THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL AND NUCLEAR ENGINEERING

MOTORIZED TESTBED FOR AUTOMATED RC VEHICLE CONTROL

RAVEEN FERNANDO
FALL 2014

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree in Mechanical Engineering
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Sean N. Brennan
Associate Professor of Mechanical Engineering
Thesis Supervisor and Honors Advisor

H. Joseph Sommer
Professor of Mechanical Engineering
Faculty Reader

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

This research focuses on structural modifications to the Pennsylvania State University Rolling Roadway Simulator (PURRS). Operated within the Intelligent Vehicles and Systems Group, the PURRS is a large treadmill for testing automated control algorithms on small scale vehicles before they are implemented on full scale vehicles. Testbeds with scaled vehicles have a number of benefits compared to testing on full scale vehicles, which is a strong motivation for the completion of this project. For example, algorithm testing on a scale vehicle is more cost effective, convenient, and significantly reduces safety concerns in the event of failure.

One of the major modifications undertaken in this thesis ws the installation of a new air bearing under the running belt. Like an air hockey table, the air bearing uses air flow to lift the treadmill belt, reducing the friction between the deck and the belt and improving motor efficiency. Speed tests were performed with and without the air bearing, to provide conclusive results on the benefits of adding the air bearing to the system.

Additionally, a new small scale vehicle for the PURRS was also used to test position-based feedback control algorithms. Specifically, the vehicle was tested with the use of proportional and proportional-derivative control algorithms. High resolution encoders were used to determine vehicle position relative to a centerline and lateral position errors were used as inputs for controlling the vehicle steering. Perturbation analysis of controller performance was done by manually offsetting vehicle and then enabling control algorithm to correct the position, and both the P- and PD-controlled vehicles were able to maintain centerline position while driving on the roadway.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to give thanks to everyone who has helped me find my way through my years as an undergraduate. A lot has changed in how I view myself and my future. I would like to give special thanks to:

-To my parents, for all their support and guidance throughout the years. My parents were great role models to have growing up. I am thankful for all the late night lessons in mathematics with my father, from middle school and even a little bit of college. My place in engineering would not be the same otherwise.

-To Dr. Brennan, for being a great advisor over the years, and assisting me the research that went into this thesis.

Chapter 1

Introduction

1.1 Thesis Overview

The purpose of this research is to complete modifications of the existing rolling roadway testbed that is operated within the Intelligent Vehicles and Systems Group of the Pennsylvania State University. The testbed is a rolling treadmill platform that allows testing control algorithms for automated vehicles but on a reduced-scale size. Scaled vehicle testing for control algorithms has advantages over full scale vehicles. For example, the use of a scaled vehicle such as an RC car is cheaper to implement control algorithms and run tests than it would be a full size vehicle tested on a test track. The enclosed, small, testing area of a rolling-roadway decreases the need to schedule and close off large areas for testing, such as a test track. An indoor testbed is unaffected by weather and always available for researchers to use, thus making it convenient for rapid testing and modification of code. Aside from convenience, a test bed is safer than the alternative of a full sized vehicle, especially when there is a possibility of an unpolished control algorithm causing a vehicle to steer erratically or off-course. The risk of damaging the test vehicle or even harming personal is also considerably higher with full sized vehicles.

This research continues the work of previous students to complete such an indoor treadmill testbed to be in working condition for current and future automated testing of an RC car. A discussion of scale vehicle research and the history of the current test bed are in the following two sections. The goals of what is to be done is explained in Section 1.4

1.2 Literature Review

The history of research within automated vehicles and highways shows a historic concern with the cost and limitations of running automation tests on full scale vehicles in full scale environments. Full scale testing of experimental vehicles is costly and presents a potentially dangerous environment for human operators in the vicinity [6][7][8]. Computer simulations negate the safety concern, but simulations can be overly complicated and may not display all outcomes from unexpected real life scenarios [8].

Between these two extremes, experimentation on a small scale vehicle provides a bridge between pure simulation and implementation on full scale vehicles. Scale vehicles have been used in research in small scale vehicles move on a fixed roadway [4][12][13][14]. For example, in 1996 researchers at the Virginia Polytechnic Institute and State University developed a scaled vehicle testing environment called the Flexible Low-cost Automated Scale Highway (FLASH) laboratory [4]. The laboratory used multiple scale vehicles that run on a modular highway that could be adapted to present different scenarios or roadway conditions. Small scale vehicle testing allowed for hardware-in-the-loop experiments. Impervious to outside weather conditions, these indoor testbeds offered convenience and a removed the danger of human operators with full scale vehicles [4].

Alternatively, instead of fixing the roadway in place, the scale vehicle can be fixed in location while the roadway is moved underneath it using a treadmill [10][11]. This is analogous to wind tunnels in testing the aerodynamic qualities of an airplane. An example of this, in 1996, the University of Illinois created the Illinois Roadway Simulator (IRS). The IRS used a moving roadway platform in the form of a 4 x 8 ft treadmill to simulate driving over distance, allowing for tests involving steering performance [2][3]. The IRS used optical encoders on a jointed boom arm to measure position and direction of the test vehicle. The IRS treadmill testbed was also

designed with the considerations that it will reduce cost of trials, increase the safety by being in a sealed environment, and convenience. The PURRS system took the testbed design one step further with the implementation of actuators on the rolling roadway itself, allowing for the entire system to be angled [8]. This allows for further control of the surface to simulate more environments, such as a vehicle corning an embankment.

A common note about scale vehicle testing, especially using a rolling roadway testbed, is that it allows for quick iterations and advancements in new control algorithms; which in turn makes a testbed highly desirable testing equipment for vehicular research.

1.3 History of the PURRS

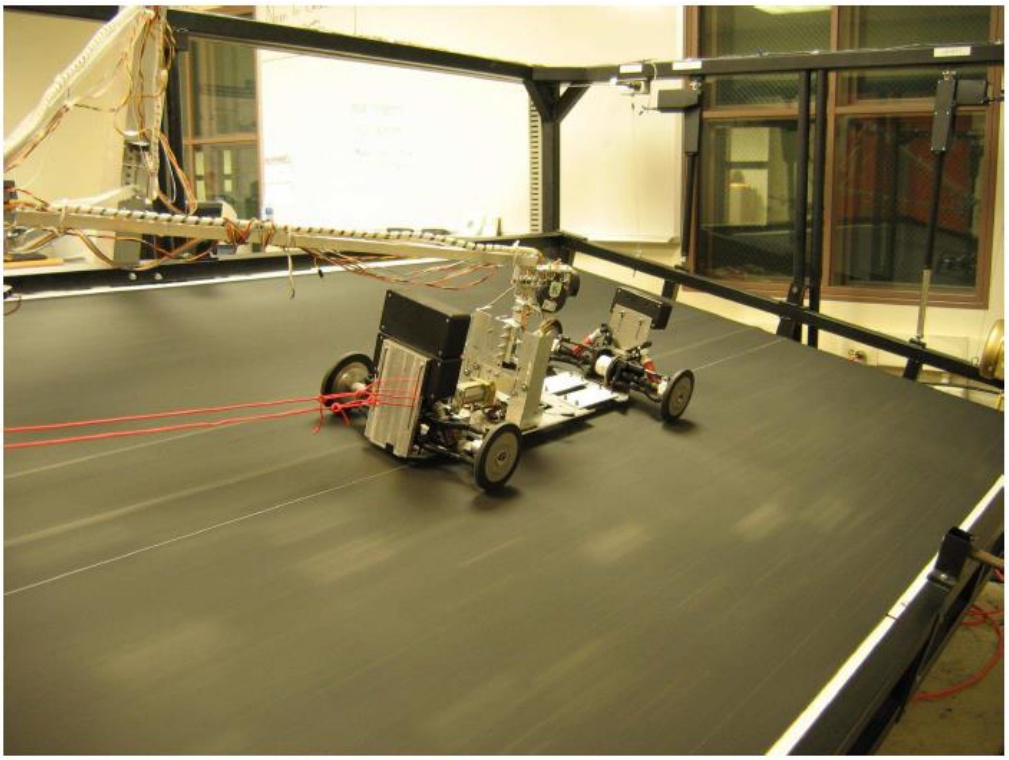

Figure 1 Original PURRS

The Intelligent Vehicles and Systems Group uses a large treadmill to act as a roadway surface the moves under the vehicle, hence the name Rolling Roadway simulator. The Pennsylvania State University Rolling Roadway Simulator (PURRS) features a 9 ft by 6 ft deck allowing scale vehicles to have moderate mobility both longitudinally and laterally. The testbed is run by a 2 HP AC motor controlled by a Variable Frequency Drive. The original testbed (Figure 1) featured 4 actuators attached to the PURRS frame to allow the entire platform to be angled[However future modifications done by masters student Anthony Mangus, increased the overall weight of the PURRS, necessitating that the actuators be removed and the PURRS rest flat on the ground. Work by Mangus also introduced an air bearing system to be installed directly underneath the belt, to decrease friction between the belt and the upper deck surface. The installation of the new deck was incomplete at the end of Mangus' thesis, and thus was the starting point of the present work described in this thesis.

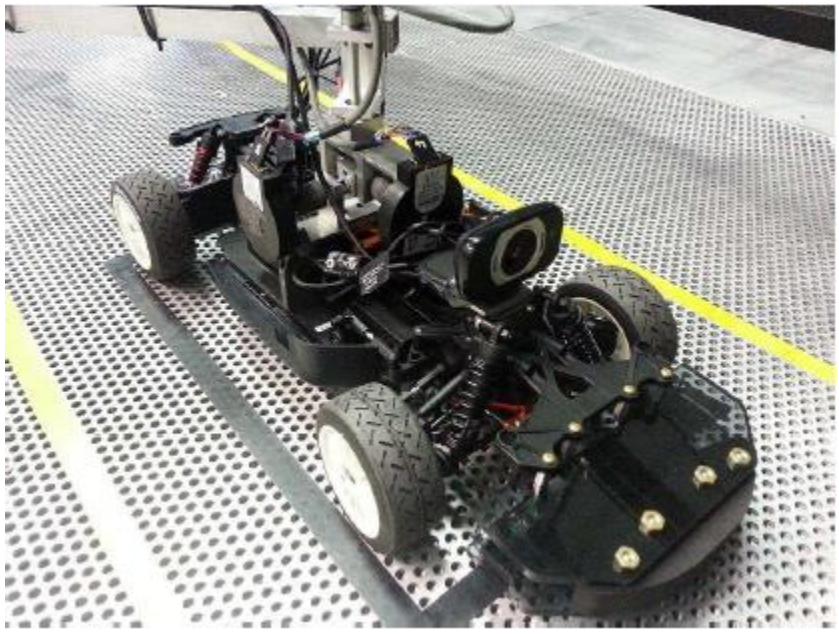

Figure 2 HPI WR8 Flux RC car

Additional changes saw the old car (Figure 1) replaced with a new RC car (Figure 2), and as well the vehicle was fitted with an Arduino to be controlled via manual remote control or serial communication.

1.4 Goals of this thesis

The primary goal of this thesis is to complete the PURRS testbed with a functioning air bearing. The second goal is to create a control scheme for the RC vehicle to run autonomously on the working PURRS.

The design process used to achieve these goals is ultimately an experimental approach. The system was decomposed into subsystems, each sequentially calibrated and tested, and then interconnected so that that each subsystem can communicate to each other. For this endeavor, there are three main subsystems to control.

-The position determination of vehicle using encoders an Arduino Due

-Control Algorithm and servo control of vehicle using an Arduino Uno

-Communication between the separate Arduinos

Details on this process are described in the remainder of the thesis. First, the discussion on the modifications and completion of the PURRS occurs in the following chapter. A presentation of the subsystem related to the control architecture is given in Chapter 3. Chapter 4 details the performance of the algorithm. The concluding chapter discusses the primary outcomes of the thesis, and presents some areas in which future research can proceed with the completed PURRS testbed.

Chapter 2

The PURRS

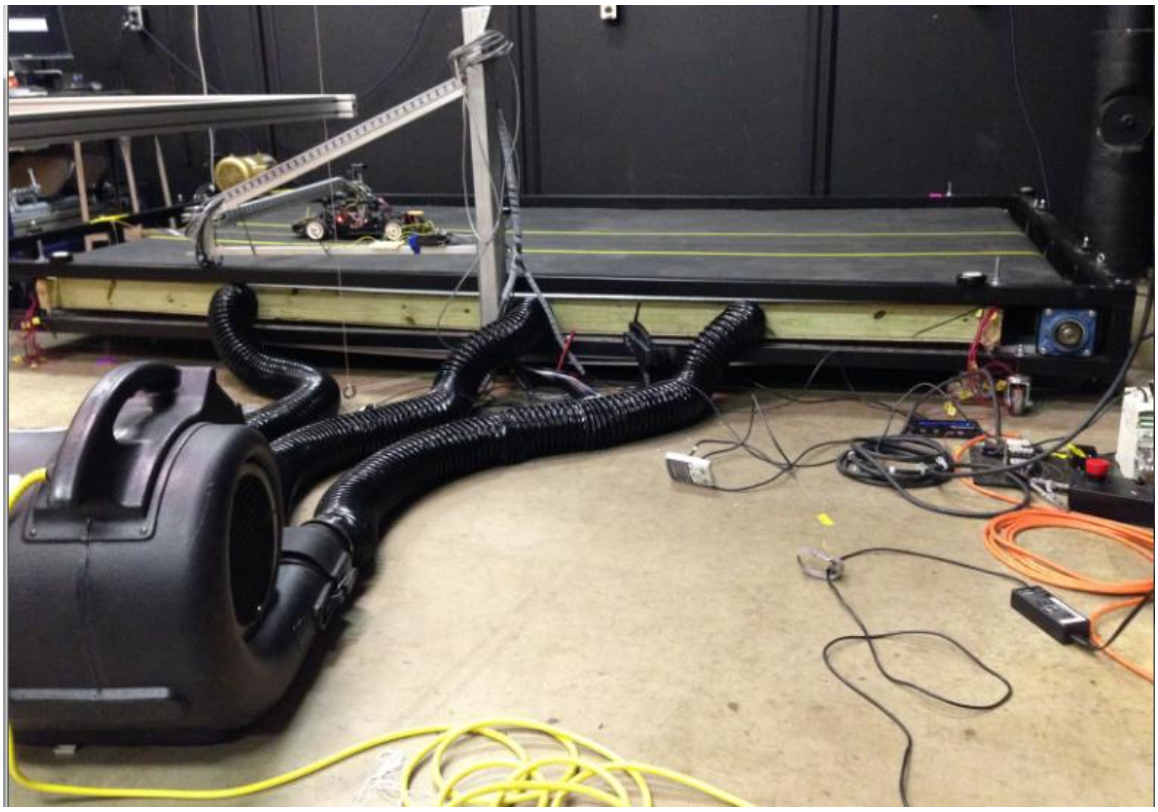

Figure 3 Completed PURRS Set-Up

One of the main facets of this project is to have a completed scale testbed to be able to test control algorithms on scale vehicles. The Pennsylvania State University Rolling Roadway (PURRS) is the testbed for the Intelligent Vehicles and Systems Group. The modifications as mentioned in the history section were not complete as of this project, and so the majority of the thesis work was to implement hardware and structural changes to the physical design. These efforts are described task-by-task in the following sections, and are presented in the approximate time sequence used for the construction of the new PURRS.

2.1 Description of Overall Structural Changes to PURRS

The original design of the rolling roadway had the upper surface of the treadmill belt riding across a plywood surface. It was determined that, with such a large deck area of 36 cubic feet, there was a lot of friction between the physical tread and the supporting deck underneath. Excess friction started to cause the deterioration both the belt and the deck during prolonged use. As part of Mangus' MS thesis work mentioned earlier, it was proposed that an air bearing be installed to reduce the friction and thereby prolong the longevity of the PURRS components.

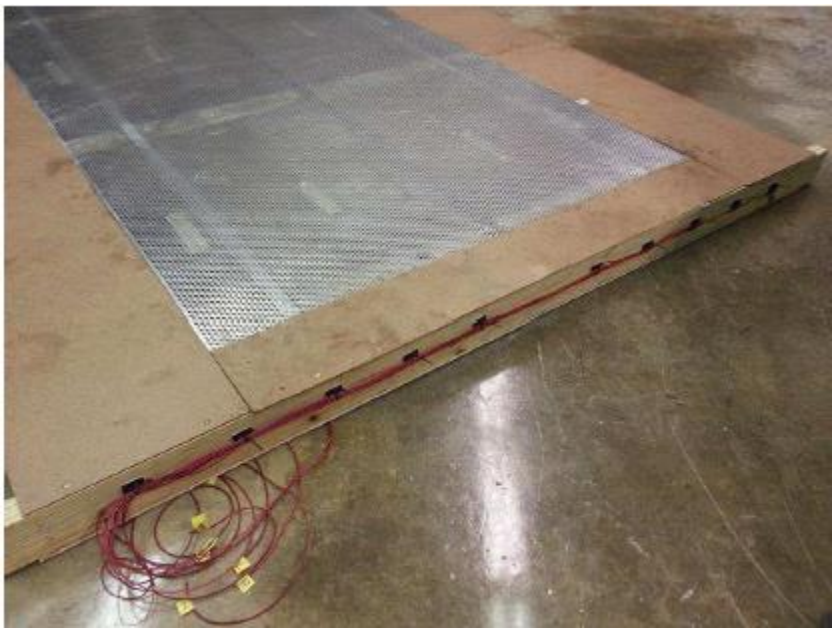

Figure 4 New Deck with Holes in Upper Deck

The design ultimately constructed and implemented in this thesis replaces the plywood deck with a box with tiny holes covering the top surface (Figure 4). A blower pumps air into the box and the pressure and force of the escaping air along the top surface is enough to create a small layer of air between the belt and the deck, such that it reduces the friction between the belt and the deck surface. To implement this concept, 5 inch tall wooden box was created to replace the existing particleboard deck. This box had to be sufficient in strength to both hold the vehicle and distribute the air underneath the belt surface, but also had to be thin enough to fit between the

2 layers of moving belt. The top surface was created with two 4ft x 8ft aluminum plates that are 1/8 inch thick and perforated with ¼ inch holes. Aluminum was chosen because it was a cost-effective solution for weight, strength, and also because it was non-magnetic. The use of non-ferromagnetic materials was important for future considerations, because the treadmill was intended for testing positioning algorithms that utilize magnetic fields. The 2 aluminum plates sit on top of eachother and are able to slide relative to each other such that the alignment of the holes alter size and therefore the pressure of the air flow.

Additional modification was required as the belt has a ½ in wide by ½ in deep protrusion continuously along the inside of the belt to ensure proper alinement of the belt on the 2 rollers on either end. This protrusion rubbed against the deck and risked damaging the belt. To avoid this rubbing effect, a 1 in x 1/2 in channel had to built into the deck to allow obstruction free passage of the protrusion. This channel is shown in figure 1.

When Anthony Mangus left the project, the air box was only partially built and was not functioning. When the work for this thesis began, the following had to be done:

-Construct channel for belt protrusion
-Install new deck within PURRS frame
-Install Belt
-Install Motor

2.2 Construction of the guide channel

The construction of a channel within the deck was necessary to allow the belt's guide mechanism to operate correctly. This required the installation of an aluminum U-channel. Aluminum was chosen again for the cost effectiveness and being non-magnetic. The channel needed to fit flush and fit precisely with the sheet-metal surface, such that there was not a large gap for air to escape from. The sheet metal was taken to a metal shop and was sheared into four

segments, each measuring 23 ½ in x 8 ft. The rest of wooden frame was slotted using a router and a 1 in bit to carve a slot for the channel.

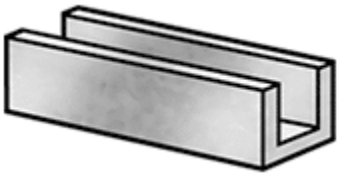

Figure 5 U-Channel for PURRS Deck

Once the aluminum plates and U-channel were installed, the entire deck was fitted into the PURRS steel frame, as seen in Figure 5.

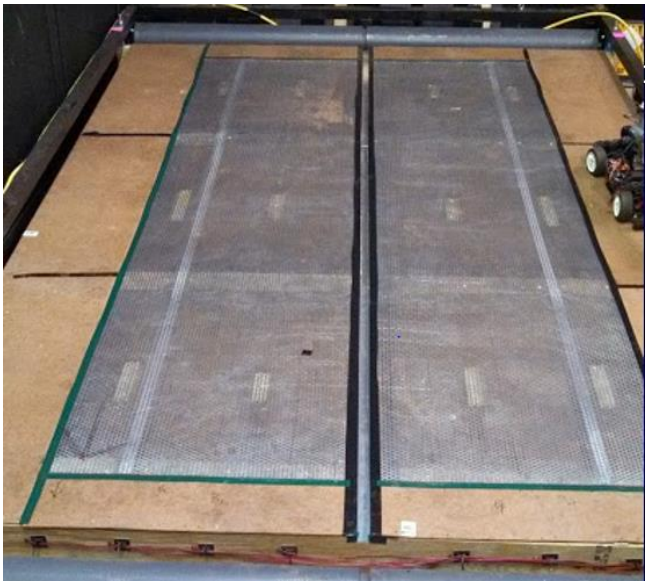

Figure 6 PURRS Deck with Perforated Sheets and Channel

2.3 Installation of the belt and drive motor

Once the frame was completed, the treadmill belt was fitted over the rollers. Afterwards, the drive motor was mounted to the PURRS. With the motor and belt attached, both the motor belt, and the PURRS belt had to be tensioned properly. Proper tensioning of the belt ensures that

belt slack is addressed and that both rollers of the belt are parallel. Uneven rollers will cause the belt to slide off the rollers with continued use. One set of rollers is adjustable on a set of tracks and needs a wrench to adjust both ends until the belt is adequately tensioned to prevent roll-off as mentioned earlier. The motor (Figure 7) position is also adjustable, using a bolt and nut, to ensure that that the motor drive belt is also kept a proper tension to rotate the treadmill rollers.

Figure 7 2 HP AC Motor

Figure 8 One of Three Airways for Blower Attachment

Attached to the deck are 3 airway channels (Figure 8) that are attached to an industrial blower with hose attachments (Figure 9) to it supply sufficient airflow to the entire box. The fan

does not provide a great deal of force with the air flow pressure, but in an enclosed area such as the box within the PURRS, it provides enough air flow to lift the belt off the deck slightly.

Figure 9 Industrial Blower for Air Flow

The treadmill runs with a 2 HP AC motor (Figure 7) that is controlled via a Variable Frequency Drive (Figure 10) that allows for variable speed and direction of the treadmill. The Variable Frequency Drive is attached to the power supply along with an emergency stop, which stops the belt within ¼ of a belt revolution.

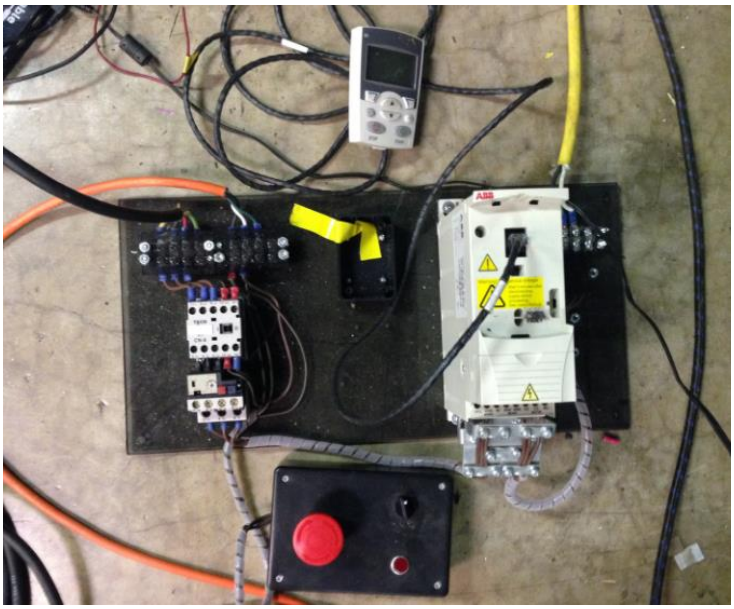

Figure 10 Variable Frequency Drive, Power Supply, & E-Stop

The introduction of the air bearing has reduced the effects of friction, allowing the treadmill to move the belt with less torque. The motor does not have feedback of how fast the belt is actually moving, but instead is an open loop control with a speed set via the VFD. Table 1 displays the average time for 1 complete revolution taken over multiple trials. 12 Hz is the speed at which the control algorithms were tested, but lower frequencies were looked at as well. From time alone, it is evident there is a slight decrease in time when the air bearing is engaged.

Table 1 Average time for 1 revolution

	5 Hz	10 Hz	12 Hz
W/O Air Avg time (s)	12.22	5.63	4.65
W/ Air Avg Time (s)	11.47	5.46	4.54

The length of the belt is 254.25 in. so the average speed of each trial is listed in**Error! Reference source not found.**.

Table 2 Average Speed of Belt

	5 Hz	10 Hz	12 Hz
W/O Air Speed (mph)	1.18	2.57	3.11
W/ Air Speed (mph)	1.26	2.65	3.18

The speed is higher in each case with the air bearing engaged, and Table 3**Error! Reference source not found.** displays the percentage by how much the speed increase. As shown, the air bearing has a greater effect on the low speeds and as speed is increased, the air bearing while still offering an increase in speed, is less noticeable.

Table 3 Percentage Speed Increase

	5 Hz	10 Hz	12 Hz
Percent Increase (%)	6.51%	3.18%	2.50%

Chapter 3

Sensors and Equipment

For the PURRS system to be used for vehicle guidance, a method is needed to accurately measure the lateral position of the scale vehicle, in this case an RC car, and use that position to control the steering of the car. This thesis used high resolution encoders on an arm to determine position, and used Arduino microcontrollers to calculate and change the steering angle accordingly. Details of this process are given in this chapter.

3.1 Encoders

Available in the current set up are 5 high resolution US Digital S2-2048-IB encoders. Two are attached to the position arms and three are attached to the vehicle to measure roll, pitch, and yaw (Figure 11). For the purposes of this research, only the 2 encoders attached to the positioning arms are necessary.

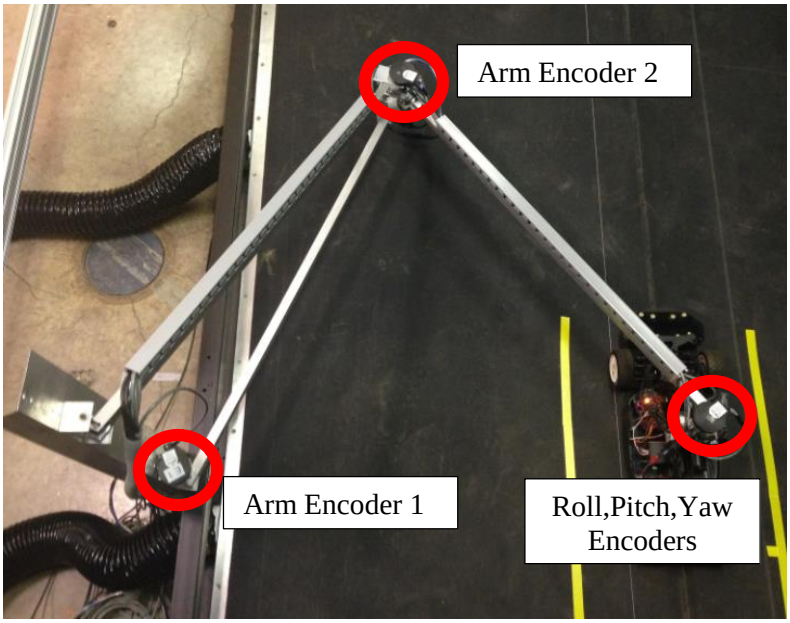

Figure 11 Encoder & Arm Set-Up

The relative-position encoders were used in this thesis to measure angular position with respect to a reference position. These measurements, and taking the length of each arm, allow one to calculate the position of the center of the vehicle as well as the vehicle's orientation. The encoders (Figure 12) have a resolution of 8000 counts per revolution, so it is accurate to 0.045°. High resolution is important especially when dealing with long radii of arms. With such resolution, and 36 inch arms, the measured position has a resolution of roughly 5/100th of an inch.

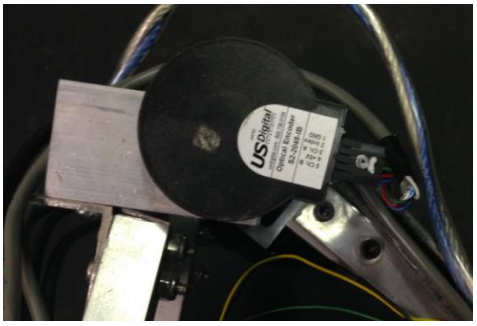

Figure 12 S2-2048-IB Optical Encoder

The encoders are incremental as opposed to absolute, so in order to read the angle correctly, they have to be initialized at a known starting point. To achieve this, the arms have to be set at a 90° angle, with the first arm parallel to the PURRS side, and the 2nd arm facing the belt. (**Error! Reference source not found.**) Using this set up and the manner in which the ncoder counts are set up, the frame of reference is set such that 0° is forward, and positive angle clockwise to that.

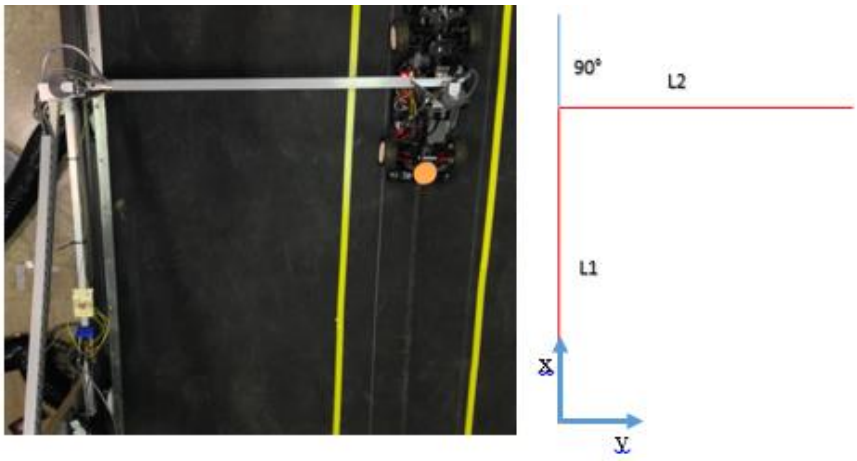

Figure 13 Initialization Set-Up

Taking the first segment of the arm attached to the PURRS as arm 1, and the segment attached to the car as arm 2, the lengths and angles of each can be used to determine the x and y position of the vehicle via the following equations:

$$X_p = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2 + 90°)$$

$$Y_p = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2 + 90°)$$

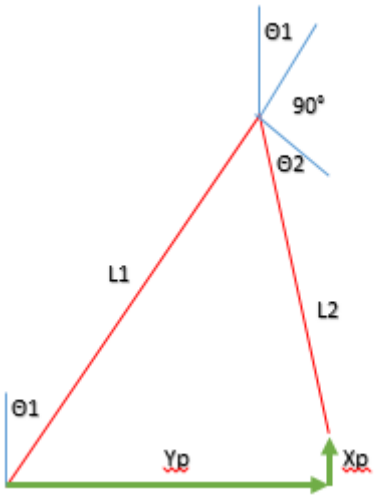

Figure 14 Geometry of Encoder Arms for Position Equations

The difference of the vehicle lateral position and the target lateral position of the car gives the lateral position error, which will go into the proportional control discussed later.

3.2 Arduino Due

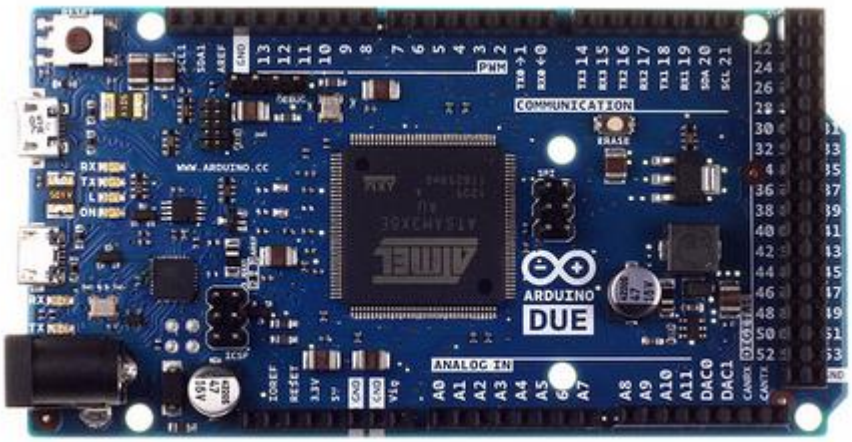

Figure 15 Arduino Due (Pin attachments in Appendix A)

The processing for the control system is implemented using 2 Arduino microcontrollers. One is an Arduino Due which processes the encoders and sends a displacement value to the

second Arduino, the Uno model, on the RC car which uses the displacement to calculate a steering command as appropriate.

Arduinos are powerful but low-cost microcomputers. They are suitable for this project because of the ease of programming and community support available for the devices. Each Arduino is equipped with a "shield", a circuit that connects to the top of the microcontroller that enables circuitry to be hardwired for a more permanent solution. The use of a shield board allows the Arduino to be removed and repurposed in other applications, or easily replaced if there is a failure.

The Arduino Uno is the most popular and commonly used board for projects; however, it is not adequate for the encoder measurements of this particular project. Arduino Unos only have 2 interrupt pins, which is insufficient for the accurate use of the minimum of 3 encoders on this system which require a minimum of 6 interrupt pins. Rotational optical encoders work by shining a light through a slotted Mylar disk with a light sensor on the other side. When light passes, the sensor registers 5 volts, and when light is obstructed, it registers 0 volts. An encoder is able to measure rotation based on the number of changes the sensor undergoes. A quadrature encoder has 4x the resolution by simply having a second light sensor offset from the first sensor. An example is seen below in Figure 16.

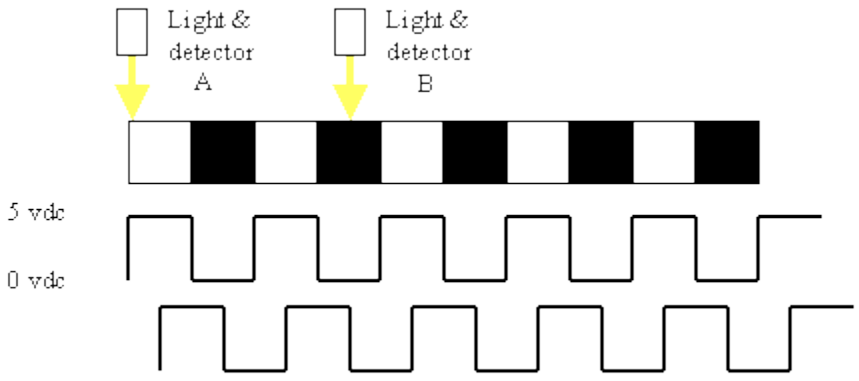

Figure 16 Encoder example. Channel A on top, Channel B on bottom

A normal rotation creates 2 square waves as seen in the example, but depending on the direction, signal A will lead signal B, or signal B will lead signal A. The encoder looks for changes in states for signal A or B, and this change can occur at any time, necessitating interrupts in the code to detect these changes. The Arduino Due has a faster processor and is able to use any of its pins as interrupt pins, making it ideally suited to read the encoders here. On the shield, 7 ports have been added to allow up to 7 encoders to be read simultaneously if needed. However only two are needed for the two arm encoders. (**Error! Reference source not found.**).

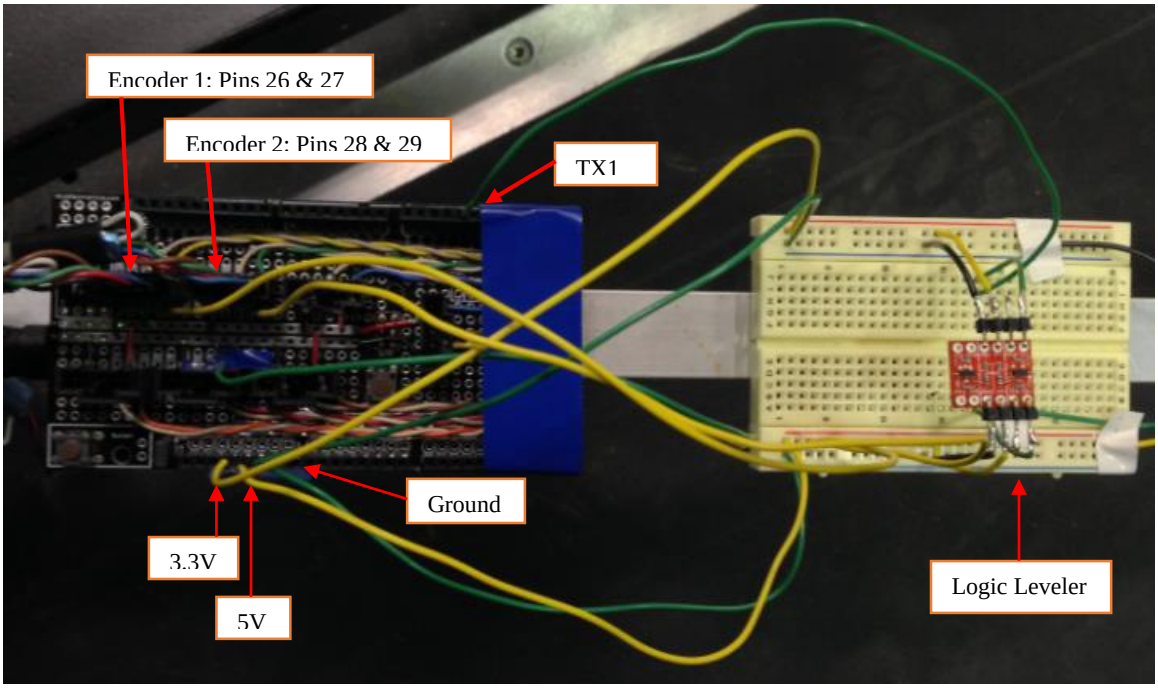

Figure 17 Arduino Due with Breadboard for 5V Distribution and Serial Communication w/ Logic Leveler

The only drawback to using the Due is that it is natively a 3.3V board, while the encoders require a 5V power supply to perform correctly. Previous research by Mangus used the 3.3V power supply, and while this usually was sufficient to power the encoders, the performance was inconsistent when the encoders were attached to the Due when the shield was soldered together. The culprit was the power supply was not providing enough voltage to the encoders. This was rectified by diverting a 5V supply from the one 5V pin available to the Due to a breadboard

where the 5V supply could then be routed back to each encoder cable. This is seen with the yellow wires connecting to the lower left of the breadboard from the two encoder ports in Figure 17 Arduino Due with Breadboard for 5V Distribution and Serial Communication w/ Logic Leveler

The Due monitors the interrupt pins for changes associated with each encoders 2 signal channels (A and B) and, when changes are detected, it converts the counts to radians. Then using the equation mentioned above, it calculates the position, and subtracts the steady state value to determine the position error value.

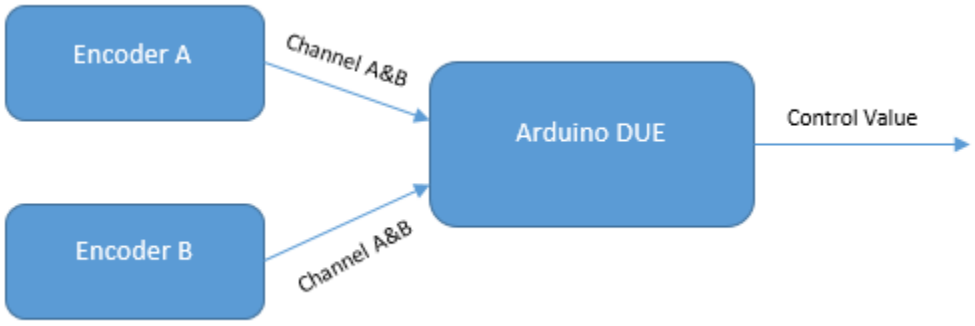

Figure 18 Signal Flow: Arduino DUE

The Due uses serial connection to transmit the position error to the Arduino Uno on the car to control the steering servos. However, the 3.3V of the Due again has a slight drawback in that it cannot talk directly with the Arduino Uno which is a 5V board. In order to boost the voltage of the signal without adding noise and distorting the message, a logic leveler from Sparkfun is used. The chip was connected to both power supplies and as seen in the yellow arrows on Figure 19, it allows signals to be sent from (TXI) the low voltage source and sent to (TXO), the high voltage receiver.

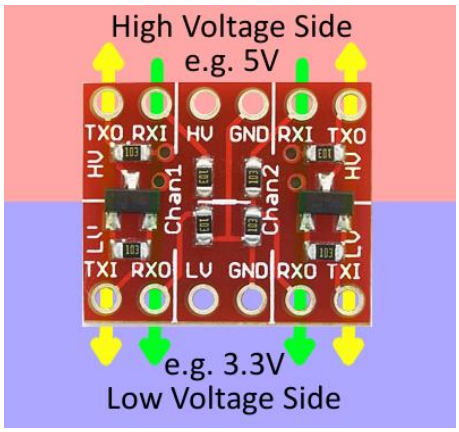

Figure 19 Logic Leveler from Sparkfun

The logic leveler was attached to the breadboard for easy manipulation (Figure 20). The green wires in the figure represent the serial communication lines. Only one connection line is necessary as it is one-way communication with the Due broadcasting and the Uno receiving. Testing of this system showed that the Arduino Uno can now receive the signal over serial communication properly, using the logic leveler.

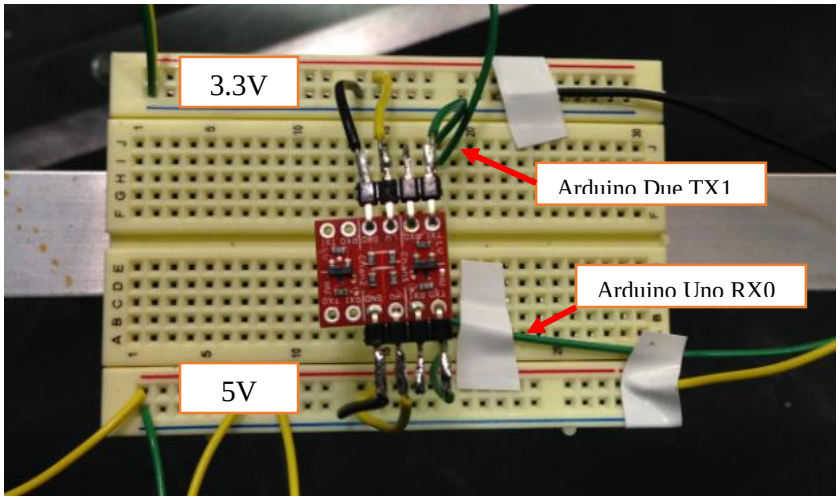

Figure 20 Logic Leveler. (Flipped from figure 17) 5V on bottom. 3.3V on top.

3.3 Arduino Uno

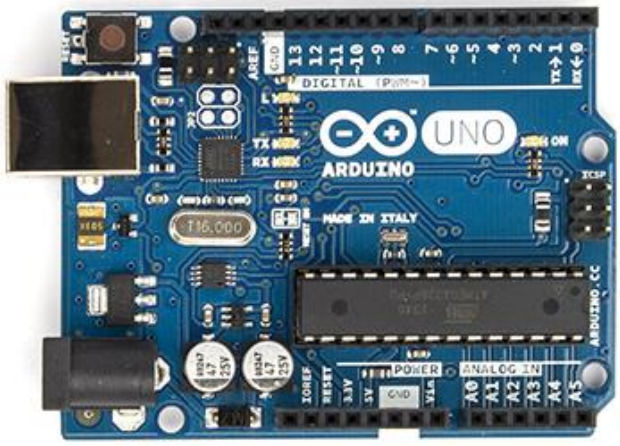

Figure 21 Arduino Uno (Pin Attachments in Appendix B)

The Arduino Uno is connected to both the steering servo on the RC car, and the electric speed control which in turn controls the drive motor of the RC car. However, where it gets the instructions on how to control the servos depends on the operator. There are two modes of operations that the Uno can be in: the first mode is manual mode, in which the Arduino receives steering and speed control signals from the RC transmitter, as they RC connections are connected to pins on the Arduino, allowing the RC car to be completely controlled from the RC controller. The control architecture of the Arduino Uno can be seen in Figure 22.

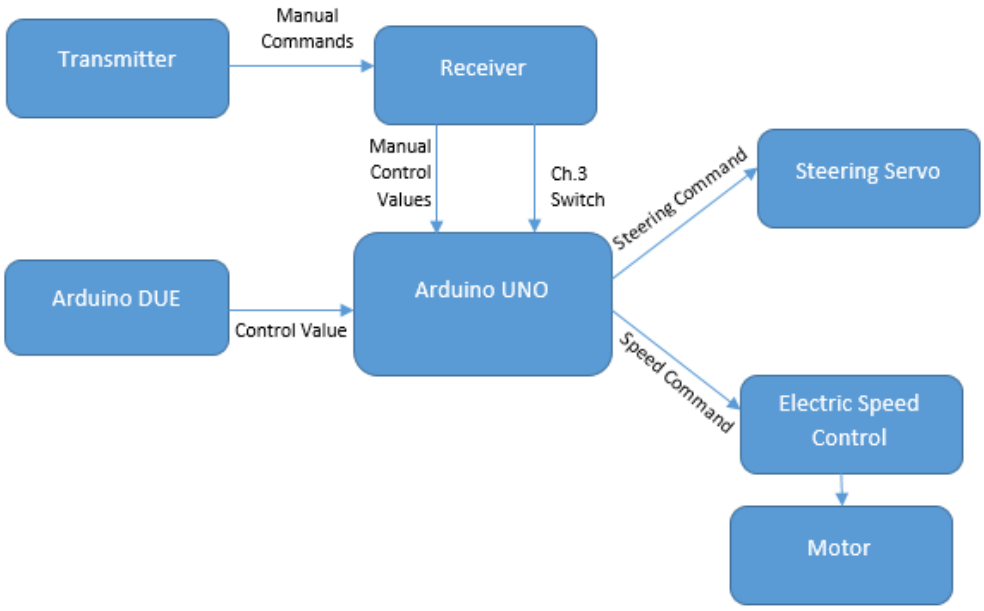

Figure 22 Signal Flow: Arduino UNO

The third RC transmitter connection is for the channel 3 switch, to allow the vehicle to be controlled via a manual transmitter (Figure 24). The Arduino continually checks the state of the signal from the channel 3 switch. If the switch is pressed, the top LED (Figure 24) is off, and the Arduino takes the commands for the servos from the RC controller. The controller signals are between 1100 and 1900, with 1500 being the neutral signal.

Figure 23 RC Controller. Ch. 3 Switch

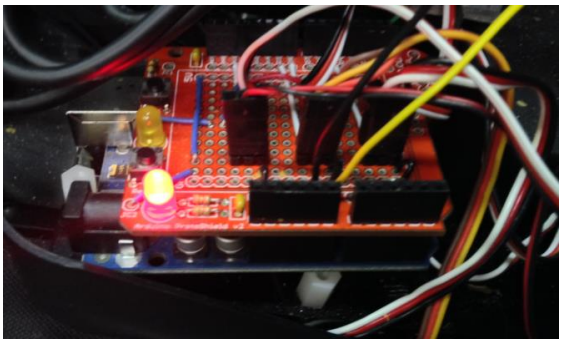

Figure 24 Arduino Uno with Shield and RC Transmitter Connections. Manual RC Mode

If the channel 3 switch is pushed out, the Arduino recognizes the change in the signal, and turns on the LED (Figure 25) to indicate the Arduino is now receiving information from the serial communication instead of the RC transmitter pins. Over the serial communication is the displacement value which, in this thesis, is an integer error value indicating how many $1/00^{th}$ of an inch the car is away from its steady state position. The Arduino Uno uses a simple gain to multiply with the error value and adds that to the neutral value of 1500 to make a control value for the servos to use. In this mode, only the steering is controlled via serial communication, and speed is still controlled from the RC controller.

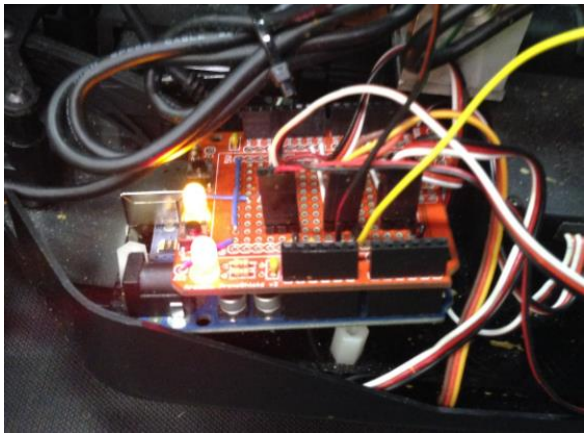

Figure 25 Arduino Uno with Shield and RC Transmitter Connections. Serial Control Mode

Chapter 4

Controller Implementation and Performance

Basic control algorithms can be applied to the system using the 2 Arduinos, and this chapter explains several implementations of such algorithms. Given that the scale RC vehicle is to be run at relatively high speeds, proportional gain was used as a starting point for feedback control. Maintaining speed with the belt speed was not in the scope of this project, and to isolate steering dynamics from longitudinal dynamics, a simple tow line was incorporated to tow the vehicle at a safe distance from the front of the PURRS. It is offset from the center line as is seen in Figure 26 to ensure any that influences from the rope will not help keep the car in desired position even if the control algorithm is lacking, thereby ensuring that centering of the vehicle is being achieved through steering control rather than via the tow line.

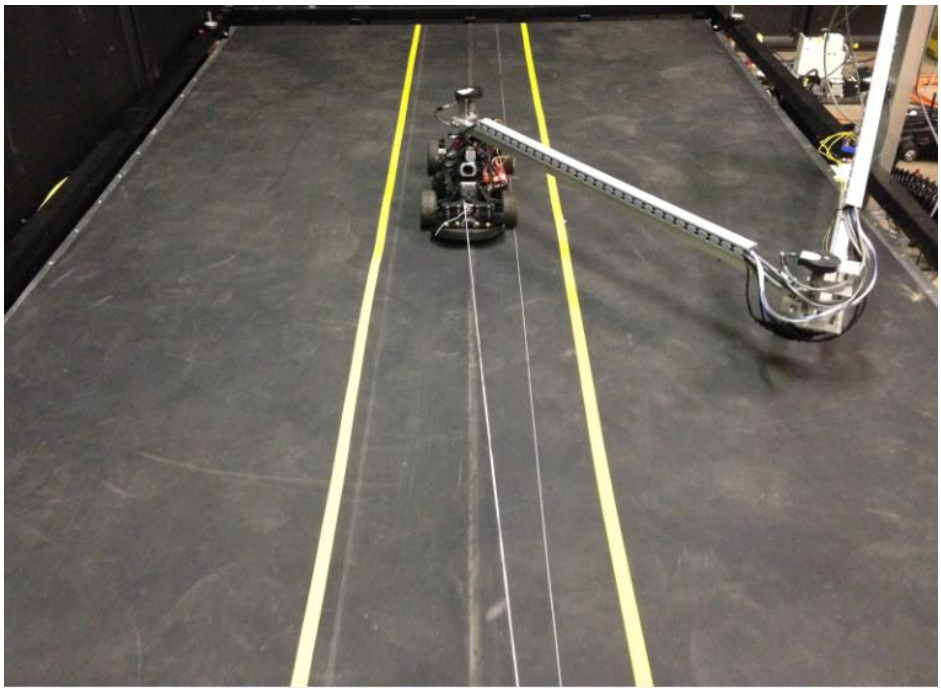

Figure 26 Steady State Position of Car

4.1 Proportional Control

The error values given from the Arduino Due mark how many 1/100ths of an inch the vehicle is from the steady state position. The steering input values have a range between 0-1500 in either direction, so a proportional gain of 50 was chosen to multiply with the error value. Using only the proportional gain, a plot of the lateral distance given a disturbance is seen in Figure 27.

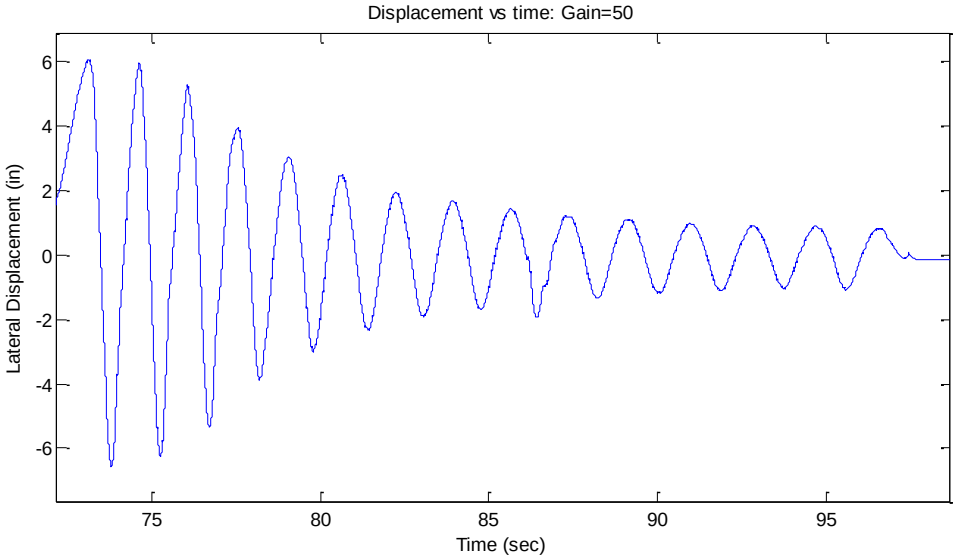

Figure 27 Proportional Gain Plot

Evidently, with only proportional gain, there is a steady oscillations, 1 inch to either side of the steady position. In an attempt to decrease the oscillations, a derivative element was added.

4.2 Derivative Gain

Derivative gain was added in an attempt to minimize the oscillations seen in purely proportional gain with the RC car. While the resulting performance does effectively remove the steady state oscillations, there is significant lateral deviations in the behavior of the vehicle

(Figure 28). The results of the derivative gain are not definitive, and additional retuning would likely achieve smoother results. However, they do show that derivative control is having an effect on performance and that the system

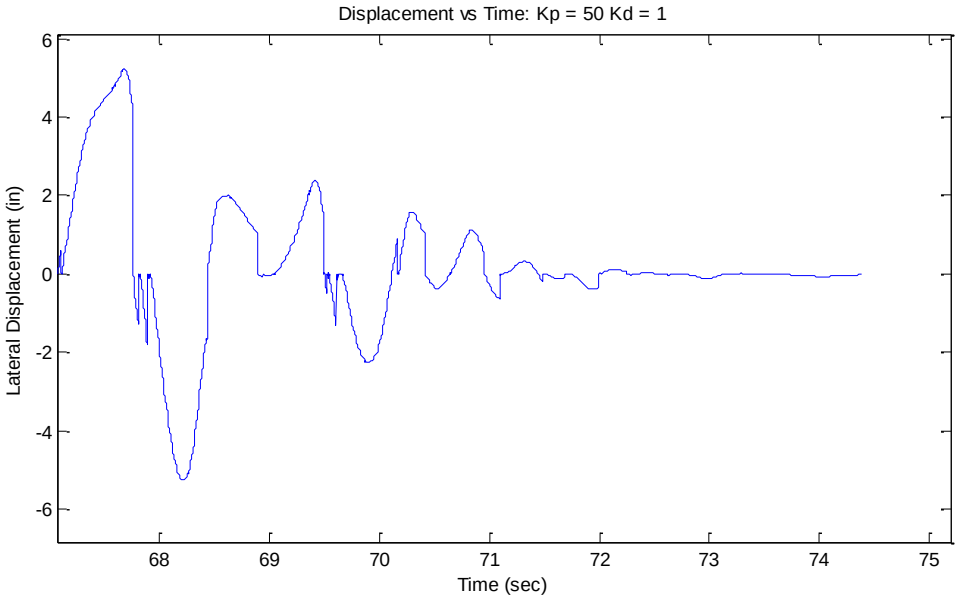

Figure 28 Proportional and Derivative Gain

4.3 Performance and Errors

The proportional gain works well on the vehicle for light disturbances away from the center, roughly 10 inches right or left. However if the vehicle is perturbed greater than that, it appears to malfunction and a new "center line" or steady state is perceived by the vehicle. As seen below. The wheels in figure 30 show that the vehicle now drives straight roughly 4 inches to the right of where it should be, shown in figure 29, after one of these perturbations.

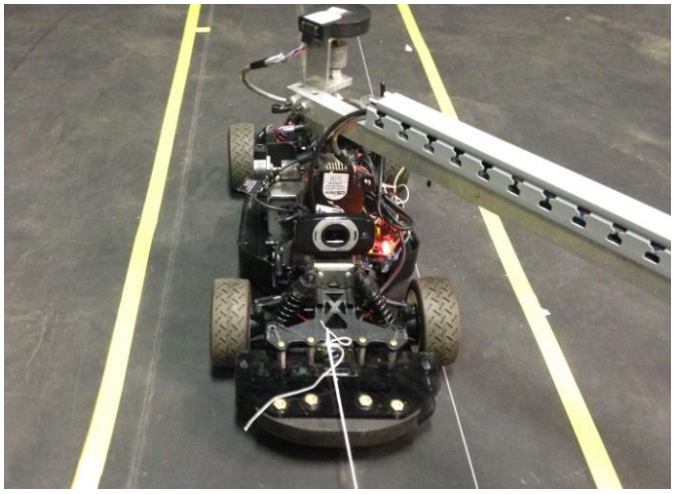

Figure 29 Desired Steady State Position

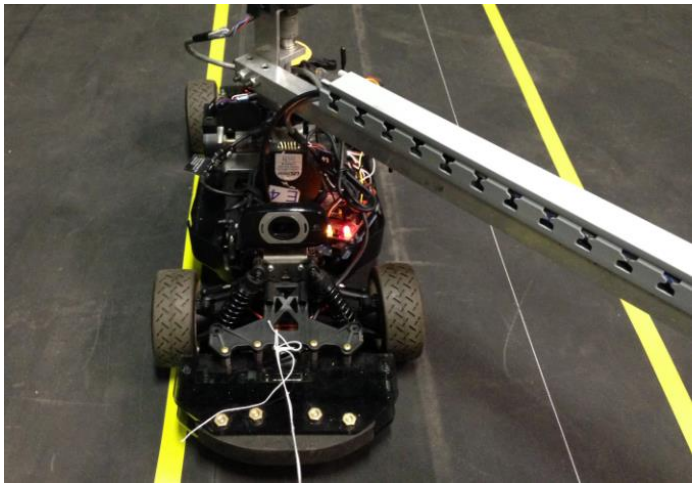

Figure 30 Error Produced Steady State Position

The correct wheel alignment in that position should be turned slightly to the left, as the control algorithm is compensating for the 4 inch error to the right.

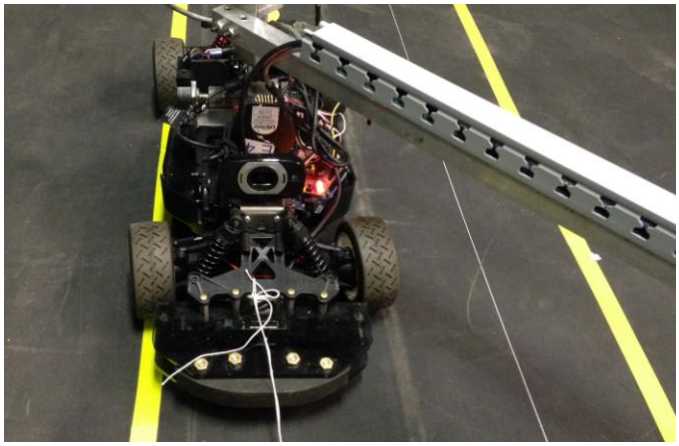

Figure 31 Desired Wheel Position in Same Location

Through testing of how the steering wheels react to forced movements, while operating under the control algorithm, it is concluded that the wheels can become misaligned due to a slight force on the tires themselves. If the car is in the center position, with the wheels straight and the front wheels are forcibly turned to the right for a brief moment, they will attempt to return back to straight, but a slight turn to the right will remain, until the signal is reset.

However the error is most likely electrical instead of mechanical as mentioned in the paragraph above. The 5V control signal is sent to the Arduino Uno over a relatively large distance, 40 in., without properly shielded wires. The current set up uses simple 24 gauge wire, which may falter at times. The error in electrical signal could cause misalignment of the wheels until they are reset.

Chapter 5

Future Considerations

The PURRS test bed is equipped for a more in depth look at methods for positioning and control methods. There is a need to develop a control structure for the speed of the vehicle so it does not have to rely on the tow rope to remain in position, but can work under the vehicles own power. This change might have an effect on the dynamics of how the vehicle turns and responds to wide lateral disturbances.

There are also additional encoders on the vehicle to measure the roll, pitch, and yaw, of the vehicle to aid in the analysis of the vehicles dynamics. Future work could be conducted that utilize these sensors in a feedback control algorithm.

Available are additional sensors to assist in vehicle positioning, including: an in-vehicle, forward facing camera for lane detection; an overhead camera for simulated GPS tracking with fiducials; 2 magnetic sensors mounted on vehicle bumpers for magnetic guidance. These systems are available through USB interfacing with ROS using a Linux-based operating system. Refinement of any of these systems can lead to more research on the reliability of one positioning method vs the other. This would be a major new utilization of rolling-roadway simulators, and is an obvious next step in this research.

Appendix A

Arduino Due Encoder Code

```
/*
Code to read multiple (seven) encoders with the Arduino DUE. The Arduino DUE is a 3.3V platform but has several advantages over the Arduino UNO, namely a faster processor (84 MHz vs. 16 MHz) and it natively has interrupt capabilities on all pins. The syntax for attaching interrupts is slightly different and the Arduino website should be consulted.

Code Core written by: Jesse Pentzer, The Pennsylvania State University
Code Modified by: Anthony Mangus, The Pennsylvania State University
Code Modified by: Raveen Fernando, The Pennsylvania State University

Edits:
8/21/13: Start of the edit tracking (better late than never)
8/22/13: finished/commented the more robust communication protocol, specifically for [ PC ----> arduino ] communication
*/

// Assign your channel in pins
#define CHANNEL_A1_PIN 26
#define CHANNEL_B1_PIN 27
#define ENC_1_INT_PIN 32
#define CHANNEL_A2_PIN 28
#define CHANNEL_B2_PIN 29
#define ENC_2_INT_PIN 33
#define CHANNEL_A3_PIN 30
#define CHANNEL_B3_PIN 31
#define ENC_3_INT_PIN 35
#define CHANNEL_A4_PIN 52
#define CHANNEL_B4_PIN 53
#define ENC_4_INT_PIN 46
#define CHANNEL_A5_PIN 50
#define CHANNEL_B5_PIN 51
#define ENC_5_INT_PIN 47
#define CHANNEL_A6_PIN 48
#define CHANNEL_B6_PIN 49
#define ENC_6_INT_PIN 44
#define CHANNEL_A7_PIN 34
#define CHANNEL_B7_PIN 38
#define ENC_7_INT_PIN 40

volatile long unCountShared1;
volatile long unCountShared2;
volatile long unCountShared3;
volatile long unCountShared4;
volatile long unCountShared5;
volatile long unCountShared6;
volatile long unCountShared7;
char incomingDataBuffer[3];
char dataTransmit;

int intRef1 = 0;
int intRef2 = 0;
int intRef3 = 0;
int intRef4 = 0;
```

```
int intRef5 = 0;
int intRef6 = 0;
int intRef7 = 0;
float deg1;
float deg2;
float rad1;
float rad2;
float L1 = 36;
float L2 = 36.5;
float x;
int x_int;
const float pi=3.14;

void setup()
{
  Serial.begin(115200);
  Serial1.begin(115200);

  //attach the interrupts
attachInterrupt(CHANNEL_A1_PIN, channelA1,CHANGE);
attachInterrupt(CHANNEL_B1_PIN, channelB1,CHANGE);
attachInterrupt(ENC_1_INT_PIN, interrupt1,CHANGE);
attachInterrupt(CHANNEL_A2_PIN, channelA2,CHANGE);
attachInterrupt(CHANNEL_B2_PIN, channelB2,CHANGE);
attachInterrupt(ENC_2_INT_PIN, interrupt2,CHANGE);
attachInterrupt(CHANNEL_A3_PIN, channelA3,CHANGE);
attachInterrupt(CHANNEL_B3_PIN, channelB3,CHANGE);
attachInterrupt(ENC_3_INT_PIN, interrupt3,CHANGE);
attachInterrupt(CHANNEL_A4_PIN, channelA4,CHANGE);
attachInterrupt(CHANNEL_B4_PIN, channelB4,CHANGE);
attachInterrupt(ENC_4_INT_PIN, interrupt4,CHANGE);
attachInterrupt(CHANNEL_A5_PIN, channelA5,CHANGE);
attachInterrupt(CHANNEL_B5_PIN, channelB5,CHANGE);
attachInterrupt(ENC_5_INT_PIN, interrupt5,CHANGE);
attachInterrupt(CHANNEL_A6_PIN, channelA6,CHANGE);
attachInterrupt(CHANNEL_B6_PIN, channelB6,CHANGE);
attachInterrupt(ENC_6_INT_PIN, interrupt6,CHANGE);
attachInterrupt(CHANNEL_A7_PIN, channelA7,CHANGE);
attachInterrupt(CHANNEL_B7_PIN, channelB7,CHANGE);
attachInterrupt(ENC_7_INT_PIN, interrupt7,CHANGE);

}

void loop()
{

  // create local variables to hold a local copies of the channel inputs
  // these are declared static so that thier values will be retained
  // between calls to loop.

  static long unCount1;
  static long unCount2;
  static long unCount3;
  static long unCount4;
  static long unCount5;
  static long unCount6;
  static long unCount7;

  noInterrupts(); // turn interrupts off quickly while we take local copies of the shared variables
```

```
  unCount1 = unCountShared1;
  unCount2 = unCountShared2;
  unCount3 = unCountShared3;
  unCount4 = unCountShared4;
  unCount5 = unCountShared5;
  unCount6 = unCountShared6;
  unCount7 = unCountShared7;


  rad1=(pi/4000)*unCount1;
  deg1=(9/200)*unCount1;
  rad2=(pi/4000)*unCount2;
  deg2=(9/200)*unCount2;
  x=L1*sin(rad1)+L2*sin(rad1+rad2+(pi/2))-36.5;
  x_int=x*100;

  Serial.print(deg1);
  Serial.print(" ");
  Serial.print(deg2);
  Serial.print(" ");
  Serial.print(x);
  Serial.print(" ");
  Serial.println(x_int);


  Serial1.println(x_int);



  interrupts(); // turns interrupts back on from the noInterrupts() command

//  /*
//  This section is for communication via USB with ROS, which is for future consideration of the project.
//  Serial communication protocol for checking incoming data. This block of code looks for a three unit long
//  incoming piece of data on the serial (USB) interface. This data is brought in as a three value long character
//  array. The messages that are useful for this code are R0S (with a zero) to stop the flow of data and R1S (with a one)
to
//  start the flow of data. The idea is the first character is the TRANSMITTING device (designed for use with ROS,
//  therefore the R), the middle is the data, and the last character is a STOP (akin to the STOP used in telegrams,
//  therefore the S).
//  */
//  if (Serial.available() == 3){ // checks for three units of data on the serial buffer
//    Serial.readBytes(incomingDataBuffer, 3); //reads the three units of data into a char[3] variable
//
//    /* For Debugging within the arduino serial monitor
//    Serial.println("Data String Received");
//    Serial.println(incomingDataBuffer); //shows the char[3] variable
//    Serial.println("Data Parts");
//    Serial.println(incomingDataBuffer[0]); //shows the first unit in the char[3]
//    Serial.println(incomingDataBuffer[1]); //shows the second unit in the char[3]
//    Serial.println(incomingDataBuffer[2]); //shows the last unit in the char[3]
//    */
//
//    if (incomingDataBuffer[0] == 'R'){ // checks for the start character "R" (from ROS)
//      if (incomingDataBuffer[2] == 'S'){ // checks for the end character "S" (STOP)
//        dataTransmit=incomingDataBuffer[1]; // puts the data of the message into the dataTransmit variable
//
//        /* For Debugging within the arduino serial monitor
//        Serial.println("Data Received");
//        Serial.println(dataTransmit); // shows the data from the message
```

```
//       */
//     }
//   }
//   Serial.read(); // this command is CRITICAL for the operation of the serial communication. This line ditches any
data
//   // left in the arduino serial buffer to clear it out for the next command
//  }
//  if (dataTransmit == '1'){ //checks to see if the arduino should send the encoder data
//   Serial.print("A");
//   Serial.print(" : ");
//   Serial.print(unCount1);
//   Serial.print(" : ");
//   Serial.print(unCount2);
//   Serial.print(" : ");
//   Serial.print(unCount3);
//   Serial.print(" : ");
//   Serial.print(unCount4);
//   Serial.print(" : ");
//   Serial.print(unCount5);
//   Serial.print(" : ");
//   Serial.print(unCount6);
//   Serial.print(" : ");
//   Serial.print(unCount7);
//   Serial.print(" : ");
//   Serial.println("S");
//  }
}


// simple interrupt service routines

//Encoder 1 interrupts
void channelA1()
{
  if (digitalRead(CHANNEL_A1_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_B1_PIN) == LOW)
    {
      unCountShared1++;
    }
    else
    {
      unCountShared1--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_B1_PIN) == HIGH)
    {
      unCountShared1++;
    }
    else
    {
      unCountShared1--;
    }
  }
}

void channelB1()
{
```

```
  if (digitalRead(CHANNEL_B1_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_A1_PIN) == HIGH)
    {
      unCountShared1++;
    }
    else
    {
      unCountShared1--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_A1_PIN) == LOW)
    {
      unCountShared1++;
    }
    else
    {
      unCountShared1--;
    }
  }
}

void interrupt1()
{
  unCountShared1 = 0;
}

//Encoder 2 interrupts
void channelA2()
{
  if (digitalRead(CHANNEL_A2_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_B2_PIN) == LOW)
    {
      unCountShared2++;
    }
    else
    {
      unCountShared2--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_B2_PIN) == HIGH)
    {
      unCountShared2++;
    }
    else
    {
      unCountShared2--;
    }
  }
}

void channelB2()
{
  if (digitalRead(CHANNEL_B2_PIN) == HIGH)
  {
```

```
    if (digitalRead(CHANNEL_A2_PIN) == HIGH)
    {
      unCountShared2++;
    }
    else
    {
      unCountShared2--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_A2_PIN) == LOW)
    {
      unCountShared2++;
    }
    else
    {
      unCountShared2--;
    }
  }
}

void interrupt2()
{
  unCountShared2 = 0;
}

//Encoder 3 interrupts
void channelA3()
{
  if (digitalRead(CHANNEL_A3_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_B3_PIN) == LOW)
    {
      unCountShared3++;
    }
    else
    {
      unCountShared3--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_B3_PIN) == HIGH)
    {
      unCountShared3++;
    }
    else
    {
      unCountShared3--;
    }
  }
}

void channelB3()
{
  if (digitalRead(CHANNEL_B3_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_A3_PIN) == HIGH)
    {
```

```
      unCountShared3++;
    }
    else
    {
      unCountShared3--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_A3_PIN) == LOW)
    {
      unCountShared3++;
    }
    else
    {
      unCountShared3--;
    }
  }
}

void interrupt3()
{
  unCountShared3 = 0;
}

//Encoder 4 interrupts
void channelA4()
{
  if (digitalRead(CHANNEL_A4_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_B4_PIN) == LOW)
    {
      unCountShared4++;
    }
    else
    {
      unCountShared4--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_B4_PIN) == HIGH)
    {
      unCountShared4++;
    }
    else
    {
      unCountShared4--;
    }
  }
}

void channelB4()
{
  if (digitalRead(CHANNEL_B4_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_A4_PIN) == HIGH)
    {
      unCountShared4++;
    }
```

```
    else
    {
      unCountShared4--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_A4_PIN) == LOW)
    {
      unCountShared4++;
    }
    else
    {
      unCountShared4--;
    }
  }
}

void interrupt4()
{
  unCountShared4 = 0;
}

//Encoder 5 interrupts
void channelA5()
{
  if (digitalRead(CHANNEL_A5_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_B5_PIN) == LOW)
    {
      unCountShared5++;
    }
    else
    {
      unCountShared5--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_B5_PIN) == HIGH)
    {
      unCountShared5++;
    }
    else
    {
      unCountShared5--;
    }
  }
}

void channelB5()
{
  if (digitalRead(CHANNEL_B5_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_A5_PIN) == HIGH)
    {
      unCountShared5++;
    }
    else
    {
```

```
      unCountShared5--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_A5_PIN) == LOW)
    {
      unCountShared5++;
    }
    else
    {
      unCountShared5--;
    }
  }
}

void interrupt5()
{
  unCountShared5 = 0;
}

//Encoder 6 interrupts
void channelA6()
{
  if (digitalRead(CHANNEL_A6_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_B6_PIN) == LOW)
    {
      unCountShared6++;
    }
    else
    {
      unCountShared6--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_B6_PIN) == HIGH)
    {
      unCountShared6++;
    }
    else
    {
      unCountShared6--;
    }
  }
}

void channelB6()
{
  if (digitalRead(CHANNEL_B6_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_A6_PIN) == HIGH)
    {
      unCountShared6++;
    }
    else
    {
      unCountShared6--;
    }
```

```
  }
  else
  {
    if (digitalRead(CHANNEL_A6_PIN) == LOW)
    {
      unCountShared6++;
    }
    else
    {
      unCountShared6--;
    }
  }
}

void interrupt6()
{
  unCountShared6 = 0;
}

//Encoder 7 interrupts
void channelA7()
{
  if (digitalRead(CHANNEL_A7_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_B7_PIN) == LOW)
    {
      unCountShared7++;
    }
    else
    {
      unCountShared7--;
    }
  }
  else
  {
    if (digitalRead(CHANNEL_B7_PIN) == HIGH)
    {
      unCountShared7++;
    }
    else
    {
      unCountShared7--;
    }
  }
}

void channelB7()
{
  if (digitalRead(CHANNEL_B7_PIN) == HIGH)
  {
    if (digitalRead(CHANNEL_A7_PIN) == HIGH)
    {
      unCountShared7++;
    }
    else
    {
      unCountShared7--;
    }
  }
  else
```

```
  {
    if (digitalRead(CHANNEL_A7_PIN) == LOW)
    {
      unCountShared7++;
    }
    else
    {
      unCountShared7--;
    }
  }
}

void interrupt7()
{
  unCountShared7 = 0;
}
```

Appendix B

Arduino Uno RC Code

```
// Required Libraries
#include <Servo.h>
#include <PinChangeInt.h>

/*
This code is used to control the new vehicle designed for use on the Pennsylvania
State University Rolling Roadway Simulator. The vehicle is an off the shelf R/C car
that has been lightly modified to ease in control and for sensor mounts. The vehicle
is controlled in an extremely simple manner: the arduino is nothing but a gate between
the reciever (which receives manual commands from the transmitter given by the operator)
and the servo/electronic speed control (ESC). If the switch on the transmitter is
activated, the arduino ignores the commands from the receiver and uses commands given
from ROS. This achitecture gives the operator the ability to manually control the vehicle,
or let ROS control the vehicle. This architecture will also revert to manual mode if the
transmitter is turned off, effectively bringing the vehicle to a stop. There is also an LED
the arduino shield to determine which mode the arduino is in.

Copyright Anthony Mangus 2013, The Pennsylvania State University
Modified Raveen Fernando 2014, The Pennsylvania State University

*/

int LED = 12;

// Defines the pins used for servo, esc, and reciever
int rxservopin = 4; // receiver servo pin
int rxescpin = 8; // receiver ESC pin
int rxcontrolstat = 13; // receiver 3rd channel (control button) pin

int val;

int steeringservopin = 5; // servo pin
int ESCservopin = 9; // ESC pin

Servo steeringservo; // creates a servo object for the steering servo
Servo ESCservo; // creates a servo object for the ESC (which works like a servo)

int current;
char incomingDataBuffer[20];
char dataTransmit;
int numbytes;

volatile long countservo;
volatile long posservoint;

volatile long countesc;
volatile long posescint;

volatile long countch3;
volatile long posch3int;

void setup() {

  Serial.begin(115200);
```

```
  pinMode(LED, OUTPUT);

  steeringservo.attach(steeringservopin);
  steeringservo.write(90);
  ESCservo.attach(ESCservopin);
  ESCservo.write(90);
  delay(2000);

  /*
  This code uses interrupts to read the three signals from the receiver.
  This is done by looking for the leading and trailing edge of the analog
  signal and converting it to calculate the position. The PinChangeInt library
  allows any digital pin on the arduino to have interrupt capability
  */

  PCintPort::attachInterrupt(rxservopin, servostatus, CHANGE);
  PCintPort::attachInterrupt(rxescpin, escstatus, CHANGE);
  PCintPort::attachInterrupt(rxcontrolstat, ch3status, CHANGE);


}


void loop() {
  //ESCservo.write(90);
  //current=steeringservo.read();

  static long posservo;
  static long posesc;
  static long posch3;
  static int RXservo = 1500;
  static int ROSESC = 1500;
  static int ROSstat = 1500;
  boolean gotdata = 0;



  noInterrupts();

  posservo = posservoint;
  posesc = posescint;
  posch3 = posch3int;
  interrupts();


  int incoming = Serial.available();
  if (incoming != 0)
  {
    val = Serial.parseInt();  //Reads integers as integer rather than ASCI. Anything else returns 0
    RXservo = 1500 + val;       //Gain=1
  }


  /*
  Following code section is for ROS communication via Serial Communication.
  Serial communication protocol for checking incoming data. This block of code looks for a three unit long
  incoming piece of data on the serial (USB) interface. This data is brought in as a three value long character
  array. The messages that are useful for this code are R0S (with a zero) to stop the flow of data and R1S (with a one) to
  start the flow of data. The idea is the first character is the TRANSMITTING device (designed for use with ROS,
  therefore the R), the middle is the data, and the last character is a STOP (akin to the STOP used in telegrams,
```

```
 therefore the S).
 */
 //  if (Serial.available() != 0){ // checks for three units of data on the serial buffer
 //    numbytes = Serial.readBytesUntil('\r',incomingDataBuffer, sizeof(incomingDataBuffer)); //reads the three units of
data into a char[3] variable
 //
 //    //For Debugging within the arduino serial monitor
 //    //Serial.println(numbytes);
 //    //Serial.println("Data String Received");
 //    //Serial.println(incomingDataBuffer); //shows the char[3] variable
 //    /*Serial.println("Data Parts");
 //    Serial.print(incomingDataBuffer[0]); //shows the first unit in the char[3]
 //    Serial.println(incomingDataBuffer[1]);
 //    Serial.println(incomingDataBuffer[2]); //shows the second unit in the char[3]
 //    Serial.print(incomingDataBuffer[3]); //shows the last unit in the char[3]
 //    Serial.println(incomingDataBuffer[4]);
 //    */
 //
 //    if (incomingDataBuffer[0] == 'R' && incomingDataBuffer[1] == ':'){ // checks for the start character "R" (from
ROS)
 //
 //      if (incomingDataBuffer[numbytes-2] == ':' && incomingDataBuffer[numbytes-1] == 'S'){ // checks for the end
character "S" (STOP)
 //        //dataTransmit=incomingDataBuffer[2]; // puts the data of the message into the dataTransmit variable
 //        //Serial.println("Got :S");
 //        //For Debugging within the arduino serial monitor
 //
 //        /* For troubleshooting
 //        //Serial.println("Data Received");
 //        Serial.print("A:");
 //        for (int a = 2; a < numbytes-2; a++){
 //          Serial.print(incomingDataBuffer[a]); // shows the data from the message
 //        }
 //        Serial.println(":S");
 //        */
 //        int count = 0;
 //        int ROSvarcount = 0;
 //        char dataConvertBuffer[4];
 //        for (int a = 2; a < numbytes-1; a++){
 //          if(incomingDataBuffer[a] != ':'){
 //            dataConvertBuffer[count] = incomingDataBuffer[a];
 //            count++;
 //            //Serial.println(count);
 //          }
 //          else {
 //            count = 0;
 //            // note: the atoi function converts a string to an integer
 //            switch (ROSvarcount){
 //              case 0:
 //                ROSservo = atoi(dataConvertBuffer); // captures the servo position from ROS
 //                //Serial.println(ROSservo);
 //                break;
 //              case 1:
 //                ROSESC = atoi(dataConvertBuffer); // captures the ESC positon from ROS
 //                //Serial.println(ROSESC);
 //                break;
 //              case 2:
 //                ROSstat = atoi(dataConvertBuffer); // captures the control status from ROS
 //                //Serial.println(ROSstat);
 //                break;
```

```
//           }
//           ROSvarcount++;
//           char dataConvertBuffer[4];
//         }
//
//          //Serial.print(incomingDataBuffer[a]); // shows the data from the message
//        }
//
//        //Serial.println(posch3);
//        // this next section prints the ROS command back to ROS if the vehicle is in ROS mode
//        if (posch3 < 1500 && posch3 > 800) // range of the control button singal to put the vehicle in ROS mode
//          {
//            //Serial.println("Data Out");
//            Serial.print("A:");
//            Serial.print(ROSservo);
//            Serial.print(":");
//            Serial.print(ROSESC);
//            Serial.print(":");
//            Serial.print(posch3);
//            Serial.println(":S");
//          }
//
//
//
//      }
//    }
//    Serial.read(); // this command is CRITICAL for the operation of the serial communication. This line ditches any
data
//    // left in the arduino serial buffer to clear it out for the next command
//  }
//
//  /*for (int angle=40; angle <140; angle++){
//    steeringservo.write(angle);
//    //Serial.println(angle);
//    delay(10);
//  }
//  */

  // this next section prints the manual positions back to ROS if the vehicle is in manual mode
  if (posch3 > 1500 && posch3 < 2000) // range of the control button singal to put the vehicle in manual mode
  {
    digitalWrite(LED, LOW);

    ESCservo.writeMicroseconds(posesc); // sets the ESC to the value sent from the receiver
    steeringservo.writeMicroseconds(posservo); // sets the servo to the value sent from the receiver

    //Serial.println("Data Out");
    //      Serial.print("A:");
    //      Serial.print(posservo);
    //      Serial.print(":");
    //      Serial.print(posesc);
    //      Serial.print(":");
    //      Serial.print(posch3);
    //      Serial.println(":S");

  }

  if (posch3 < 1500 && posch3 > 800) // range of the control button singal to put the vehicle in ROS mode
  {
    digitalWrite(LED, HIGH);
```

```
    ESCservo.writeMicroseconds(posesc); // sets the ESC to the value sent from ROS
    steeringservo.writeMicroseconds(RXservo); // sets the servo to the value sent from ROS
  }

}

/*
The following funtions are the interrupt driven reading of the servo/ESC/Channel 3 status. This is done
by starting a clock when the leading edge of the signal and stopping when the trailing edge is found.
*/

void servostatus() {
  if (digitalRead(rxservopin) == HIGH) {
    countservo = micros(); //positive edge
  }
  else {
    posservoint = micros() - countservo;
  }
}

void escstatus() {
  if (digitalRead(rxescpin) == HIGH) {
    countesc = micros(); //positive edge
  }
  else {
    posescint = micros() - countesc;
  }
}

void ch3status() {
  if (digitalRead(rxcontrolstat) == HIGH) {
    countch3 = micros(); //positive edge
  }
  else {
    posch3int = micros() - countch3;
  }
}
```

BIBLIOGRAPHY

[1] Brennan, S., and A. Alleyne. 2001. Using a scale testbed: Controller design and evaluation. IEEE Control Systems 21, (3): 15.

[2] Brennan, S., and A. Alleyne. 2000. The illinois roadway simulator: A mechatronic testbed for vehicle dynamics and control. IEEE/ASME Transactions on Mechatronics 5, (4): 349-359.

[3] Brennan, S., and A. Alleyne. 1999. A scaled testbed for vehicle control: The IRS. Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No.99CH36328) 1, : 327-332 vol. 1.

[4] Kachroo, P., K. Ozbay, R. G. Leonard, and C. Unsal. 1995. Flexible low-cost automated scaled highway (FLASH) laboratory for studies on automated highway systems. 1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century 1, : 771-776 vol.1
.

[5] Brennan, S., A. Alleyne, and M. DePoorter. 1998. The illinois roadway simulator-a hardware-in-the-loop testbed for vehicle dynamics and control. Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207) 1, : 493-497 vol.1

[6] Mangus, Anthony J., Sean Brennan, and Schreyer Honors College. 2013. Robust vehicle localization using GPS, in-vehicle camera, magnetic guidance and kalman filtering. Ph.D. diss., Pennsylvania State University.

[7] Brennan, S. N. (1999) Modeling and Control Issues Associated with Scaled Vehicles, Master's thesis, University of Illinois at Urbana-Champaign.

[8] Lapapong, Sittikorn. 2007. Vehicle similitude modeling and validation of the pennsylvania state university rolling roadway simulator. Ph.D. diss., .

[9] Woodley, R. and L. Acar (2004) "Autonomous Control of a Scale Model of a Trailer-Truck Using an Obstacle-Avoidance Path-Planning Hierarchy," in Proceedings of the 2004 American Control Conference, vol. 4, Boston, MA, pp. 3399–3404.

[10] Langer, W. (1995) "Validation of Flat Surface Roadway Technology," SAE Technical Paper Series, (950310).

[11] Schultz, G., I. Tong, K. Kefauver, and J. Ishibashi (2005) "Steering and Handling Testing Using Roadway Simulator Technology," International Journal of Vehicle Systems Modeling and Testing, 1(1/2/3), pp. 32–47.

[12] Sampei, M., T. Tamura, T. Kobayashi, and N. Shibui (1995) "Arbitrary Path Tracking Control of Articulated Vehicles Using Nonlinear Control Theory," IEEE Transactions on Control Systems Technology, 3(1), pp. 125–131.

[13] Henry, R. D. (2001) Automatic Ultrasonic Headway control for a Scaled Robotic Car, Master's thesis, Virginia Polytechnic Institute and State University.

[14] Whitehead, R., B. Clark, M. Breland, K. Lambert, D. M. Bevly, and G. Flowers (2005) "Scaled Vehicle Electronic Stability Control," in ESV International Collegiate Student Safety Technology Design Competition.

ACADEMIC VITA

Raveen Fernando
751 Stratford Dr. State College PA, 16801 / Raveen.L.Fernando@gmail.com

Education

The Pennsylvania State University
University Park, PA
Schreyer Honors College Fall 2014
B.S in Mechanical Engineering, Minor in Spanish

Computer Skills

SolidWorks (Associate's Certificate)	C++	Simulink
AutoCAD	MATLAB	Excel

Work History

Internship- Ford Motor Company Summer '14
- Analyzed behavior and accuracy of lane detection software used for autonomous vehicles.

Researcher-Intelligent Vehicles and Systems Group Spring 2013-Present
- Helped build a 360 degree multi-degree of freedom simulator for trucks and tractors
- Developing a test bed for RC controlled autonomous vehicles

Internship- ArcelorMittal USA Summer 2011
- Created electronic parts databases for mobile equipment, which increased the efficiency of repairs.
- Evaluated and designed safety devices for working at heights, which increased the safety rating of the plant, and decreased the insurance deductible.

Internship- Caravansar, Gestión Cultural Spain. Spring 2012
- Captured images to document the clients, classes, and theatrical events.
- Edited and produced the images for promotional uses of the company.

Internship- Center County Democratic Campaign Headquarter Summer 2008
- Oversaw canvassing in sectors of State College to get accurate voter positions.
- Visited local high school to educate seniors on the voting process and to register them to vote

Activities

Penn State IFC/Panhellenic Dance Marathon 2013-2014
- Merchandise Licensing Captain
  - Oversaw all merchandise design requests for THON organizations
  - Protected the trademarks of THON and associated groups

Beta Theta Pi Fraternity 2010-Present

- Community Service Chair 2011
    - Organized brotherhood-wide community service events
    - Ensured brothers fulfill community service hours

Penn State Ballroom Dance Team 2012-Present

NAACP 2010

- Fundraising Events Committee
    - Contacted businesses about prospects for future fundraisers

Achievements

Louis A. Harding Memorial Scholarship 2012, 2013

Beta Theta Pi Men of Principle Scholarship 2010

Penn State President's Freshman Award 2010

Academic Excellence Scholarship 2009-2013

Golden Key International Honour Society 2010-Present

Dean's List