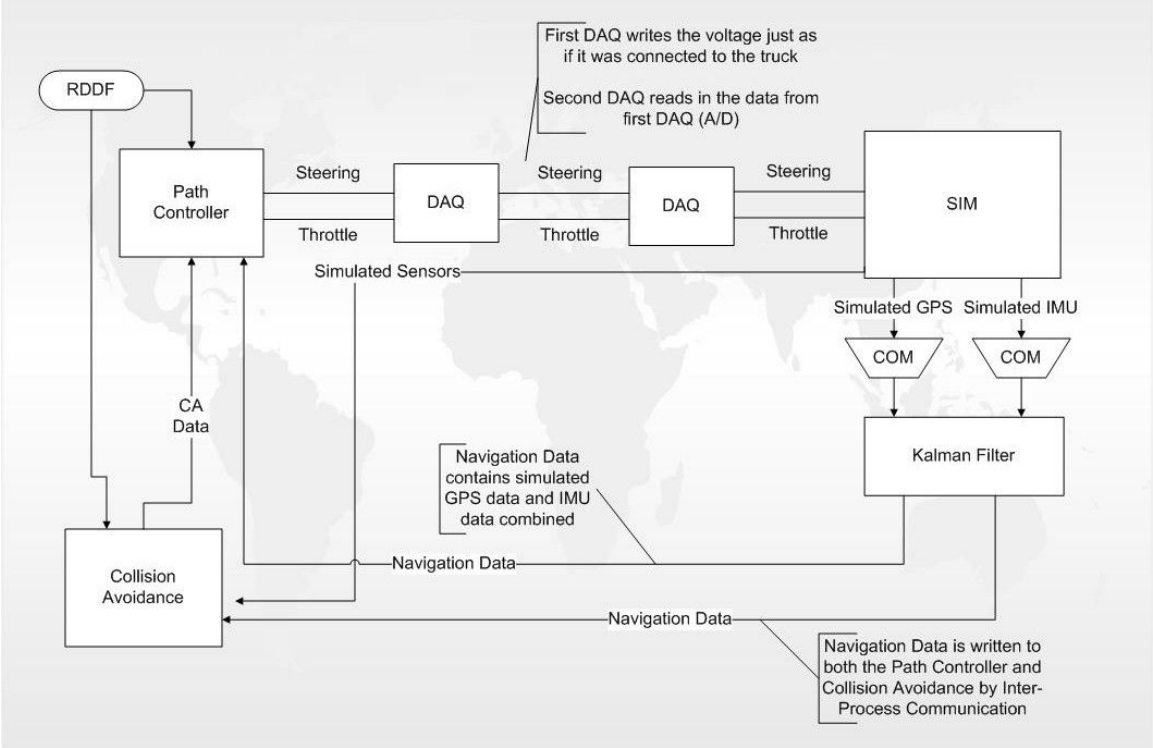

Figure 39: Flow diagram for software setup for simulation [10]

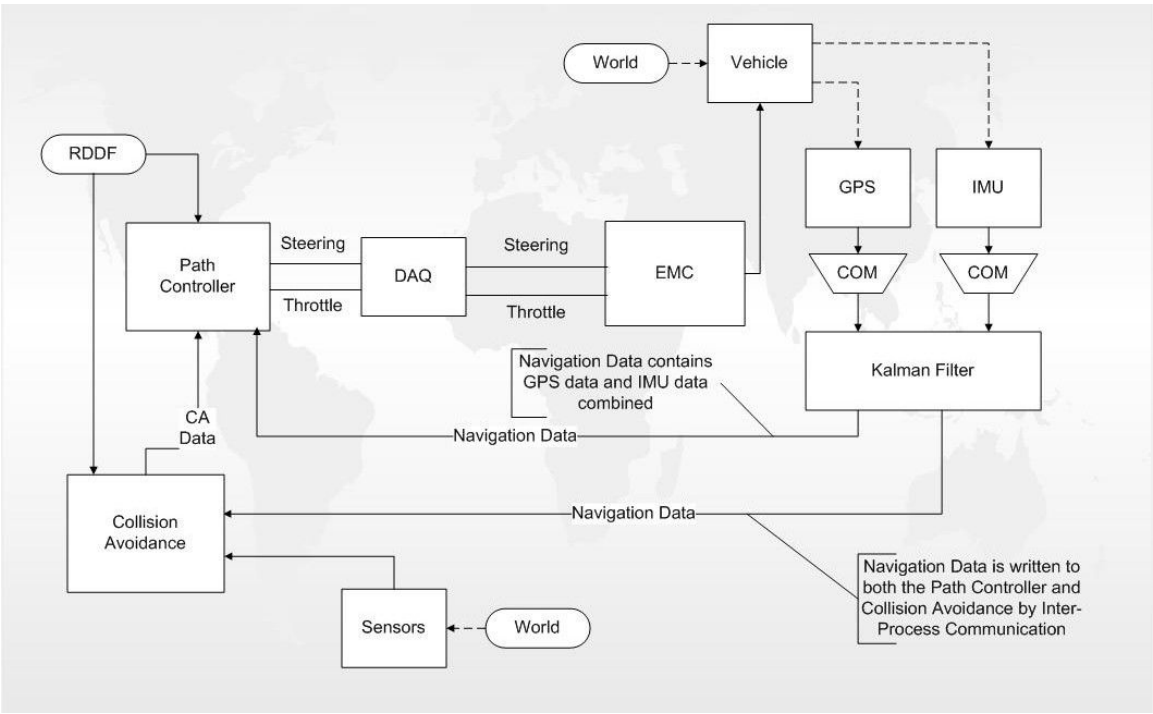

Figure 40: Flow diagram for software setup on physical vehicle [10]

GPS TO CARTESIAN COORDINATE TRANSFORMATION

The waypoints are supplied by DARPA in latitude and longitude, and must be converted to a Cartesian coordinate system for compatibility with the controller calculations. The Cartesian system is based on differences in the X and Y direction from the origin, which is just an arbitrary waypoint specified somewhere close to the start of the course. To begin the calculations, the waypoint must be converted from degrees and minutes (DDMM.MMM) to decimal degrees (DDDD.DDD). The latitude and longitude are divided by 100, and then the remainder is divided by 60 and added to the quotient. This converts the latitude and longitude into degrees so they can be used in normal arithmetic calculations. The equation to convert latitude and longitude to Cartesian coordinates is as follows: [11]

$$X = 2R\sin\left(\frac{(Lat1 - Lat0)}{2}\right)$$
$$Y = 2R\sin\left(\frac{(Long1 - Long0)}{2}\right)\cos\left(\frac{(Lat1 - Lat0)}{2}\right) \quad (59)$$

R = Radius of the earth in meters (6366564.864m)

The accuracy of this method decreases with distance from the origin. An option which is not currently used, is to update the origin using preceding waypoints. As long as the vehicle stays within 20 miles of the origin, the calculation errors are less than the GPS accuracy.

VOLTAGE SCALING AND D/A CONVERSION

The differential equations in the simulation require only a steering input in radians, and a throttle/brake input as a fraction of 100% throttle/brake capabilities. Before the steering or throttle is sent to the D/A, it must be scaled so the voltage corresponds to the proper steering angle. Figure 41 represents the steering calibration that was performed on the vehicle, and equation (60) gives the equation of the calibration. Steering angle was measured by driving a full circle at a certain voltage, and backing out the angle from the radius of the turn using equation (20). The throttle voltage was calibrated in the same

way, using the assumption that the relationship would be linear. The neutral voltage was measured, as well as full brake and full throttle. Equation (61) shows the results, where μ is the throttle/brake percentage in decimal form.

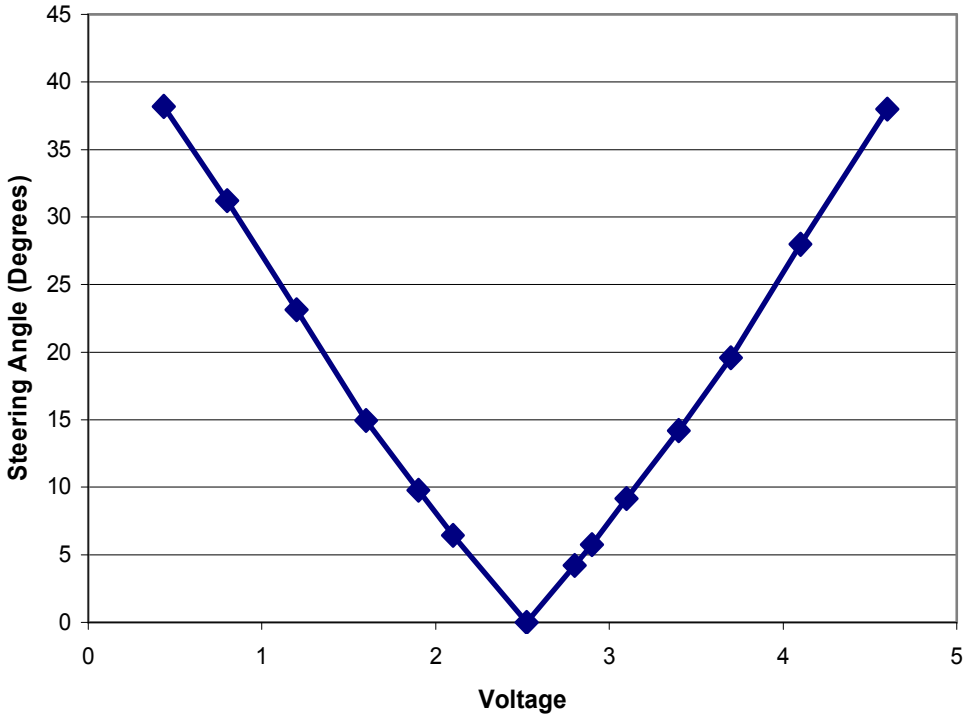

Figure 41: Steering voltage calibration

The voltage, V1, sent to the steering servo is:

$$3.1334\delta + 2.525 = V1 \qquad (60)$$

The voltage, V2 sent to the throttle/brake servo is:

$$1.211\mu + 2.699 = V2 \qquad (61)$$

The manufacturer of the DAQ card supplied C programming libraries, so sending these voltages to the D/A card is easy. The function is simply called, with the channel and the desired voltage in mV.

CONTROL TESTING ON PHYSICAL VEHICLE

Before using the controller to guide the vehicle around a course defined by waypoints, the first step was to verify that the controller design described in the 'Controller Design' section is adequate on the physical vehicle. A short program was written to control the vehicle along a constant heading at a constant speed. All algorithms except the safe steering saturation were disabled. For the first test, the vehicle was pointed perpendicular to the desired global heading, and released from rest, with a desired speed of 10 m/s. It quickly became evident that the control gains were too high. The truck was stable, but had high overshoot. The gains were tuned until overshoot was minimized. The result is shown in Figure 42.

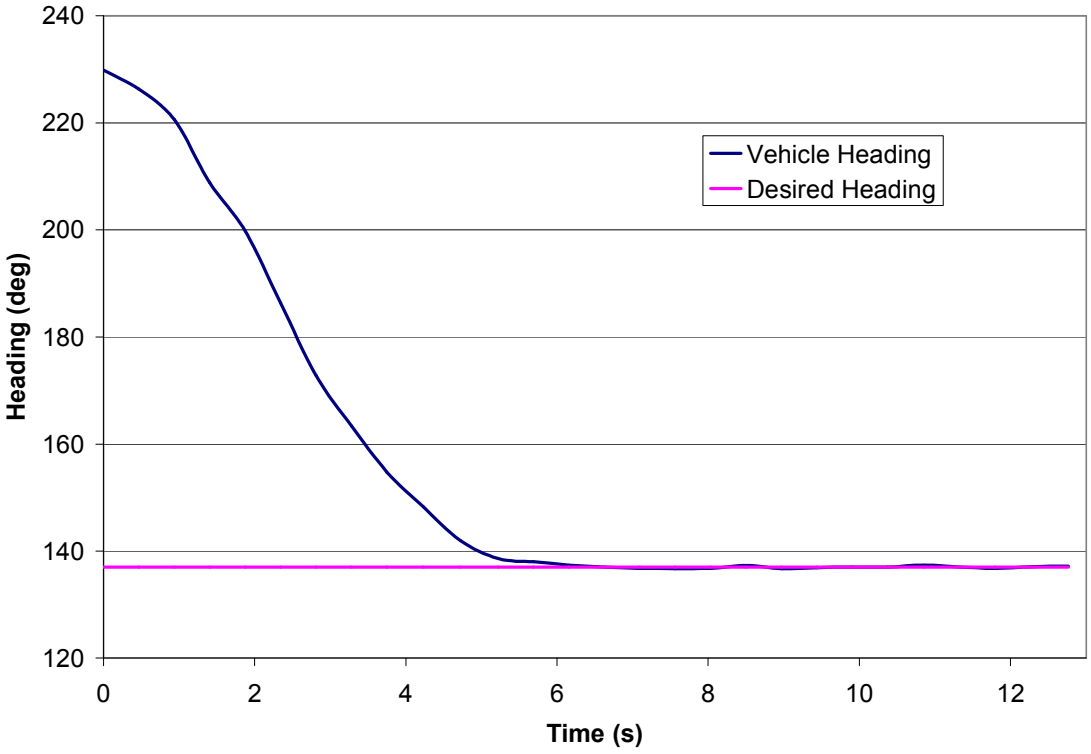

Figure 42: Plot of physical vehicle heading tracking a constant desired heading vs. time at 10 m/s

As can be seen from the graph, the overshoot is almost zero, and it takes approximately 6 seconds to reach the desired heading. If the control algorithm is started while the vehicle is already traveling at a higher speed, this time is decreased. Also, the lack of a large steady state error shows that a heading integrator is not worth the increased overshoot it causes. As mentioned previously the final gains used on the actual truck were smaller than those used in the simulation. This could be primarily due to a suspected delay in the GPS feedback, though this is difficult to prove. The proportional gain's equation was changed slightly and is given below, along with the value of the derivative gain.

Heading control gains:

$$K_{PH} = 3.3V_x^{-0.8} \qquad 0.2 < K_{PH} < 4 \tag{62}$$

$$K_{DH} = 0.04$$

A derivative gain above 0.04 was found to actually increase the overshoot. Because the heading output from the GPS is naturally noisy, the derivative gain multiplies this noise. Even without a derivative gain, the steering output sent to the DAQ card must be smoothed using an averaging filter to damp noise. Furthermore, a RC low-pass filter was used on the output of the DAQ card to filter noise that originated there. A higher quality DAQ card would eliminate the need for this filter.

This same approach was used to test the longitudinal control. The vehicle was pointed along its desired heading, and released from rest. Again, the gains used in the simulation were too high, especially in braking. The gains were scaled down and tuned, and are given below in equation (63). It was also found that adding the operating torque used while linerizing the longitudinal equations to the speed controller was not useful, as the gains on the physical had to be tuned to such a degree. Using these gains, the result of an acceleration test is found in Figure 43. Tracking a decreasing ramp input as the vehicle slowed for a turn was the main reason for lowering the proportional gain.

Longitudinal control gains:

$$K_{PV} = 0.2$$
$$K_{DV} = 0.015 \quad (63)$$
$$K_{IV} = 0.04$$

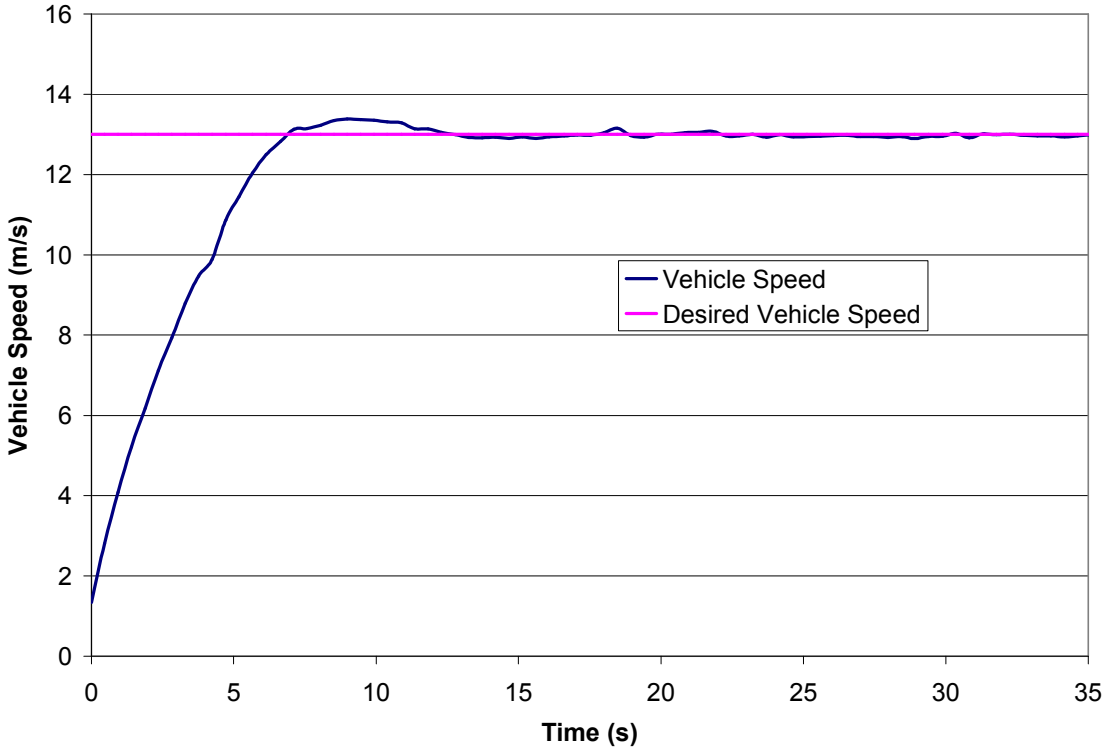

Figure 43: Plot of physical vehicle speed tracking a constant desired speed vs. time using gains shown in equation (63)

Slight overshoot is evident, as a direct result of using an integrator in the control algorithm. Because the vehicle is naturally slow to respond, it spends a lot of time away from the desired speed. Thus an anti-windup was used to limit the maximum integral effect. It was found that a maximum integral value of 5 m/s*s worked well. The integral is definitely needed, as shown in Figure 44, as there is a large steady state error present. Here, the same proportional and derivative gains were used without an integrator to illustrate this point.

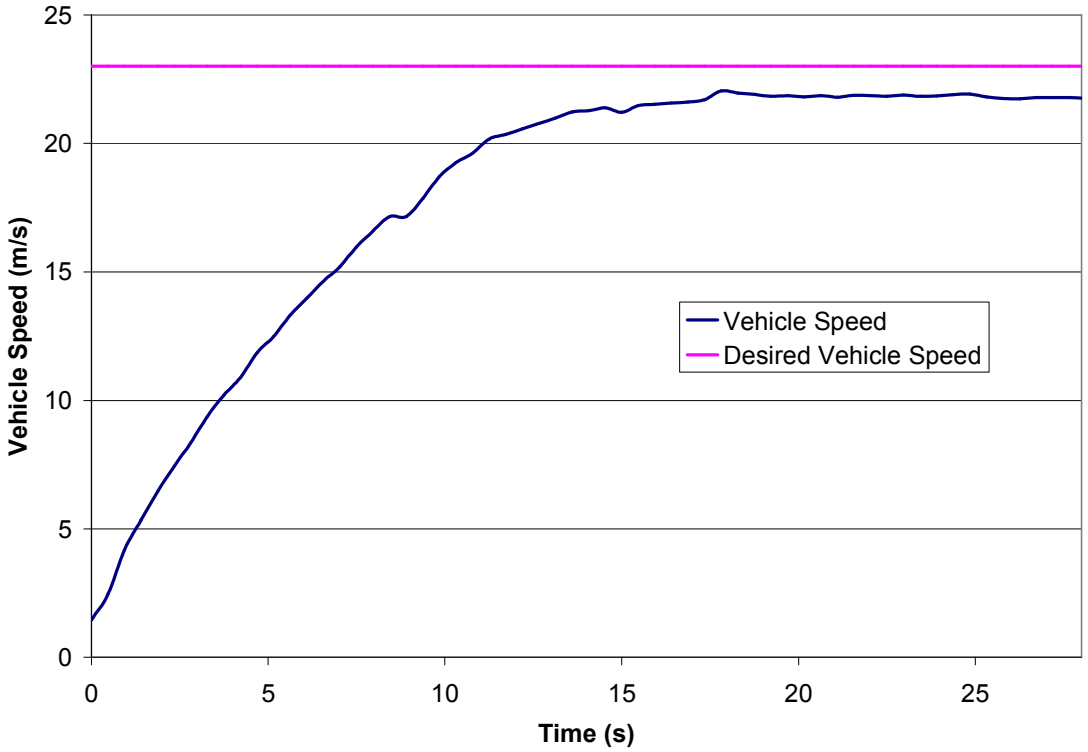

Figure 44: Plot of actual vehicle speed tracking a constant desired speed with no integral gain.

WAYPOINT FOLLOWING USING PHYSICAL VEHICLE

After many month of debugging the control program, the vehicle was ready to follow waypoints. A simple eight waypoint course was set up on a runway at Texas A&M's Riverside Campus. The runway is approximately 300 feet across, and over half a mile long, thus the course is long and narrow. The course was arranged to test as many different conditions as possible, including high speed runs, 180 degree turns, and slower speed slaloms. The waypoints are shown Figure 45, and the RDDF file created for this course is given in the following table:

Waypoint #	Latitude	Longitude	Speed (mph)
0	30.63413	-96.482413	-
1	30.631968	-96.479497	45
2	30.632005	-96.47987	20
3	30.632542	-96.479965	20
4	30.632622	-96.480702	20
5	30.633275	-96.480782	20
6	30.634358	-96.481828	35
7	30.63349	-96.481352	35
8	30.632152	-96.479558	35

Table 4: RDDF file for Riverside Campus test course

The path control gains were able to be tested here, and again the gains designed were too high for good performance, and were lowered until the response was acceptable.

Originally K_{PP} was 0.05 and K_{DP} was 0.0125. The resulting path gains are shown below.

Path error control gains:

$$K_{PP} = 0.004$$
$$K_{DP} = 0.0001 \quad (64)$$

The goal of the path error feedback in the controller is not to have it be the dominant factor in the desired heading of the vehicle, but to have it bias back to the straight line path between waypoints. This gain can be raised to force the vehicle back to the path quicker if it would suit the course the vehicle must follow.

The path of the vehicle can be seen in the following figure. On the test track, the vehicle was not started at the origin but it still drove back toward the centerline between the origin and the first waypoint. This is the reason for the curved path leading up to waypoint 1.

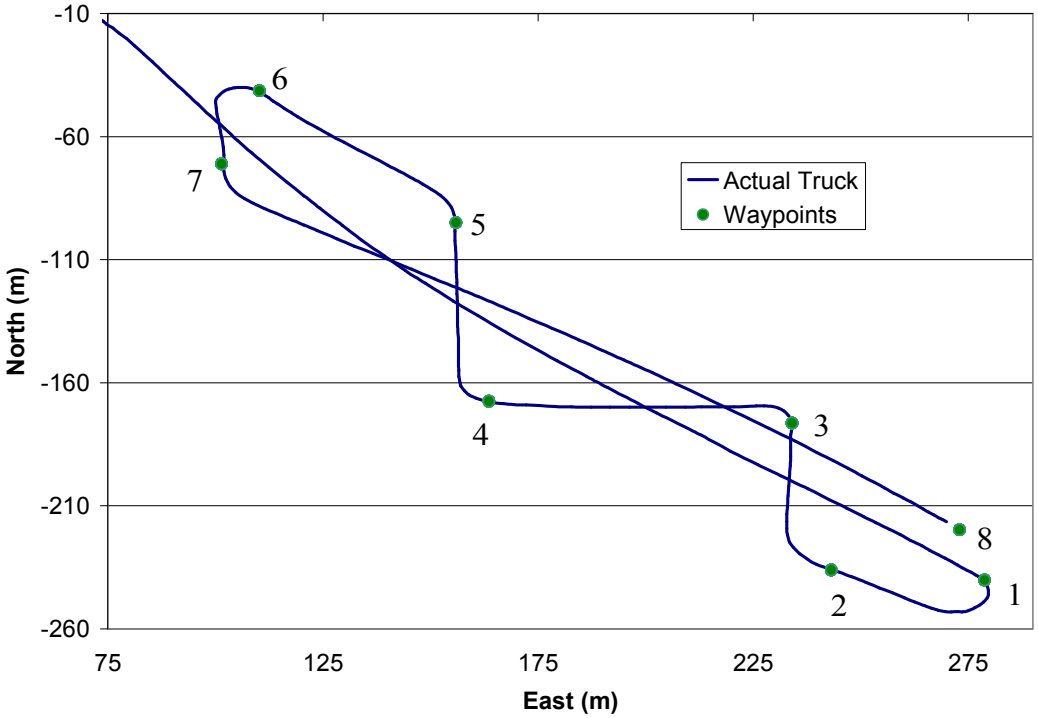

Figure 45: Path of vehicle following waypoints on a test course

The vehicle starts in the upper left hand corner, progressing at 45 mph toward the extreme lower right, after reaching waypoint 1 it turns around and stair-steps back and forth several times thru waypoints 2 - 7. Then it again travels at a high speed to the finish at the lower right hand corner at waypoint 8.

This course was also completed by the simulation to verify that the two matched. The same gains that were found to work best on the truck were used in the simulation. The two were extremely similar, which is shown in Figure 46. The only slight discrepancies were slight and happened during sharp turns, with little effect on the overall path of the vehicle.

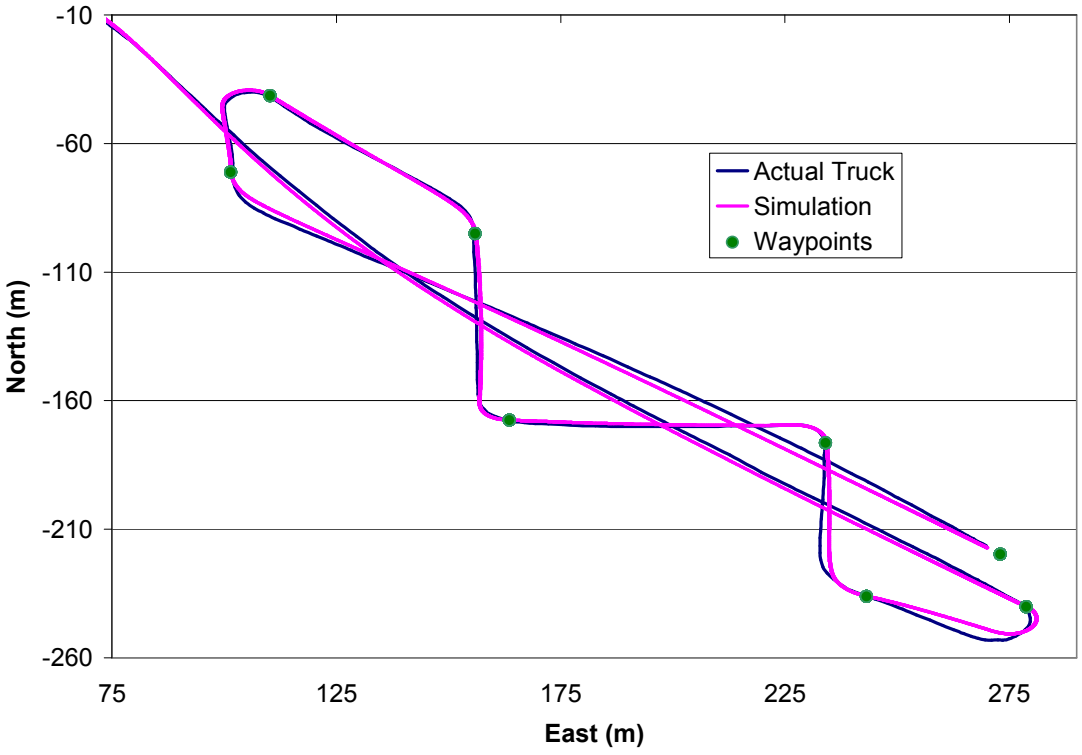

Figure 46: Comparison of actual vehicle path vs. vehicle simulation path when following waypoints on a test course

The graphs of speed versus time are almost identical, with very little propagating error. The simulation ended the run within a second of the actual vehicle. Also worth noting is the consistency of the timing of the different physical runs. Many different runs

were completed using the same starting location and controller gains. The graphs were basically identical, so only one is shown for comparison.

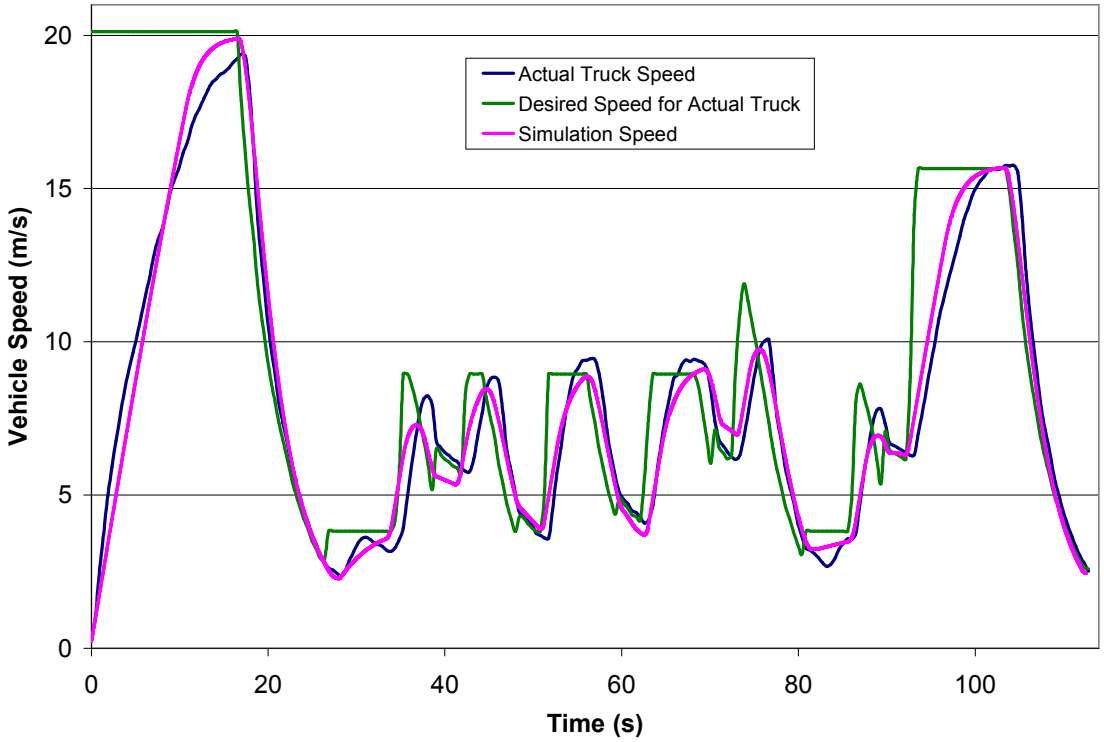

Figure 47: Comparison of actual vehicle vs. vehicle simulation speed when following waypoints on a test course

In Figure 47, note the almost zero steady state error in speed response to the decreasing ramp desired speed during braking before hitting the first waypoint. (Approximate time is 15-20 seconds into the run.) The error almost decreases to zero by the time the vehicle reaches the turning speed before the waypoint. Again, slight overshoots are visible when reaching a constant desired velocity, but they are small enough to be acceptable. To better show this, the graph of velocity error is shown in Figure 48.

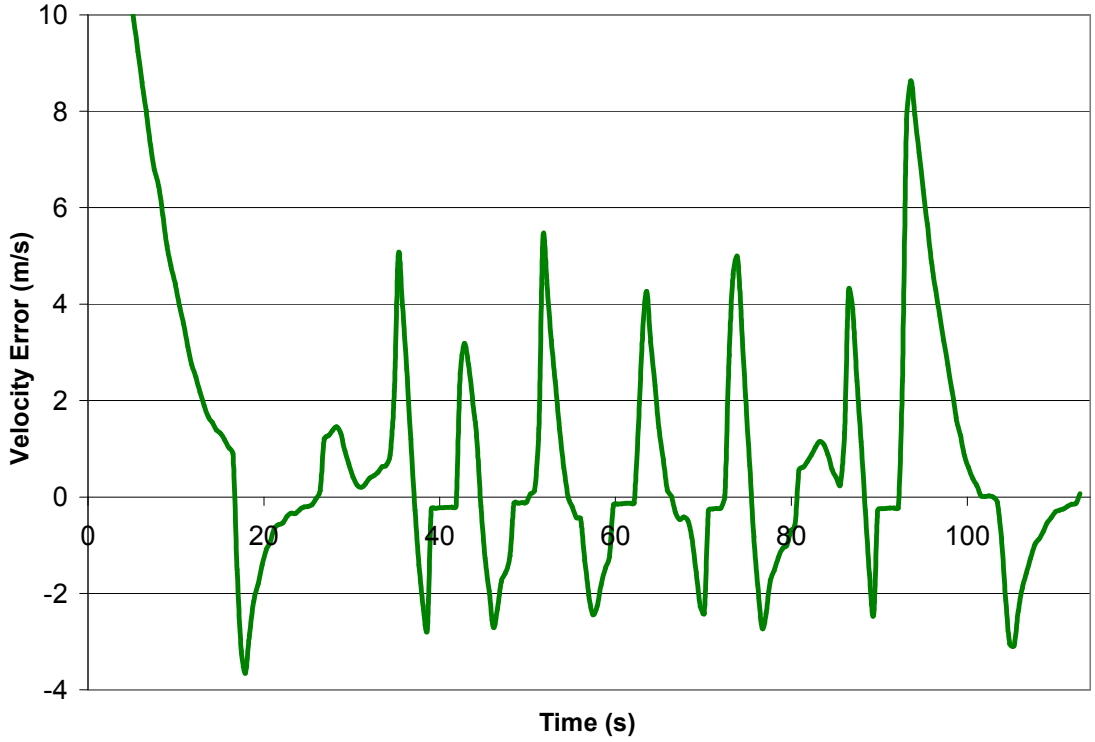

Figure 48: Graph of velocity error of actual vehicle vs. time

Analysis of the heading error and path error are more complicated. A slight steady state heading error does exist, which is due to the competition between the two errors, as well as the changing of the desired heading as the vehicle moves towards a waypoint. In Figure 49, a negative error creates a left turn and a positive error creates a right turn. Heading error is more dominant, as its gain is higher, and the path error gain is lowered as a function of heading error. Notice the noise in the heading error, which is on the order of 3 degrees. GPS heading is noisy because it is calculated from changes in location. Any noise in the position cause much bigger errors in the heading. As mentioned before, the output to the steering servo must be filtered to avoid damaging the gears with constant vibration. The path error is inherently smooth, as large position errors are needed to cause noise in it.

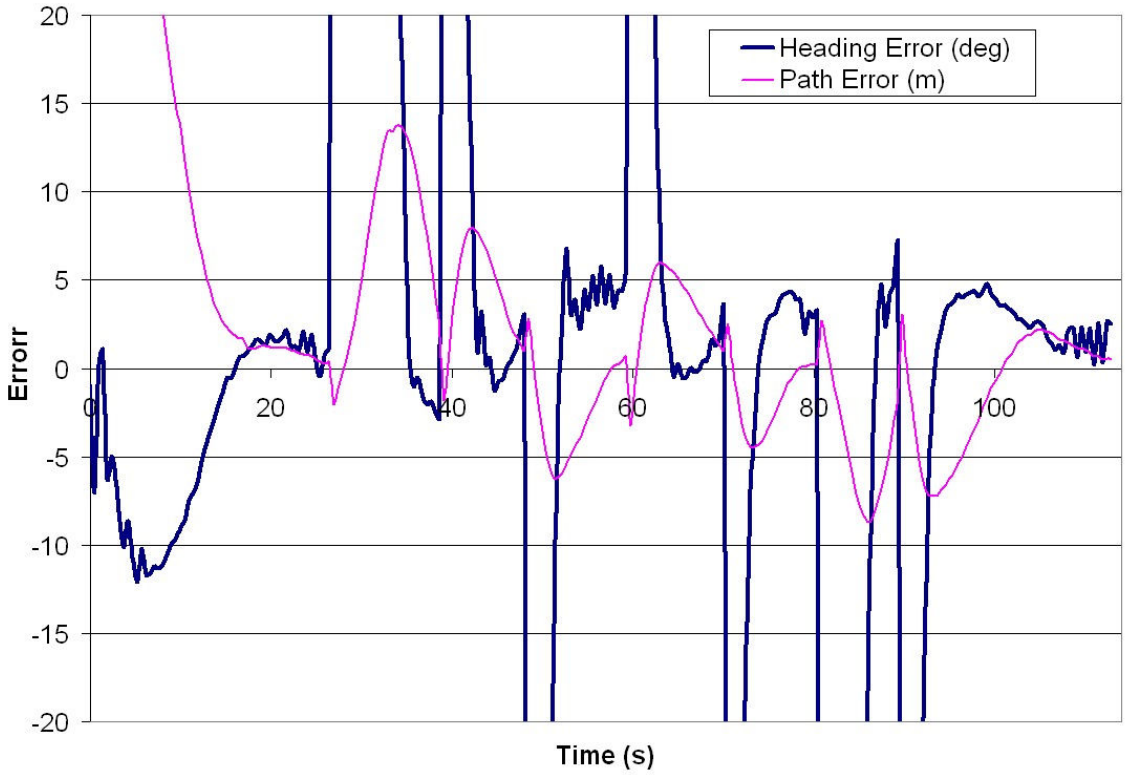

Figure 49: Graph of path and heading error of actual vehicle vs. time

As discussed before, the control gains used in the actual truck were much lower than what was designed and used in simulation. The simulation accurately predicts the path of the truck using the low gains that were tuned while testing on the truck. However, the simulation does not begin to oscillate as the truck does when the gains are raised. This could be due to a string of delays in the GPS, computer computations, D/A conversion, and filtering required to smooth heading noise. The time it takes the steering wheel to turn to the desired position is modeled in the simulation, but everything else is assumed to be instantaneous. If a 0.35 second delay is placed on the state feedback in the simulation, it begins to mimic the truck's decreased performance at higher gains. See Figure 50 for the path of the simulated vehicle once the delay is added. The gains used here were those designed in the 'Controller Design' section. If a delay is added to the simulation while using low gains, the path does not change at all, and is the same as the path in Figure 46.

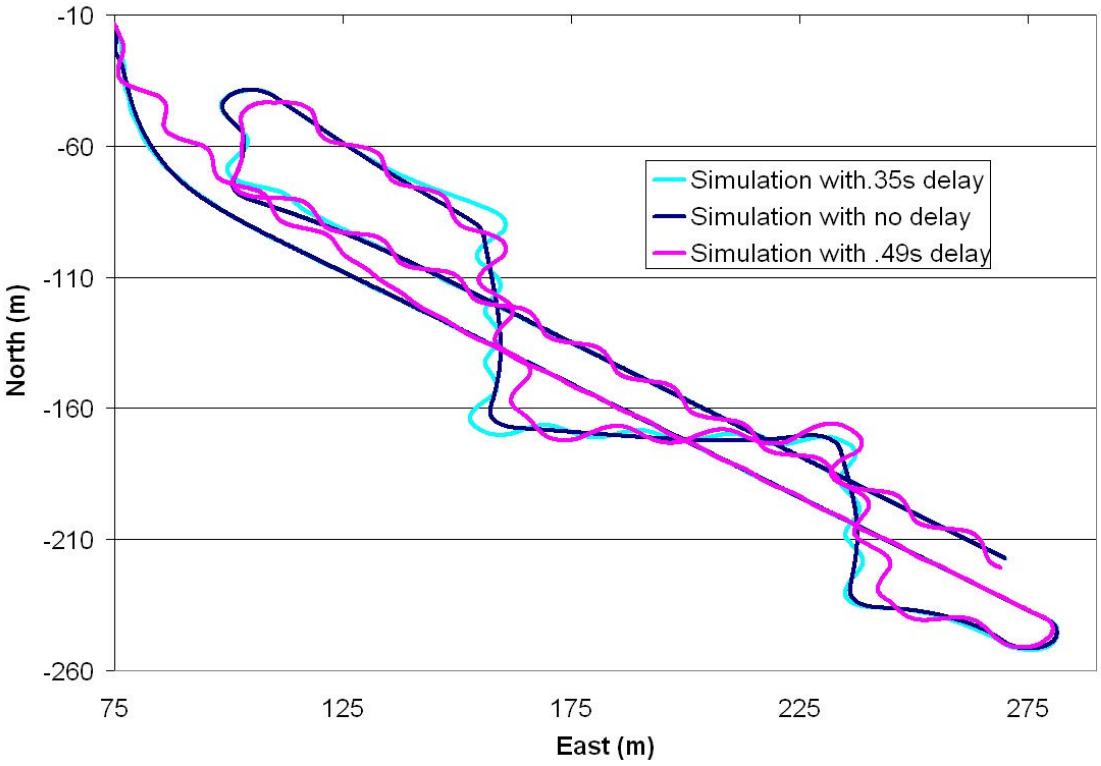

Figure 50: Effect of feedback delay in waypoint simulation

CONCLUSION AND RECOMMENDATIONS

The Texas A&M Autonomous Ground Vehicle is currently capable of completing a course comprised of any combination of waypoints. The heuristics which decide desired states of the vehicle do so in a way that mimics the reactions of a human driver, while adding increased precision and repeatability. The control software has also been proven to track constant heading inputs, which will come directly from obstacle avoidance software that is currently being developed.

The simulator can accurately compute the path of the vehicle, both in closed loop and open loop scenarios, so that new algorithms and programs can be quickly tested. The software is set up so that it can be used either in simulation or on the test track by simply selecting from a menu upon startup.

The following recommendations for future work are based on my experience with this project:

- Test the control and waypoint following algorithms in a variety of off road situations. Observe whether an integrator is needed in the heading control scheme to overcome un-modeled disturbances, such as off-chamber terrain.
- Replace the current DAQ card with a better model. The current card creates chatter in the output which must be smoothed by an RC filter before input in the EMC drive-by-wire system.
- The algorithms developed in this thesis can be combined with future obstacle avoidance software to create a better interface between the two software components. The current setup involves a binary switch when the obstacle avoidance software decides the desired states that the controller will track. In a dynamic off-road environment it is difficult to determine when to begin avoiding obstacles, as small adjustments in the vehicle's path are needed often.
- Incorporate a kinematics model that assumes no tire slip in the simulator to avoid the instability of the current vehicle dynamics model at low speeds.
- Anticipate change in direction at waypoint and swing wide beforehand to split path error before and after waypoint.

REFERENCES

1. DARPA, 28 Dec. 2005, "DARPA Grand Challenge," 3 Jan. 2006. http://www.darpa.mil/grandchallenge/overview.html.
2. Skyjacker, 2005, "Lift Kit-Suspension for a 2004 F-150 Pickup", 15 Jan. 2006. http://www.skyjacker.com/products-search-detail.asp?ID=69057&vyear=2004&vmake=FORD&vmodel=F-150%20PICKUP&vsubmod=BASE%20MODEL&partnumber=F4601KS.
3. Ford Motor Company, 2006, "F-150 Specifications", 15 Jan. 2006. http://www.fordvehicles.com/trucks/f150/features/specs/
4. Transformyx, 2006, "EMC Electronic Mobility Controls", 10 Oct. 2004. http://www.emc-digi.com/
5. Chapra, S. C., and Canale, R. P., 2002, "Numerical Methods for Engineers," 4th ed. New York, NY: McGraw-Hill.
6. Vehicle Dynamics Standards Committee, Jul. 1976, "Vehicle Dynamics Terminology," Warrendale, PA: Society of Automotive Engineers.
7. Gillespie, Thomas D., 1992, "Fundamentals of Vehicle Dynamics," Warrendale, PA: Society of Automotive Engineers.
8. Milliken, W. F., and Milliken, D. L., 1995, "Race Car Vehicle Dynamics," Warrendale, PA: Society of Automotive Engineers.
9. Franklin, G. F., Powell, J. D., and Emami-Naeini, A., 2002, "Feedback Control of Dynamics Systems," 4th ed. Upper Saddle River, NJ: Prentice Hall.
10. Wells, Caleb, 2006, "Software Flow Diagrams for the Texas A&M Autonomous Ground Vehicle", Unpublished TAMU AGV Project Report, Texas A&M University.
11. Palermo, Aaron, 2005, "Latitude/Longitude to Cartesian Coordinate System Transformations for the Texas A&M Autonomous Ground Vehicle", Unpublished TAMU AGV Project Report, Texas A&M University.

APPENDIX

Controller.c – The source file for the program.

```
#include "DAQ_Interface.h"
#include "gpsparse.h"
#include "controller.h"
#include <conio.h>
#include <time.h>
#include "RDDF.h"
#include "ini.h"

#define sleeptime 40
char q;
double steervolt,steervoltout, throttlevolt, steervolt_old ,
throttlevolt_old;
double realtime;
double dangle=0.0, totalerror=0.0;
int timer =0;
char *filename, *filenameext;
int stopBit;   //digital value (0 will make controller run 1 will stop
the controller)
FILE* output;
reading* GPSdata;
WP* waypoints;  //array of waypoints as returned by RDDF file
int numWP=0, i;
gains G;
float d[17];
int sim;

void init(void)
{
      glClearColor(217.0/256.0, 217.0/256.0, 217.0/256.0, 0.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(-590, 590, -462, 462);

      filename=(char *)malloc(75*sizeof(char));
      filenameext=(char *)malloc(75*sizeof(char));
      printf("Enter data log file name\n");
      scanf("%s",filename);
      printf("Do you want to sim? 1) Yes 2)No\n");
      scanf("%d",&sim);
      if(sim!=1)
            sim=0;
      readINI();
      stopBit=1;
      GPSdata=NULL;
      startSerial(getGPSstruct(),sim);

      //configPorts(0);
      //while(stopBit==1)
      //{
      //    getDigital1(&stopBit,0);
```

```
	//}

	readFromFile(&waypoints,&numWP);   //read in the RDDF file and
store the array
	setupControl(numWP,G = getGains());
	sprintf(filenameext,"%s.xls",filename);
	output = fopen(filenameext,"w");
	if(!output)
		MessageBox(NULL,"Error","Error opening log file",MB_OK);
	else
	{
		//fprintf(output,"Log File for Controller\n");

	fprintf(output,"pathP\tpathD\theadingP\theadingD\tvelP\tvelD\tcon
1\tcon2\tcon3\tcon4\tcon4\tcon5\tnumber of filtered\tSleep ms\n");

	fprintf(output,"%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%f\t%f\t%f\t%f\t%f\
t%f\t%u\t%u\t%f\n\n",ga._Kph2,Kdh2,ga._Kph,Kdh,Kpv,Kdv,con1,con2,con3,c
on4,con5,con6,filt,sleeptime,allowederror);

	fprintf(output,"timestamp\txtruck\tytruck\theading\tVx\tsteerangl
e\tdesired angle\theading error\tpath error\tpath
mult\tthrottle\tflagsteer\tflagvel\tdvelocity\tvelocity
error\tturnangle\tsafeturn\tslowdown\ttotalerror\txway\tyway\tlat\tlong
\tthrottlevolt\tsteervolt\n\n");
	}

	//To do:  GET ORIGIN
	q='a';
	steervolt=2.5;

	d[P1] = G._Kph;
	d[D1] = G._Kdh;
	d[P2] = G._Kph2;
	d[D2] = G._Kdh2;
	d[P] = G._Kpv;
	d[D] = G._Kdv;
	d[C1] = con1;
	d[C2] = con2;
	d[C3] = con3;
	d[C4] = con4;
	d[C5] = con5;
	d[C6] = con6;
	d[C7] = con7;
	d[C8] = con8;
	d[C9] = con9;
	d[C10] = con10;
	d[AE] = allowederror;

	for(i = 0; i < NUM_PIPES; i++)
	{
		pipe_handles[i] = CreateNamedPipe(
			pipe_names[i],             // pipe name
			PIPE_ACCESS_DUPLEX,       // read/write access
			PIPE_TYPE_MESSAGE |       // message type pipe
			PIPE_READMODE_MESSAGE |   // message-read mode
			PIPE_WAIT,                // blocking mode
```

```
                  PIPE_UNLIMITED_INSTANCES, // max. instances
                  BUFSIZE,                  // output buffer size
                  BUFSIZE,                  // input buffer size
                  NMPWAIT_USE_DEFAULT_WAIT, // client time-out
                  NULL);                    // default security
attribute

            if (pipe_handles[i] == INVALID_HANDLE_VALUE)
            {
                  printf("CreatePipe failed");
                  return;
            }
      }
      if(sim)
            writeMessage("Giving Control");
}

void run_truck(void)
{
      if(sim)
      {
            getData_block(&GPSdata);
      }
      else
      {
            // READ FROM GPS
            getData(&GPSdata);
            //make sure that GPS has updated once before running the
controller for the first time
            while(GPSdata==NULL)
                  getData(&GPSdata);
      }

      //SEND TO CONTROLLER
      if(GPSdata!=NULL)
      {

            control(&steervolt,&throttlevolt,GPSdata,waypoints, d,
&dangle, &totalerror);

            //SEND TO DAQ
            if (fabs(steervolt-steervolt_old)>.002) //only update daq
when a new voltage is required
            {
                  steervoltout=steervolt*1.042; //mult factor for RC
filter on 9-7-05 - if new daq card is bought, throw this out and
recalibrate
            setAnalog1((int)(steervoltout*1000),0);
            steervolt_old=steervolt;
            }
            if (fabs(throttlevolt-throttlevolt_old)>.002)
            {
            setAnalog2((int)(throttlevolt*1000),0);
            throttlevolt_old=throttlevolt;
            }
            realtime=clock();
            //LOG FILE
```

```
            if(timer==5) //changes log frequency - watch this!
        {

      fprintf(output,"%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%u\t%
u\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t\n",realtime,xtruck,
ytruck,heading*180/pi,Vx,steerangle*180/pi,dangle*180/pi,headingerror*1
80/pi,patherror,pathmultiplier,throttle,flagsteer,flagvel,dvelocity,vel
ocityerror,turnangle,safeturn,slowdown,totalerror,xway,yway,GPSdata-
>lat,GPSdata->lon,throttlevolt,steervolt);
            timer = 0;
        }
        else
        {
              timer++;
        }

      }
      //    getDigital1(&stopBit,0);
      //    if(stopBit==1)
      //          break;

      if(sim)
            writeMessage("Giving Control");
      else
            Sleep(sleeptime);


      glutPostRedisplay();
}

void set_color(color c)
{
      switch(c)
      {
      case black:
            glColor3f(0, 0, 0);
            break;
      case white:
            glColor3f(1, 1, 1);
            break;
      case red:
            glColor3f(204.0/256.0, 0, 1.0/256.0);
            break;
      case green:
            glColor3f(0, 128.0/256.0, 0);
            break;
      case blue:
            glColor3f(15.0/256.0, 15.0/256.0, 255.0/256.0);
            break;
      default:
            break;
      }
}

void display_string(int x, int y, char *string, color c)
{
      set_color(c);
```

```
    glRasterPos2f(x, y);
    while (*string) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, *string);
        string++;
    }
      set_color(black);
}

void display_large_string(int x, int y, char *string, color c)
{
      set_color(c);
    glRasterPos2f(x, y);
    while (*string) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *string);
        string++;
    }
      set_color(black);
}

void display_PDs(void)
{
      char string[50];

      display_string(0, 0, "Steering PD", black);

      sprintf(string, "Q) P1   %.4f", d[P1]);
      display_string(15, -20, string, black);

      sprintf(string, "W) D1   %.4f", d[D1]);
      display_string(15, -35, string, black);

      sprintf(string, "E) P2   %.4f", d[P2]);
      display_string(15, -50, string, black);

      sprintf(string, "R) D2   %.4f", d[D2]);
      display_string(15, -65, string, black);

      glPushMatrix();
            glTranslatef(100, 0, 0);
            display_string(0, 0, "Velocity PD", black);

            sprintf(string, "T) P   %.4f", d[P]);
            display_string(15, -20, string, black);

            sprintf(string, "Y) D   %.4f", d[D]);
            display_string(15, -35, string, black);
      glPopMatrix();
}

void display_Cs()
{
      char string[50];

      display_string(0, 0, "Desired Velocity Constants", black);

      sprintf(string, "1) C1   %.4f", d[C1]);
      display_string(0, -20, string, black);
```

```
	sprintf(string, "2) C2   %.4f", d[C2]);
	display_string(0, -35, string, black);

	sprintf(string, "3) C3   %.4f", d[C3]);
	display_string(0, -50, string, black);

	sprintf(string, "4) C4   %.4f", d[C4]);
	display_string(0, -65, string, black);

	sprintf(string, "5) C5   %.4f", d[C5]);
	display_string(0, -80, string, black);

	glPushMatrix();
		glTranslatef(100, 0, 0);

		sprintf(string, "6) C6   %.4f", d[C6]);
		display_string(0, -20, string, black);

		sprintf(string, "7) C7   %.4f", d[C7]);
		display_string(0, -35, string, black);

		sprintf(string, "8) C8   %.4f", d[C8]);
		display_string(0, -50, string, black);

		sprintf(string, "9) C9   %.4f", d[C9]);
		display_string(0, -65, string, black);

		sprintf(string, "0) C10   %.4f", d[C10]);
		display_string(0, -80, string, black);
	glPopMatrix();
}

void display_allowed_error(void)
{
	char string[50];

	sprintf(string, "P) Allowed Error   %.4f", d[AE]);
	display_string(0, 0, string, black);
}

void display_IMU(void)
{
	char string[50];

	display_string(0, 0, "IMU Output", black);

	sprintf(string, "Lat %.6f, N", GPSdata->lat);
	display_string(0, -15, string, black);

	sprintf(string, "Long %.6f, W", GPSdata->lon);
	display_string(0, -30, string, black);

	sprintf(string, "Heading %i", (int)(GPSdata-
>heading*RAD_TO_DEG));
	display_string(0, -45, string, black);
```

```
	sprintf(string, "Velocity %i mph", (int)(GPSdata-
>speed*2.23693629));
	display_string(0, -60, string, black);
}

void display_wheelspeed(void)
{
	display_string(0, 0, "Wheelspeed Output", black);
}

void display_throttle(void)
{
	int i;
	float percent = 125.0/100.0;
	char string[50];

	glColor3f(0, 0, 0);
	glBegin(GL_LINE_LOOP);
		glVertex2f(0, 0);
		glVertex2f(40, 0);
		glVertex2f(40, -250);
		glVertex2f(0, -250);
	glEnd();

	glBegin(GL_LINES);
		for(i = 0; i < 5; i++)
		{
			glVertex2f(-7, -62.5*i);
			glVertex2f(0, -62.5*i);
		}

		glVertex2f(0, -125);
		glVertex2f(40, -125);
	glEnd();

	glColor3f(204.0/250.0, 0, 1.0/250.0);
	glBegin(GL_QUADS);
		glVertex2f(2, -125);
		glVertex2f(38, -125);
		glVertex2f(38, -125 + (throttle*100*percent));
		glVertex2f(2, -125 + (throttle*100*percent));
	glEnd();

	display_string(50, -50, "Throttle", black);
	if (throttle < 0)
		sprintf(string, "%i %%", 0);
	else
		sprintf(string, "%i %%", (int)(throttle*100));
	display_string(50, -65, string, black);

	display_string(50, -180, "Brake", black);
	if (throttle > 0)
		sprintf(string, "%i %%", 0);
	else
		sprintf(string, "%i %%", (int)(-throttle*100));
	display_string(50, -195, string, black);
}
```

```
void draw_arrow(color c)
{
	set_color(c);
	glLineWidth(2);

	glBegin(GL_LINES);
		glVertex2f(0, 0);
		glVertex2f(87, 0);
	glEnd();

	glBegin(GL_TRIANGLES);
		glVertex2f(87, 4);
		glVertex2f(95, 0);
		glVertex2f(87, -4);
	glEnd();

	glLineWidth(1);
	set_color(black);
}

void display_heading_dial(void)
{
	int i;
	char string[50];

	glBegin(GL_LINE_LOOP);
		for(i = 0; i < 120; i++)
			glVertex2f(100*sin(3*DEG_TO_RAD*i),
100*cos(3*DEG_TO_RAD*i));
	glEnd();

	for(i = 0; i < 8; i++)
	{
		glPushMatrix();
			glRotatef(45*i, 0, 0, 1);
			glBegin(GL_LINES);
				glVertex2f(100, 0);
				glVertex2f(115, 0);
			glEnd();
		glPopMatrix();

		glPushMatrix();
			glTranslatef(-10, -5, 0);
			glRotatef(45*i, 0, 0, 1);
			sprintf(string, "%i", 45*i);
			display_string(130, 0, string, black);
		glPopMatrix();
	}

	glPushMatrix();
		glRotatef(heading*RAD_TO_DEG, 0, 0, 1);
		draw_arrow(blue);
	glPopMatrix();

	glPushMatrix();
		glRotatef(dangle*RAD_TO_DEG, 0, 0, 1);
```

```
            draw_arrow(red);
      glPopMatrix();

      display_string(-95, -155, "Blue - Actual Heading", black);
      sprintf(string, "%.2f", (float)(heading*RAD_TO_DEG));
      display_string(60, -155, string, blue);

      display_string(-95, -170, "Red - Desired Heading", black);
      sprintf(string, "%.2f", (float)(dangle*RAD_TO_DEG));
      display_string(60, -170, string, red);

      display_string(-95, -200, "Path Error Distance", black);
      sprintf(string, "%.2f m", patherror);
      display_string(60, -200, string, green);
}

void display_steering_dial(void)
{
      int i;
      char string[50];

      glBegin(GL_LINE_LOOP);
            for(i = 0; i < 120; i++)
                  glVertex2f(100*sin(3*DEG_TO_RAD*i),
100*cos(3*DEG_TO_RAD*i));
      glEnd();

      for(i = 0; i <= 3; i++)
      {
            glPushMatrix();
                  glRotatef(90 + 45*i, 0, 0, 1);
                  glBegin(GL_LINES);
                        glVertex2f(100, 0);
                        glVertex2f(115, 0);
                  glEnd();
            glPopMatrix();

            glPushMatrix();
                  glTranslatef(-5, -5, 0);
                  glRotatef(90 + 45*i, 0, 0, 1);
                  sprintf(string, "%i", 10*i);
                  display_string(130, 0, string, black);
            glPopMatrix();
      }

      for(i = 1; i <= 3; i++)
      {
            glPushMatrix();
                  glRotatef(90 - 45*i, 0, 0, 1);
                  glBegin(GL_LINES);
                        glVertex2f(100, 0);
                        glVertex2f(115, 0);
                  glEnd();
            glPopMatrix();

            glPushMatrix();
                  glTranslatef(-5, -5, 0);
```

```
                glRotatef(90 - 45*i, 0, 0, 1);
                sprintf(string, "%i", -10*i);
                display_string(130, 0, string, black);
            glPopMatrix();
        }

        glPushMatrix();
            glRotatef(90 + 4.5*steerangle*RAD_TO_DEG, 0, 0, 1);
            draw_arrow(black);
        glPopMatrix();

        display_large_string(-55, -25, "Steering Angle", black);
        sprintf(string, "%i", (int)(steerangle*RAD_TO_DEG));
        display_large_string(-10, -45, string, black);
}

void display_control_text(void)
{
        char string[50];

        display_large_string(0, 0, "Count Down to Start", black);

        display_large_string(0, -25, "Time    sec", black);

        display_large_string(20, -70, "Total Distance Error (m)", black);

        sprintf(string, "%.4f", totalerror);
        display_large_string(65, -100, string, red);

        sprintf(string, "Waypoint Number - %i", wp);
        display_string(50, -130, string, black);
}

void display_avoiding_LED(void)
{
        if (avoiding_flag == TRUE)
            set_color(red);
        else
            set_color(green);

        glBegin(GL_QUADS);
            glVertex2f(10, 10);
            glVertex2f(-10, 10);
            glVertex2f(-10, -10);
            glVertex2f(10, -10);
        glEnd();
        set_color(black);
        glBegin(GL_LINE_LOOP);
            glVertex2f(10, 10);
            glVertex2f(-10, 10);
            glVertex2f(-10, -10);
            glVertex2f(10, -10);
        glEnd();

        display_string(15, -5, "Collision Avoidance", black);
}
```

```
void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);

	glPushMatrix();
		glTranslatef(350, -350, 0);
		display_PDs();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(350, -230, 0);
		display_Cs();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(350, -190, 0);
		display_allowed_error();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(205, -245, 0);
		display_IMU();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(205, -350, 0);
		display_wheelspeed();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(80, -180, 0);
		display_throttle();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(180, 60, 0);
		display_heading_dial();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(180, 320, 0);
		display_steering_dial();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(325, 420, 0);
		display_control_text();
	glPopMatrix();

	glPushMatrix();
		glTranslatef(350, 150, 0);
		display_avoiding_LED();
	glPopMatrix();

	glutSwapBuffers();
	glFlush();
}
```

```
void key_intercept(unsigned char key, int x, int y)
{
	static int i = -99;

	switch(key)
	{
	case '1': i = C1; break;
	case '2': i = C2; break;
	case '3': i = C3; break;
	case '4': i = C4; break;
	case '5': i = C5; break;
	case '6': i = C6; break;
	case '7': i = C7; break;
	case '8': i = C8; break;
	case '9': i = C9; break;
	case '0': i = C10; break;
	case 'q': case 'Q': i = P1; break;
	case 'w': case 'W': i = D1; break;
	case 'e': case 'E': i = P2; break;
	case 'r': case 'R': i = D2; break;
	case 't': case 'T': i = P; break;
	case 'y': case 'Y': i = D; break;
	case 'p': case 'P': i = AE; break;
	default: break;
	}

	if (i != -99)
	{
		switch(key)
		{
		case 'a': case 'A': d[i] += 0.1; break;
		case 'z': case 'Z': d[i] -= 0.1; break;
		case 's': case 'S': d[i] += 0.01; break;
		case 'x': case 'X': d[i] -= 0.01; break;
		case 'd': case 'D': d[i] += 0.001; break;
		case 'c': case 'C': d[i] -= 0.001; break;
		case 'f': case 'F': d[i] += 0.0001; break;
		case 'v': case 'V': d[i] -= 0.0001; break;
		}
	}
}

void close_window(void)
{
	int i;

	fclose(output);
	endSerial();

	for(i = 0; i < NUM_PIPES; i++)
	{
		FlushFileBuffers(pipe_handles[i]);
		DisconnectNamedPipe(pipe_handles[i]);
		CloseHandle(pipe_handles[i]);
	}
}
```

```
void main(void)
{
	atexit(close_window);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowPosition(45, 30);
	glutInitWindowSize(1180, 924);
	glutCreateWindow("Autonomous Ground Vehicle");

	init();
	glutDisplayFunc(display);
	glutIdleFunc(run_truck);
	glutKeyboardFunc(key_intercept);

	glutMainLoop();
}
```

Controller.h – This file in the main controller file, which does the majority of computations.

```
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include <GL/glut.h>
#include <time.h>
#include <stdlib.h>
#include <winbase.h>

#include "pipe.h"

#define NS_TO_S 0.0000001
#define radius 6366564.864
#define pi 3.14159265359
#define MPHtoMPS .44704
#define RAD_TO_DEG 57.2957795
#define TRUE 1
#define FALSE 0
#define acmax .3*9.81   //max turning acceleration
#define velmax 17.88     //Max overall speed m/s (40_mph)

//Gains
#define Kpv ga._Kpv
#define Kdv ga._Kdv
#define Kph ga._Kph
#define Kdh ga._Kdh
#define Kph2 ga._Kph2
#define Kdh2 ga._Kdh2

#define steeringSat .61085
#define a 1.55
#define b 1.65
#define filt 5

//time_step stuff
LARGE_INTEGER ticksPerSecond;
LARGE_INTEGER tick;   // A point in time
LARGE_INTEGER start_ticks, end_ticks, cputime;

double velocityerrorold,velocityerror_intergral,Kiv, headingerrorold,
patherrorold,patherror,dangle,
headingerror,velocityerror,totalerror,pathmultiplier;
double heading_gps_old,initalsmallesterror,smallesterror,Kph_new;
double xway,yway,safesteer,safeturn,turnangle,slowdown,dt;
double
steerangle,tempsteerangle,oldsteerangle,steerangle_trunc,throttle,oldth
rottle,xtruck, ytruck,Vx,heading,dvelocity;
int wp,numways;
int flagvel,flagsteer, arrived_flag = FALSE, avoiding_flag;
double* throttle_av;
double* steerangle_av;
```

```
float con1 = 4.7611, con2 = -0.576, con3 = 5, con4 = 0.1802, con5 = 3,
con6 = -0.4911, allowederror = 4; //allowed error to 'hit' waypoint
float con7 = .1, con8 = -.8519, con9 = 0, con10 = 0;
gains ga;

//prototypes
void control(double*,double*,reading*,WP*,float*,double*,double*);
void changeToCartisian(double,double, double *,double*,double ,double);
void setupControl(int,gains);
void copy_maneuver_buffer(TCHAR*, Maneuver_Vector*);

typedef enum {black, white, red, green, blue} color;
typedef enum {P1, D1, P2, D2, P, D, C1, C2, C3, C4, C5, C6, C7, C8, C9,
C10, AE} variables;

void control(double* steervolt, double* throttlevolt,reading* GPS,WP*
array, float *d, double *dheading, double *total_distance_error)
{
      double xway2, yway2, xerror, yerror;
      double aa, bb, cc;
      int i;
      double aaa,bbb,ccc,xway0,yway0,theta1,patherrorsign;
      double heading_gps, Vx_gps, xtruck_gps, ytruck_gps;
      BOOL fSuccess;
      DWORD cbWritten, cbRead;
      TCHAR chBuf[BUFSIZE];
      Vehicle_State_Vector *vehicle = (Vehicle_State_Vector*)
malloc(sizeof(Vehicle_State_Vector));
      Maneuver_Vector *maneuver = NULL;

      ga._Kph= d[P1];
      ga._Kdh = d[D1];
      ga._Kph2 = d[P2];
      ga._Kdh2 = d[D2];
      ga._Kpv = d[P];
      ga._Kdv = d[D];
      con1 = d[C1];
      con2 = d[C2];
      con3 = d[C3];
      con4 = d[C4];
      con5 = d[C5];
      con6 = d[C6];
      con7 = d[C7];
      con8 = d[C8];
      con9 = d[C9];
      con10 = d[C10];
      allowederror = d[AE];


      //changing RDDF to cartisian
      changeToCartisian(array[wp].lat,array[wp].lon,&yway,&xway,array[0
].lat,array[0].lon);        //current waypoint
      changeToCartisian(array[wp+1].lat,array[wp+1].lon,&yway2,&xway2,a
rray[0].lat,array[0].lon);  //next waypoint
      changeToCartisian(array[wp-1].lat,array[wp-
1].lon,&yway0,&xway0,array[0].lat,array[0].lon);       //previous
waypoint
```

```
      //change gps lat long to cartisian
      changeToCartisian(GPS->lat,GPS-
>lon,&ytruck_gps,&xtruck_gps,array[0].lat,array[0].lon);

      Vx_gps= GPS->speed;
      heading_gps = GPS->heading; //when parceded, it converts to SAE
coordinates in radians


      if (heading_gps != heading_gps_old) //reassigns state variables
if the gps reads a new value, xtruck_gps_old initalized at 0
      {                                                    //if
xtruck_gps=0, then wont get undefined error b/c all vehicle states are
initalized at 0 also
            xtruck=xtruck_gps;
            ytruck=ytruck_gps;
            Vx=Vx_gps;
            heading=heading_gps;

      }
      //CAC stuff
                  vehicle->x = xtruck;                              /*
update the vehicle      */
                  vehicle->y = ytruck;                              /*
      state vector            */
                  vehicle->heading = heading;
                  vehicle->pitch = 0;
                  vehicle->roll = 0;
                  vehicle->speed = Vx;
                  vehicle->start_x = xway0;
                  vehicle->start_y = yway0;
                  vehicle->goal_x = xway;
                  vehicle->goal_y = yway;
                  fSuccess = WriteFile(pipe_handles[VEHICLE_PIPE],
vehicle, sizeof(Vehicle_State_Vector), &cbWritten, NULL);
                  fSuccess = WriteFile(pipe_handles[ARRIVED_PIPE],
&arrived_flag, sizeof(int), &cbWritten, NULL);
                  do
                  {
                        fSuccess =
ReadFile(pipe_handles[AVOIDING_PIPE], chBuf, sizeof(int), &cbRead,
NULL);
                        if (!fSuccess && GetLastError() !=
ERROR_MORE_DATA)
                              break;
                  }
                  while (!fSuccess);
                  avoiding_flag = chBuf[0];

                  if (avoiding_flag == TRUE)
                  {
                        do
                        {
                              fSuccess =
ReadFile(pipe_handles[MANEUVER_PIPE], chBuf, sizeof(Maneuver_Vector),
&cbRead, NULL);
```

```
                              if (!fSuccess && GetLastError() !=
ERROR_MORE_DATA)
                                    break;
                        }
                        while (!fSuccess);
                        copy_maneuver_buffer(chBuf, maneuver);
                  }
                  else
                        maneuver = NULL;

      //get the x error, y error and total error
      xerror=xway-xtruck;
      yerror=yway-ytruck;
      totalerror=sqrt(xerror*xerror+yerror*yerror);

      //when "close" to waypoint, head towards the next one
      if( (totalerror <allowederror) && (wp<numways) )//doesnt advance
to non-exsistant wp in RDDF
      {
            wp++;
            smallesterror=initalsmallesterror; //reset missed wp
initalerror
      }
      else //if the total error increases by 1 m after going with 10 m
of the waypoint, then the waypoint is missed
      {

            if (totalerror<=initalsmallesterror)
            {
                  if ((totalerror-smallesterror)>.5)
                  {
                        printf("missed wp number %u\n",wp);
                        wp++;
                        smallesterror=initalsmallesterror;

                  }
                  else
                  {
                        if (totalerror<smallesterror)
                        smallesterror=totalerror;
                  }
            }
      }

      /////////////////Begin Steering Control//////////////
      //Calculate angle
      aaa=sqrt((xway-xway0)*(xway-xway0)+(yway-yway0)*(yway-yway0));
      bbb=sqrt((xway-xtruck)*(xway-xtruck)+(yway-ytruck)*(yway-
ytruck));
      ccc=sqrt((xtruck-xway0)*(xtruck-xway0)+(ytruck-yway0)*(ytruck-
yway0));
      theta1=acos((bbb*bbb-ccc*ccc-aaa*aaa)/(-2*ccc*aaa));
      //mag of patherror
      patherror=ccc*sin(theta1);
      if (fabs(aaa-bbb-ccc)<.0000001)
      {
            patherror=.0000001;
```

```
	}

	//cross product
	patherrorsign=((xway-xway0)*(ytruck-yway0)-(xtruck-xway0)*(yway-
yway0));
	if (patherrorsign<0)
	{
		patherror=-patherror;
	}
	//If don't want to track path on first waypoint
	/*if(wp==1)
	{
		patherror=0;
		patherrorold =0;
	}*/
	//desired angle calculation
	dangle=atan2(yerror,xerror);
	if (dangle <0)
	{
		dangle=dangle+2*pi;
	}

	if (maneuver)
	{
		dangle = maneuver->new_heading;
		patherror = 0;
	}
	//negative feedback
	headingerror=dangle-heading;

	//correction for errors greater than 180
	if (headingerror > pi)
	{
		headingerror=headingerror-2*pi;
	}
	if (headingerror < -pi)
	{
		headingerror=headingerror+2*pi;
	}

	//path multiplier calculation
	if (headingerror<=(10*pi/180))
	{
		pathmultiplier=1.0;
	}
	else if (headingerror>(80*pi/180))
	{
		pathmultiplier=0.0;
	}
	else
	{
		pathmultiplier=(headingerror-80*pi/180)*(-1/(70*pi/180));
	}

	//Finding loop duration for derivative gains only if GPS give a
new location
	if (heading_gps != heading_gps_old)
```

```
            {
            QueryPerformanceFrequency(&ticksPerSecond); //gets CPU
speed
            QueryPerformanceCounter(&end_ticks);
            //Finds time step in seconds
            cputime.QuadPart = end_ticks.QuadPart-
start_ticks.QuadPart;

      dt=((float)cputime.QuadPart/(float)ticksPerSecond.QuadPart);
            //printf("\ntime step = %f",dt); //prints loop timing for
testing
            }
      else
            printf("\n   Skipped loop");

      //heading gains as a function of velocity
      Kph_new=Kph*pow(abs(Vx),-.8);
      if (Kph_new>5) //saturate at 5 for low velocities
            Kph_new=5.0;
      if (Kph_new<0.2) //saturate at .2 for high velocities
            Kph_new=0.2;

      //steer control
      steerangle=headingerror*Kph_new+patherror*Kph2*pathmultiplier+(he
adingerror-headingerrorold)/(double)(dt)*Kdh+(patherror-
patherrorold)/(double)(dt)*Kdh2;

      // steering staturation
      if (steerangle > steeringSat)
      {
            steerangle=steeringSat;
      }
      if (steerangle < -steeringSat)
      {
            steerangle=-steeringSat;
      }

      flagsteer=0; //reseting steering algorithm flag

      //safe steer saturation
      safesteer=(acmax*(a+b))/(Vx*Vx);

      if (steerangle > safesteer)
      {
            steerangle=safesteer;
            flagsteer=1;
      }

      if (steerangle < -safesteer)
      {
            steerangle=-safesteer;
            flagsteer=1;
      }


      //////////////Begin Velocity Control/////////////////
```

```
	///Turn Angle Calculator

	aa=sqrt((xway2-xtruck)*(xway2-xtruck)+(yway2-ytruck)*(yway2-
ytruck));
	bb=sqrt((xway-xway2)*(xway-xway2)+(yway-yway2)*(yway-yway2));
	cc=sqrt((xtruck-xway)*(xtruck-xway)+(ytruck-yway)*(ytruck-yway));
	turnangle=pi-acos((aa*aa-bb*bb-cc*cc)/(-2*bb*cc));

	//safe turning speed
	safeturn=con1*pow(turnangle,con2)+fabs((totalerror-con3)*con4);

	//darpa max speed
	//dvelocity=wpdata[wp][2];
	dvelocity = array[wp].MPH*MPHtoMPS;

	flagvel=0; //reseting velocity algorithm flag

	//picks smallest of the two
	if (safeturn<dvelocity)
	{
		dvelocity=safeturn;
		flagvel=1;
	}

	//Lowering velocity with steer angle
	tempsteerangle=steerangle;
	if (tempsteerangle<0)
		tempsteerangle*=-1;
	slowdown=con5*pow((tempsteerangle),con6);

	//picks smallest
	if (slowdown<dvelocity)
	{
		dvelocity=slowdown;
		flagvel=2;
	}

	velocityerror=dvelocity-Vx;

	//throttle control///////
	//Intergral gain calucation
	velocityerror_intergral=velocityerror_intergral+velocityerror*dt;
	//Intergral windup protection
	if (velocityerror_intergral>5.0)
	velocityerror_intergral=5.0;
	if (velocityerror_intergral<-5.0)
	velocityerror_intergral=-5.0;

	Kiv=.04; //intergral gain, not in INI file

	throttle=velocityerror*Kpv+(velocityerror-
velocityerrorold)/dt*Kdv+velocityerror_intergral*Kiv;


	//storing old state errors for derivative gains, only when a new
GPS location is given
	if (heading_gps != heading_gps_old)
```

```
	{
		velocityerrorold=velocityerror;
		headingerrorold=headingerror;
		patherrorold=patherror;
		heading_gps_old=heading_gps;  //reassigns old gps to be
compared in this if statement on the next loop

		//Start loop timing for derivative gains
		QueryPerformanceCounter(&start_ticks);
	}

	// throttle staturation
	if (throttle > .70)
	{
		throttle=.7;
	}
	if (throttle < -.70)
	{
		throttle=-.7;
	}

	//steer angle filtering
			for(i=filt-1;i>0;i--)
				{
					steerangle_av[i]=steerangle_av[i-1];
				}
			steerangle_av[0]=steerangle;
			steerangle=0.0;
			for(i=0;i<filt;i++)
				{
					steerangle+=steerangle_av[i];
				}

			steerangle=(steerangle+steerangle_av[0])/(filt+1.);

	//calculation of steer voltage
	*steervolt=3.1334*steerangle+2.525;

	if(Vx<.1)                    //For slow speeds at takeoff,
doesnt jerk tires around
		*steervolt=2.525;

	//throttle filtering
			for(i=filt-1;i>0;i--)
				{
					throttle_av[i]=throttle_av[i-1];
				}
			throttle_av[0]=throttle;
			throttle=0.0;
			for(i=0;i<filt;i++)
				{
					throttle+=throttle_av[i];
				}
			throttle=(throttle+throttle_av[0])/(filt+1.);

	//calculation of throttle voltage
	*throttlevolt=1.211*throttle+2.699;
```

```
	//if we are at the last waypoint, stop and center wheel
	if (wp>=numways) //remeber matrix indice starts at 0
		{
			*steervolt=2.525; //straight ahead
			*throttlevolt=2.0; //-.2 brake
			//make stop log here--------------------------------
---------------------------------
			printf("\nYou have finished the course!");
			arrived_flag = TRUE;
		}

	//CAC stuff
	*dheading = dangle;
	*total_distance_error = totalerror;
	if (vehicle)
		free(vehicle);
	if (maneuver)
		free(maneuver);
	}


void changeToCartisian(double Lat,double Long,double *y,double
*x,double origLat,double origLong)
{
	//Changes lat/long to SAE cartisian coordinates
	*x = 2*radius * sin((Lat-origLat)*pi/180/2.);
	*y = 2*radius * sin((Long-
origLong)*pi/180/2.)*cos((Lat+origLat)*pi/180/2);
}

void setupControl(int numWPs,gains gain)
{
	int i;
	wp=1; //inital wp (the first wp is the origin)
	throttle_av=(double*)malloc(filt*sizeof(double));
	steerangle_av=(double*)malloc(filt*sizeof(double));
	ga = gain;
	//finds number of waypoints
	numways=numWPs;
	//initialize the previous error for PD controllers
	headingerrorold=0.0;
	velocityerrorold=0.0;
	patherror=0.0;
	patherrorold=0.0;
	heading_gps_old=0.0;
	xtruck=0.0;
	ytruck=0.0;
	Vx=0.0;
	heading=0.0;
	throttle=0.0;
	steerangle=0.0;
	slowdown = 0.;
	oldsteerangle=0.0;
	oldthrottle = 0.0;
	steerangle_trunc=0.0;
	initalsmallesterror=15.0;
```

```
	smallesterror=initalsmallesterror;
	QueryPerformanceCounter(&start_ticks); //inital tick time
for(i=0;i<filt;i++)
{
	throttle_av[i]=0.0;
	steerangle_av[i]=0.0;
}
}

void copy_maneuver_buffer(TCHAR *chBuf, Maneuver_Vector *maneuver)
{
	Maneuver_Vector *temp_maneuver = (Maneuver_Vector*)chBuf;
	maneuver = (Maneuver_Vector*) malloc(sizeof(Maneuver_Vector));

	maneuver->dive_rate = temp_maneuver->dive_rate;
	maneuver->heading_rate = temp_maneuver->heading_rate;
	maneuver->new_depth = temp_maneuver->new_depth;
	maneuver->new_heading = temp_maneuver->new_heading;
}
```

DAQ_Interface.h

```
//Must have the lib file referenced by project
//Project->Properties->Linker->Command Line.... add C:/MCC/C/cbw32.lib
//39003

#include "cbw.h"
#include <stdio.h>


#define MIN_STEERING 0       //mins and maxes will be specified later
#define MAX_STEERING 4095
#define MIN_THROTTLE 0
#define MAX_THROTTLE 4095

//set port numbers
#define ANALOG_OUT_1 0
#define ANALOG_OUT_2 1
#define ANALOG_IN_1 0
#define ANALOG_IN_2 1
#define DIG_IN_1 0
#define DIG_IN_2 1
#define DIG_IN_3 2
#define DIG_IN_4 3
#define DIG_IN_5 4
#define DIG_IN_6 5
#define DIG_IN_7 6
#define DIG_IN_8 7
#define DIG_OUT_1 8
#define DIG_OUT_2 9
#define DIG_OUT_3 10
#define DIG_OUT_4 11
#define DIG_OUT_5 12
#define DIG_OUT_6 13
#define DIG_OUT_7 14
#define DIG_OUT_8 15


int config = 0;   //make sure all digital ports are configured


void checkConnect(int boardNum)
{
      cbFlashLED(boardNum);
}

/***************************
***************************
**      ANALOG SIGNALS    **
***************************
***************************/
void setAnalog1(int voltage,int boardNum)
{
      int check;
      if(voltage>4095)
            voltage=4095;
```

```
	if(voltage<0)
		voltage=0;
	check=cbAOut(boardNum,ANALOG_OUT_1,UNI4VOLTS,voltage);
	if(check!=0)
		printf("ERROR %d Refer to Documentation\n",check);

}

void getAnalog1(double* voltage,int boardNum)
{
	int check;
	unsigned short data;
	check = cbAIn(boardNum,ANALOG_IN_1,BIP10VOLTS,&data);
	if(check!=0)
	{
		printf("ERROR %d Refer to Documentation\n",check);
		*voltage = -9999;
	}
	else
		*voltage = (data-2048)/204.7;
}

void setAnalog2(int voltage, int boardNum)
{

	int check;
	if(voltage>4095)
		voltage=4095;
	if(voltage<0)
		voltage=0;
	check=cbAOut(boardNum,ANALOG_OUT_2,UNI4VOLTS,voltage);

	if(check!=0)
		printf("ERROR %d Refer to Documentation\n",check);
}


void getAnalog2(double* voltage,int boardNum)
{
	int check;
	unsigned short data;
	check = cbAIn(boardNum,ANALOG_IN_2,BIP10VOLTS,&data);
	if(check!=0)
	{
		printf("ERROR %d Refer to Documentation\n",check);
		*voltage = -9999;
	}
	else
		*voltage = (data-2048)/204.7;
}

int readThrottle()
{
	//the integers read in will correspond between 0 and 4.096 volts
outputed
	//from the analog outputs on the DAQ
```

```
	int read=0;
	int check = 0;
	printf("Enter an Integer between 0 and 4095  -->> ");
	while(check==0)
	{
		check = scanf("%d",&read);

	if((read>4095||read<0)&&(read>MAX_THROTTLE||read<MIN_THROTTLE))
			check=0;
		if(check==0)
		{
			fflush(stdin);
			printf("Try Again -->> ");
		}
	}
	return read;
}

int readSteering()
{
	//the integers read in will correspond between 0 and 4.096 volts
outputed
	//from the analog outputs on the DAQ

	int read=0;
	int check = 0;
	printf("Enter an Integer between 0 and 4095  -->> ");
	while(check==0)
	{
		check = scanf("%d",&read);

	if((read>4095||read<0)&&(read>MAX_STEERING||read<MIN_STEERING))
			check=0;
		if(check==0)
		{
			fflush(stdin);
			printf("Try Again -->> ");
		}
	}
	return read;
}

/**************************
**************************
**     DIGITAL SIGNALS    **
**************************
**************************/

void configPorts(int boardNum)
{
	int check;
	config=1;
	//get will be on port 0-7
	check = cbDConfigPort(boardNum,FIRSTPORTA,DIGITALIN);
	if(check==0)
	{
		printf("\tDigital Ports 0-7 configured\n");
```

```
	}
	else
	{
		printf("\tERROR %d cannot configure ports 0-7\n",check);
		config=0;
	}
	//set will correspond to ports 0-7 by referencing 8-15 (subtract
8 to get corresponding port)
	check = cbDConfigPort(boardNum,FIRSTPORTB,DIGITALOUT);
	if(check==0)
	{
		printf("\tDigital Ports 8-15 configured\n");
	}
	else
	{
		printf("\tERROR %d cannot configure ports 8-15\n",check);
		config=0;
	}
}


/*GET SIGNALS*/

void getDigital1(int* value,int boardNum)
{
	int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_1,&bit);
	if(check!=0)
	{
		printf("\tError %d - Check Documentation\n",check);
	}
	else
		printf("\tBit Value - %d\n",(int)bit);
}


void getDigital2(int* value,int boardNum)
{
		int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_2,&bit);
	if(check!=0)
	{
		printf("Error %d - Check Documentation",check);
	}
	else
		printf("Bit Value - %d",(int)bit);
}


void getDigital3(int* value,int boardNum)
{
		int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_3,&bit);
	if(check!=0)
```

```
	{
		printf("Error %d - Check Documentation",check);
	}
	else
		printf("Bit Value - %d",(int)bit);
}


void getDigital4(int* value,int boardNum)
{
		int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_4,&bit);
	if(check!=0)
	{
		printf("\tError %d - Check Documentation\n",check);
	}
	else
		printf("\tBit Value - %d\n",(int)bit);
}


void getDigital5(int* value,int boardNum)
{
		int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_5,&bit);
	if(check!=0)
	{
		printf("\tError %d - Check Documentation\n",check);
	}
	else
		printf("\tBit Value - %d\n",(int)bit);
}



void getDigital6(int* value,int boardNum)
{
		int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_6,&bit);
	if(check!=0)
	{
		printf("\tError %d - Check Documentation\n",check);
	}
	else
		printf("\tBit Value - %d\n",(int)bit);
}


void getDigital7(int* value,int boardNum)
{
		int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_7,&bit);
	if(check!=0)
```

```
	{
		printf("\tError %d - Check Documentation\n",check);
	}
	else
		printf("\tBit Value - %d\n",(int)bit);
}

void getDigital8(int* value,int boardNum)
{
		int check;
	unsigned short bit;
	check = cbDBitIn(boardNum,FIRSTPORTA,DIG_IN_8,&bit);
	if(check!=0)
	{
		printf("Error %d - Check Documentation",check);
	}
	else
		printf("\tBit Value - %d\n",(int)bit);
}



/*SET SIGNALS*/
void setDigital1(short value,int boardNum)
{
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_1,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}

void setDigital2(short value,int boardNum)
{
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_2,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}

void setDigital3(short value,int boardNum)
{
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_3,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}

void setDigital4(short value,int boardNum)
{
```

```
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_4,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}

void setDigital5(short value, int boardNum)
{
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_5,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}

void setDigital6(short value, int boardNum)
{
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_6,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}

void setDigital7(short value,int boardNum)
{
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_7,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}

void setDigital8(short value, int boardNum)
{
	int check;
	check =cbDBitOut(boardNum,FIRSTPORTA,DIG_OUT_8,value);
	if(check!=0)
		printf("\tError %d - Check Documentation\n",check);
	else
		printf("\tSet Bit to %d\n",(int)value);
}
```

Gpsparse.h

```
/*To use you must set the code generation to multi-threaded and
the linker command line argument to include mpr.lib and wsock32.lib */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <process.h>
#include "structs.h"

#define DEG_TO_RAD .0174532925
#define KNOT_TO_MPS 0.514444444


HANDLE h;                                   //handle for serial I/O
HANDLE  RunMutex;                       /* "Keep Running" mutex */
HANDLE  ScreenMutex;                            /* "Screen update"
mutex  */
DCB dcb;
OVERLAPPED ov;
FILE *gps_Log_FILE;
int numThread;
reading data;
int flag;
char* comPort;
int baud;



//prototypes
void parse(char*,reading *);
double knotConversion(double);
double angleConversion(double);
void serialIO(char*);
void getData(reading**);  //stores new GPS value (non-blocking)
void getData_block(reading**);
void startSerial(GPS_Data,int);
double minToDeg(double);
void sim_serialIO();
void endSerial();
void writeMessage(char*);
void readData();  //just prints out the value

void writeMessage(char* sendString)
{
      //function writes sendString out of commPort at baud baud rate
      //a returned one is an error a 0 means a successful send
      //EX.  check = writeToSerial("Send This String","COM6",9600);


      int error;  //stores the error information
      DWORD write;  //stores the number of bytes written and the number
of bytes to write
```

```
	//create the timeouts



	//get the length of sendString
	if(sendString)
		write = (DWORD)strlen(sendString);
	else
		return;
	//write to the com port
	if(!WriteFile(h, sendString, write, &write, &ov))
	{
		error = GetLastError();
		if(error == ERROR_IO_PENDING)
		{
			//if not all bytes could be sent at once overlap
(this is blocking)
			if( !GetOverlappedResult(h, &ov, &write, TRUE) )
				printf("E003_GetOverlappedResult failed");
		}
		else
		{
			printf("Error %d occured while Writing to
file",error);
			return;
		}
	}
	else
		printf("E004_WriteFile failed");

}


void serialIO(char* a)
{
	//function to do serial I/O returns each string
	//most of these constants are defined in <windows.h>
	DWORD read;
	//	int error;
	char* buf1;
	int index=0;
	char* quit;
	int sflag = 0;  //test for the first string incoming
	int bflag = 0;	//flag to throw out garbage in initial run
	char* temp;
	//	char* writeToSerial;
	//	DWORD writeLen;

	//create a buffer
	buf1 = (char*)malloc(100*sizeof(char));
	temp = (char*)malloc(sizeof(char));
	quit = (char*)malloc(5*sizeof(char));
	//create handle reading in from COM1

	//writeToSerial = (char*)malloc(50*sizeof(char));
	//sprintf(writeToSerial,"log %s gprmca ontime .05",comPort);
	//writeLen = (DWORD)strlen(writeToSerial);
```

```
	//if(!WriteFile(h, writeToSerial, writeLen, &writeLen, &ov))
	//{
	//	error = GetLastError();
	//	if(error == ERROR_IO_PENDING)
	//	{
	//if not all bytes could be sent at once overlap (this is
blocking)
	//		if( !GetOverlappedResult(h, &ov, &writeLen, TRUE) )
	//			printf("E003_GetOverlappedResult failed");
	//	}
	//	else
	//	{
	//		printf("Error %d occured while Writing to
file",error);
	//	}
	//}
	//else
	//		printf("E004_WriteFile failed");

	//read in from buffer

	while((WaitForSingleObject( RunMutex, 0 ) == WAIT_TIMEOUT) )
	{
		read = 0;
		if(!ReadFile(h,temp, 1, &read,&ov))
		{
			if(GetLastError() == ERROR_IO_PENDING)   //read in
each character
				if(!GetOverlappedResult(h, &ov, &read, TRUE))
//blocking overlapped event
					printf("GetOverlappedResult failed");
		}
		if(temp[0]=='q')
			break;
		if(read==1)
		{
			//printf("%c\n",temp[0]);
			//read in 1 character
			//use the $ as a way to find a new line
			//temp is what stores the newly input data
			//buf1 is what stores each individual string
			if((temp[0]!='$'||sflag==0))  //must make sure not to
parse first string
			{
				if(temp[0]!='$'&&(sflag==0))
				{
					//if we are getting junk at first
					bflag=0;
				}
				else
				{
					bflag=1;
					buf1[index]=temp[0];
					index++;
				}
			}
			else
```

```
                  {
                        buf1[index]='\0';  //make buffer null
terminated

                        if(gps_Log_FILE)   //if the file creation was
successful, write to file
                              fprintf(gps_Log_FILE,"%s\n",buf1);

                        if(strlen(buf1)>5&&buf1[5]=='C')  //will get
only those with header $GPRMC
                              if(buf1[14]=='V')
                                    printf("Error in Input Data,
Skipping String"); //if GPS returns Error
                              else
                              {
                                    //printf("\n%s\n",buf1);
                                    flag = 1;
                                    //printf("%s",buf1);
                                    parse(buf1,&data);

                              }

                              index=1;
                              free(buf1);
                              buf1=(char*)malloc(100*sizeof(char));
                              buf1[0]=temp[0];

                  }
                  sflag=1;
            }
      }

}


void startSerial(GPS_Data g,int sim)
{
      int timeout_flag = 0;  //checks for CommMask and timeouts
      COMMTIMEOUTS cto = {0, 0, 0, 0, 0};
      numThread = 0;
      baud = g.GPS_baud;
      comPort = g.GPS_COM;
      flag = 0;



      gps_Log_FILE = fopen("GPS_Logfile.txt","w");
      if(!gps_Log_FILE)
      {
            MessageBox(NULL,"Error","Error opening file for
output",MB_OK);
      }


      //create a buffer

      //create handle reading in from COM1
```

```
	do{
		timeout_flag=0;
		h = CreateFile(comPort, GENERIC_READ|GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, FILE_FLAG_OVERLAPPED,NULL);
		if(h != INVALID_HANDLE_VALUE)
		{
			printf("Opened %s\n",comPort);
		}
		else
		{
			printf("Failed to open %s trying again\n",comPort);
			timeout_flag=1;
			FlushFileBuffers(h);
			CloseHandle(h);
		}

		if(!SetCommTimeouts(h, &cto)&&timeout_flag==1)
		{
			printf("SetCommTimeouts failed trying again\n");
			timeout_flag=1;
			CloseHandle(h);
		}
		//set up the COM port
		memset(&dcb, 0, sizeof(dcb));
		GetCommState(h, &dcb);
		dcb.DCBlength = sizeof(dcb);
		dcb.BaudRate = baud;
		dcb.fBinary = 1;
		dcb.fDtrControl = DTR_CONTROL_ENABLE;
		dcb.fRtsControl = RTS_CONTROL_ENABLE;
		dcb.Parity = NOPARITY;
		dcb.StopBits = ONESTOPBIT;
		dcb.ByteSize = 8;

		ZeroMemory(&ov, sizeof(ov));

		//overlapped events
		ov.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
		if(ov.hEvent == INVALID_HANDLE_VALUE&&timeout_flag==0)
		{
			timeout_flag=1;
			printf("CreateEvent failed");
		}
		//set Comm Mask events
		if(!SetCommMask(h, EV_RXCHAR)&&timeout_flag==0)
		{
			printf("SetCommMask failed Trying Again\n");
			timeout_flag=1;
			CloseHandle(h);

		}
	}while(timeout_flag==1);

	FlushFileBuffers(h);
```

```
	if(!sim)
	{
		ScreenMutex = CreateMutex( NULL, FALSE, NULL );	/*
Cleared */
		RunMutex = CreateMutex( NULL, TRUE, NULL );	/* Set */
		_beginthread(serialIO,0,NULL);
		numThread++;
	}

}

void endSerial()
{
	int writeLen = 9;
	//	int error;
	/*
	if(!WriteFile(h, "unlogall", writeLen, &writeLen, &ov))
	{
	error = GetLastError();
	if(error == ERROR_IO_PENDING)
	{
	//if not all bytes could be sent at once overlap (this is
blocking)
	if( !GetOverlappedResult(h, &ov, &writeLen, TRUE) )
	printf("E003_GetOverlappedResult failed");
	}
	else
	{
	printf("Error %d occured while Writing to file",error);
	}
	}
	else
	printf("E004_WriteFile failed");
	*/

	while ( numThread > 0 )
	{
		/* Tell thread to die and record its death. */
		ReleaseMutex( RunMutex );
		numThread--;
	}
	/* Clean up display when done */
	CloseHandle(h);
	WaitForSingleObject( ScreenMutex, INFINITE );


}


void getData(reading** newRead)
{
	//if the flag accepts it, the new data is transferred over
	if(flag==1)
	{
		flag=0;

	//printf("Lattitude\tLongitude\n%f\t%f\n",data.lat,data.lon);
```

```
            *newRead = &data;
      }
}

void getData_block(reading** newRead)
{
      printf("Blocking");
      sim_serialIO();
      *newRead = &data;
}


void readData()
{
      printf("Trying to Read Data -->  flag = %d\n",flag);
      if(flag==1)
      {
            flag=0;
            //printf("Lattitude - %f\n",data.lat);
            //printf("Longitude - %f\n",data.lon);
            //printf("Meters/Second - %f\n",data.speed);
            //printf("Radians - %f\n",data.heading);

      }
}

void parse(char *string , reading *data)
{
      //Precondition: character array string contains
      // a string in the format as deccribed in file header
      //Postcondition: data will be stored in reading struct
      //string will not be manipulated

      char *lattitude;  //in DDMM.MMMM format
      char *longitude;  //in DDMM.MMMM format
      char *heading;  //in degrees from N
      char *knots;
      char cur = 'A';  //current character
      int index=0;
      int west,south;
      int startIndex;    //index starting the variable
      int numComma=0;    //number of Commas found

      //assuming that none of the degrees will be larger than 19
characters
      lattitude = (char*)malloc(20*sizeof(char));
      longitude = (char*)malloc(20*sizeof(char));
      knots     = (char*)malloc(20*sizeof(char));
      heading   = (char*)malloc(20*sizeof(char));

      while(cur!='\0')
      {
            cur=string[index];
            if(cur==',')  //if a comma is found
                  numComma++;

            if(numComma==3)
```

```
		{
			startIndex = index+1;
			while(numComma==3)
			{
				index++;
				cur=string[index];


				if(cur==',')  //if a comma is found+
				{
					lattitude[index-startIndex]='\0';

					numComma++;
				}
				else
				{
					lattitude[index-startIndex]=cur;  //set
the lattitude string
				}
			}
		}
		if(numComma==4)
		{
			//check hemisphere N or south
			if(string[index]=='S')
				south=1;
			else
				south=0;
		}
		if(numComma==5)
		{
			startIndex = index+1;
			while(numComma==5)
			{
				index++;
				cur=string[index];


				if(cur==',')  //if a comma is found+
				{
					longitude[index-startIndex]='\0';

					numComma++;
				}
				else
				{
					longitude[index-startIndex]=cur;  //set
the lattitude string
				}
			}
		}
		if(numComma==6)
		{
			if(string[index]=='W')
				west=1;
			else
				west=0;
```

```
		}
		if(numComma==7)
		{
			startIndex = index+1;
			while(numComma==7)
			{
				index++;
				cur=string[index];


				if(cur==',')  //if a comma is found+
				{
					knots[index-startIndex]='\0';

					numComma++;
				}
				else
				{
					knots[index-startIndex]=cur;  //set the
lattitude string
				}
			}
		}
		if(numComma==8)
		{
			startIndex = index+1;
			while(numComma==8)
			{
				index++;
				cur=string[index];


				if(cur==',')  //if a comma is found+
				{
					heading[index-startIndex]='\0';

					numComma++;
				}
				else
				{
					heading[index-startIndex]=cur;  //set the
lattitude string
				}
			}
		}

			index++;
	}
	data->lat = minToDeg(atof(lattitude));
	if(south==1)
		data->lat*=-1;
	data->lon = minToDeg(atof(longitude));
	if(west==1)
		data->lon*=-1;

	data->speed = knotConversion(atof(knots));
```

```
            data->heading = angleConversion(atof(heading));

      ///////////////free strings b/c of memory
leak/////////////////////////////////////////////////////
}


double minToDeg(double DDMM)
{
      //converts format DDMM.MMMM to DD.DDDD
      double converted=0;
      double decimalDeg;
      decimalDeg=((double)(((int)(DDMM*10000))%1000000))/10000;  //puts
in DDMMMMMM.00 and puls out the MMMMMM
      decimalDeg/=60;  //divides the MMMMMM to DDDD.DD
      DDMM*=10000; //puts into .DDDDD
      converted = (double)((int)DDMM/1000000);   //pulls out the DD
from DDMM.MMMM
      converted+=decimalDeg;   //makes the DD.DDDD

      return converted;
}



double knotConversion(double knots)
{
      //converts parameter knots to meters/second
      return knots*(double)KNOT_TO_MPS;
}

double angleConversion(double degrees)
{
      //returns standard engineering angle in
      //radians as measured from the x-axis
      //**********This is changed to comply with SAE coordiantes
      if(degrees<=90) //quadrant 1
      {
            return (double)DEG_TO_RAD;
      }
      else
      {
            return (double)DEG_TO_RAD;
      }
}

void sim_serialIO()
{
      //function to do serial I/O returns each string
      //most of these constants are defined in <windows.h>
      DWORD read;
      char* buf1;
      int index=0;
      char* temp;
      buf1 = (char*)malloc(100*sizeof(char));
      temp = (char*)malloc(sizeof(char));
```

```
	while(1)
	{
		read = 0;
		if(!ReadFile(h,temp, 1, &read,&ov))
		{
			if(GetLastError() == ERROR_IO_PENDING)   //read in
each character
				if(!GetOverlappedResult(h, &ov, &read, TRUE))
//blocking overlapped event
					printf("GetOverlappedResult failed");
		}
		if(temp[0]=='q')
			break;
		if(read==1)
		{
			//printf("%c",temp[0]);
			//read in 1 character
			//use the ~ as a way to find a new line
			//temp is what stores the newly input data
			//buf1 is what stores each individual string

			if(temp[0]=='~')  //end of string
			{
				buf1[index]='\0';  //make buffer null
terminated
				printf("%s\n",buf1);
				parse(buf1,&data);
				break;
			}
			else   //still in the string
			{

				buf1[index]=temp[0];
				index++;
			}

		}
	}
	if(buf1)
		free(buf1);
	if(temp)
		free(temp);

}
```

Ini.h

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include "structs.h"

//GPS
char* GPSlistenPort;
int GPSbaud;
//for SIM
char* GPStalkPort;

//SICK
char* SICKcomPort;
int SICKbaud;
//for SIM
char* SICKtalkPort;

//Collision Avoidance Controller
char* CAC_IP;

//Software Controller
char* SWC_IP;
double pathPgain;
double pathDgain;
double velocityPgain;
double velocityDgain;
double headingPgain;
double headingDgain;


void readINI()
{
	//reads in the data to the ini
	FILE* input;
	char* temp;
	char* temp2;
	int num = 0;
	int index;
	int startIndex;

	temp = (char*)malloc(75*sizeof(char));
	input=fopen("agv.ini","r");
	if(!input)  //if there is an error opening the file
	{
		MessageBox(NULL,"Error Opening File","Error!",MB_OK);
	}
	else
	{
		while(fgets(temp,74,input))
		{
			if(temp[0]!='#'&&temp[0]!='['&&temp[0]!='
'&&strlen(temp)>1)
			{
```

```
                        //take out the newline at the end of each line
                        temp[strlen(temp)-1]='\0';
                        switch(num)
                        {
                        case 0:
                              //SWC IP
                              index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              SWC_IP = (char*)malloc(75*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
                                    SWC_IP[index-
startIndex]=temp[index];
                                    index++;
                              }
                              SWC_IP[index-startIndex]='\0';
                              num++;
                              break;
                        case 1:
                              //SWC pathPGain

                              index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              temp2 = (char*)malloc(50*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
                                    temp2[index-
startIndex]=temp[index];
                                    index++;
                              }
                              temp2[index-startIndex]='\0';
                              pathPgain = atof(temp2);
                              free(temp2);
                              num++;
                              break;
                        case 2:
                              //SWC pathDGain

                              index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              temp2 = (char*)malloc(50*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
                                    temp2[index-
startIndex]=temp[index];
                                    index++;
                              }
                              temp2[index-startIndex]='\0';
```

```
                        pathDgain = atof(temp2);
                        free(temp2);
                        num++;
                        break;
                case 3:
                        //SWC velocityPGain

                        index=0;
                        while(temp[index]!='$')
                                index++;
                        index++;  //one more
                        temp2 = (char*)malloc(50*sizeof(char));
                        startIndex=index;
                        while(temp[index]!='\0')
                        {
                                temp2[index-
startIndex]=temp[index];
                                index++;
                        }
                        temp2[index-startIndex]='\0';
                        velocityPgain = atof(temp2);
                        free(temp2);
                        num++;
                        break;
                case 4:
                        //SWC velocityDGain

                        index=0;
                        while(temp[index]!='$')
                                index++;
                        index++;  //one more
                        temp2 = (char*)malloc(50*sizeof(char));
                        startIndex=index;
                        while(temp[index]!='\0')
                        {
                                temp2[index-
startIndex]=temp[index];
                                index++;
                        }
                        temp2[index-startIndex]='\0';
                        velocityDgain = atof(temp2);
                        free(temp2);
                        num++;
                        break;
                case 5:
                        //SWC headingPGain

                        index=0;
                        while(temp[index]!='$')
                                index++;
                        index++;  //one more
                        temp2 = (char*)malloc(50*sizeof(char));
                        startIndex=index;
                        while(temp[index]!='\0')
                        {
                                temp2[index-
startIndex]=temp[index];
```

```
                                    index++;
                              }
                              temp2[index-startIndex]='\0';
                              headingPgain = atof(temp2);
                              free(temp2);
                              num++;
                              break;
                        case 6:
                              //SWC headingDGain

                              index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              temp2 = (char*)malloc(50*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
                                    temp2[index-
startIndex]=temp[index];
                                    index++;
                              }
                              temp2[index-startIndex]='\0';
                              headingDgain = atof(temp2);
                              free(temp2);
                              num++;
                              break;
                        case 7:
                              //GPS listenPort
                                    index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              GPSlistenPort =
(char*)malloc(75*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
                                    GPSlistenPort[index-
startIndex]=temp[index];
                                    index++;
                              }
                              GPSlistenPort[index-startIndex]='\0';

                              num++;
                              break;
                        case 8:
                              //GPS baud

                              index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              temp2 = (char*)malloc(50*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
```

```
                                    temp2[index-
startIndex]=temp[index];
                                    index++;
                              }
                              temp2[index-startIndex]='\0';
                              GPSbaud = atoi(temp2);
                              free(temp2);
                              num++;
                              break;
                        case 9:
                              //GPS talkPort (for SIM)
                                          index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              GPStalkPort =
(char*)malloc(75*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
                                    GPStalkPort[index-
startIndex]=temp[index];
                                    index++;
                              }
                              GPStalkPort[index-startIndex]='\0';

                              num++;
                              break;
                        case 10:
                              //SICK listenPort
                                          index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              SICKcomPort =
(char*)malloc(75*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
                              {
                                    SICKcomPort[index-
startIndex]=temp[index];
                                    index++;
                              }
                              SICKcomPort[index-startIndex]='\0';

                              num++;
                              break;
                        case 11:
                              //SICK baud

                              index=0;
                              while(temp[index]!='$')
                                    index++;
                              index++;  //one more
                              temp2 = (char*)malloc(50*sizeof(char));
                              startIndex=index;
                              while(temp[index]!='\0')
```

```
                                    {
                                          temp2[index-
startIndex]=temp[index];
                                          index++;
                                    }
                                    temp2[index-startIndex]='\0';
                                    SICKbaud = atoi(temp2);
                                    free(temp2);
                                    num++;
                                    break;
                              case 12:
                                    //SICK talkPort
                                    index=0;
                                    while(temp[index]!='$')
                                          index++;
                                    index++;  //one more
                                    SICKtalkPort =
(char*)malloc(75*sizeof(char));
                                    startIndex=index;
                                    while(temp[index]!='\0')
                                    {
                                          SICKtalkPort[index-
startIndex]=temp[index];
                                          index++;
                                    }
                                    SICKtalkPort[index-startIndex]='\0';

                                    num++;
                                    break;
                              case 13:
                                    //CAC IP
                                    index=0;
                                    while(temp[index]!='$')
                                          index++;
                                    index++;  //one more
                                    CAC_IP = (char*)malloc(75*sizeof(char));
                                    startIndex=index;
                                    while(temp[index]!='\0')
                                    {
                                          CAC_IP[index-
startIndex]=temp[index];
                                          index++;
                                    }
                                    CAC_IP[index-startIndex]='\0';
                                    num++;
                                    break;
                              default:
                                    break;
                              }

                        }
                  }
            }
      }

GPS_Data getGPSstruct()
{
```

```
	GPS_Data g;
	g.GPS_baud = GPSbaud;
	g.GPS_COM = GPSlistenPort;
	return g;
}

gains getGains()
{
	gains gain;

	gain._Kdh = headingDgain;
	gain._Kdh2 = pathDgain;
	gain._Kdv = velocityDgain;
	gain._Kph = headingPgain;
	gain._Kph2 = pathPgain;
	gain._Kpv = velocityPgain;
	return gain;
}
```

Pipe.h

```
#define ARRIVED_PIPE_NAME "\\\\.\\pipe\\arrived_flag_pipe"
#define VEHICLE_PIPE_NAME "\\\\.\\pipe\\vehicle_pipe"
#define AVOIDING_PIPE_NAME "\\\\.\\pipe\\avoiding_flag_pipe"
#define MANEUVER_PIPE_NAME "\\\\.\\pipe\\maneuver_pipe"

#define ARRIVED_PIPE 0
#define VEHICLE_PIPE 1
#define AVOIDING_PIPE 2
#define MANEUVER_PIPE 3
#define NUM_PIPES 4

#define BUFSIZE 4096

HANDLE pipe_handles[5];
LPTSTR pipe_names[5] = {TEXT(ARRIVED_PIPE_NAME),
TEXT(VEHICLE_PIPE_NAME), TEXT(AVOIDING_PIPE_NAME),
TEXT(MANEUVER_PIPE_NAME)};
```

RDDF.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "structs.h"


void readFromFile(WP **,int *);
WP parseWP(char*);
void printArray(WP *, int );


void printArray(WP *array,int size)
{
	int i;
	for(i=0;i<size;i++)
	{
		if((i%100)==0&&i!=0)
		{
			printf("Press <ENTER> to continue\n");  //added for
testing
			getchar();
		}
		//printf("Waypoint  - %d\n",array[i].number);
		//printf("Lat  - %.2f  ",array[i].lat);
		//printf("Long - %.2f  ",array[i].lon);
		//printf("LBO  - %.1f  ",array[i].LBO);
		//printf("MPH  - %.1f\n\n",array[i].MPH);
	}
}

void readFromFile(WP** array,int *size)
{
	FILE* input;
	int index = 0;
	char* string;
	input= fopen("waypoints.txt","r");
	string = (char*)malloc(60*sizeof(char));  //assuming strings are
< 59 characters
	if(!input)
	{
		printf("Error occured in opening file execution will
terminate\n");
		return;
	}
	while(fscanf(input,"%s",string)==1)
	{
		(*size)++;  //count how many readings there are
	}
	fclose(input);
	(*array) = (WP*)malloc((*size)*sizeof(WP));//malloc the array
	input = fopen("waypoints.txt","r");  //reopen the file
	while(fscanf(input,"%s",string)==1)
	{
		(*array)[index]=parseWP(string);
```

```
			index++;
		}
		fclose(input);
}

WP parseWP(char* string)
{
		//parses data into lat, lon, and waypoint number
		WP data;
		int startIndex;
		char cur='A';
		int index=0;
		char *lattitude;
		char *longitude;
		char *waypoint;
		char* LatBound;
		char* speedLimit;

		waypoint = (char*)malloc(6*sizeof(char));  //will support up to
9999 waypoints
		longitude = (char*)malloc(15*sizeof(char));
		lattitude = (char*)malloc(15*sizeof(char));
		LatBound = (char*)malloc(15*sizeof(char));
		speedLimit = (char*)malloc(15*sizeof(char));

		startIndex=index;
		while(cur!=',')
		{

			cur=string[index];
			if(cur!=',')
				waypoint[index-startIndex]=cur;
			index++;
		}
		waypoint[index-1-startIndex]='\0';    //put an ending character
on the string

		startIndex = index;
		cur=string[index];
		while(cur!=',')
		{
			cur=string[index];
			if(cur!=',')
				lattitude[index-startIndex]=cur;
			index++;
		}
		lattitude[index-1-startIndex]='\0';

		startIndex=index;
		cur=string[index];
		while(cur!=',')
		{
			cur=string[index];
			if(cur!=',')
				longitude[index-startIndex]=cur;
			index++;
		}
```

```
	longitude[index-1-startIndex]='\0';


	startIndex=index;
	cur=string[index];
	while(cur!=',')
	{
		cur=string[index];
		if(cur!=',')
			LatBound[index-startIndex]=cur;
		index++;
	}
	LatBound[index-1-startIndex]='\0';

	startIndex=index;
	cur=string[index];
	while(cur!=',')
	{
		cur=string[index];
		if(cur!=',')
			speedLimit[index-startIndex]=cur;
		index++;
	}
	speedLimit[index-1-startIndex]='\0';



	data.lat = atof(lattitude);
	data.lon = atof(longitude);
	data.number = atoi(waypoint);
	data.LBO = atof(LatBound);
	data.MPH = atof(speedLimit);
	return data;
}
```

Structs.h

```
//header file created to hold the structs used in the program instead
of linking includes
#define NUM_BEAM_COLS 361           /* # of columns in the sonar array
      */
#define NUM_BEAM_ROWS 1             /* # of rows in the sonar array
            */
#define NUM_RANGE_BYTES 100         /* 8 Range bins per byte
            */

//define struct for waypoint data
#ifndef _WP1
#define _WP1 typedef struct WP
_WP1   //for waypoint data
{
      int number;
      double lat;
      double lon;
      double LBO;
      double MPH;
}WP;
#endif

#ifndef _GPSDAT
#define _GPSDAT typedef struct GPS_Data
_GPSDAT
{
      char* GPS_COM;
      int GPS_baud;
}GPS_Data;
#endif;

#ifndef _CONSTRUCT
#define _CONSTRUCT typedef struct gains
_CONSTRUCT
{
      double _Kpv;
      double _Kdv;
      double      _Kph;
      double      _Kdh;
      double      _Kph2;
      double       _Kdh2;
}gains;
#endif

//define struct for GPS data
#ifndef _GPSread
#define _GPSread typedef struct reading
_GPSread   //for GPS data
{
      double lat;  //lattitude
      double lon;  //longitude
      double speed;  //in meters per second
      double heading; //in engineering degrees
}reading;
```

```
#endif

#ifndef _RDV
#define _RDV typedef struct Range_Data_Vector
_RDV
{
      unsigned char
range_bytes[NUM_BEAM_ROWS][NUM_BEAM_COLS][NUM_RANGE_BYTES];
} Range_Data_Vector;
#endif

#ifndef _VSV
#define _VSV typedef struct Vehicle_State_Vector
_VSV
{
      float x, y, depth, heading, pitch, roll, speed, dive_rate;
      float start_x, start_y, goal_x, goal_y, simulated;
} Vehicle_State_Vector;
#endif

#ifndef _MV
#define _MV typedef struct Maneuver_Vector
_MV
{
      float new_heading, heading_rate, new_depth, dive_rate;
} Maneuver_Vector;
#endif
```

Agv.ini

```
#AGV.ini
#This file tells programs which serial ports to listen on for
#specific devices, and at what speed (baud rate) to listen
#as well as specifying server ports for TCP/IP communication.
#This file will also be used by the simulator so it will know
#what ports are in use by other devices, as well as what baud
#rate to talk to those devices at.

#Vehicle controller SoftWare Controller (SWC) (IP or hostname)
#
[SWC]
IP $127.0.0.1
pathPgain $.004
pathDgain $0.0001
velocityPgain $.2
velocityDgain $0.015
headingPgain $3.3
headingDgain $0.04

[GPS]
listenPort $com1
baud $57600
#the send= lines are for use by the simulator so it
#knows which com ports to send data out on.
talkPort $comY

[SICK]
comPort $com1
baud $38400
talkPort $comY


#Collision avoidance controller address (IP or hostname)
[CAC]
IP $127.0.0.1
#127.0.0.1 is the loopback address.  We'll use this when named pipes
#are on the same machine.  The IP address of the remote machine
#will be used when the controller and simulator are on different
machines
```

Waypoints.txt (RDDF file)

```
1,30.634130,-96.482413,90,10,####,####,####
3,30.631968,-96.479497,90,45,####,####,####
4,30.632005,-96.479870,90,20,####,####,####
5,30.632542,-96.479965,90,20,####,####,####
6,30.632622,-96.480702,90,20,####,####,####
7,30.633275,-96.480782,90,20,####,####,####
1,30.633757,-96.481260,90,35,####,####,####
9,30.633490,-96.481352,90,35,####,####,####
10,30.632152,-96.479558,90,35,####,####,####
11,30.633268,-96.480913,90,40,####,####,####
```

VITA

Names:	James Patrick Massey
Address:	344 Honeysuckle Ln Longview, TX 75605
Email Address:	jpmassey@gmail.com
Education:	B.S., Mechanical Engineering, Texas A&M University, 2004 M.S., Mechanical Engineering, Texas A&M University, 2006