

AUTONOMOUS CONTROL SYSTEM WITH DRAG MEASUREMENT CAPABILITY FOR A WIND TUNNEL MODEL VEHICLE

Shadeed Mahmud

Bachelor of Engineering
Electronic Engineering Major



Department of Electronic Engineering
Macquarie University

June 27, 2016

Supervisor: Dr. Sammy Diasinos
Co-Supervisor: Dr. David Inglis

ACKNOWLEDGMENTS

I would like to acknowledge and sincerely thank Dr. Sammy Diasinos and Dr. David Inglis for their support, advice and enthusiasm during my thesis at Macquarie University.

STATEMENT OF CANDIDATE

I, Shadeed Mahmud, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Electronic Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment in any academic institution.

Student's Name: Shadeed Mahmud

Student's Signature: *Shadeed*

Date: 27 June 2016

ABSTRACT

In order to understand the aerodynamics of a model, wind tunnel testing is crucial and in wind tunnel testing there needs to be an accurate correlation between the environment inside the tunnel and the environment on the road. In the automotive industry, to make the test environment more realistic, a model vehicle being analyzed is now placed on a rolling road to simulate a moving ground. Inside wind tunnels as such, these models are supported by structures such as struts so that it can be held in place and a system of balance connected to these struts is used to measure the aerodynamic forces such as drag. These structures, however, interfere with wind tunnel results and it is standard practice to correct wind tunnel results for a model with and without presence of support structures. In order to make the wind tunnel test environment more realistic and results more accurate thus enabling us to understand the complete aerodynamics of a model, an advanced alternative is proposed and researched in this project. This project is about the development of an autonomous control system that can maintain a model vehicles position inside a wind tunnel over a moving ground with high degree of accuracy and measure the drag the vehicle is experiencing. This thesis reviews existing technology in the field of wind tunnel testing and autonomous driving and discusses the development of an advanced mathematical algorithm for the aimed control system.

Contents

Acknowledgments	iii
Abstract	vii
Table of Contents	ix
List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Project Overview	1
1.1.1 Project Specification	1
1.1.2 Aims and Goals	2
1.1.3 Project Timeline	3
2 Literature Review	5
2.1 Background	5
2.1.1 Current Wind Tunnels and Testing Methodology	6
2.1.2 Alternative Support and Testing Methodology	7
2.2 Autonomous Driving	8
2.2.1 Control Algorithms	9
2.2.2 Autonomous Control System for Wind Tunnel Model	12
2.3 Sensor Review	13
2.3.1 Arduino Compatible Sensors	13
2.4 Summary	14
3 Development of an Advanced Control System Algorithm	15
3.1 Introduction	15
3.2 Vehicle Steering Dynamics	15
3.3 Mathematical Approach to Realizing Steering Dynamics	17
3.3.1 Simulation of Steering Dynamics	21
3.4 The Normal Mode	22
3.5 The Tolerance mode	24

3.6	Determination of the Drag Force	27
3.7	Chapter Summary	28
4	Specific Criteria Based Sensor Analysis	31
4.1	Introduction	31
4.2	Sensor Research Criteria	31
4.3	Sensor and Component Selection	32
4.4	Sensor Test	34
4.4.1	LDR Array and Laser Dot Combination	35
4.4.2	Ultrasonic Sensor	37
4.5	Summary	40
5	Implementation of the Control System	41
5.1	Introduction	41
5.2	The Technique of Implementation	41
5.3	Sensor Mount Design	43
5.4	The Final Control System	45
5.4.1	Mapping The Calculated Radius to a Steering Angle	48
5.5	Summary	51
6	Interfacing with the Control System	53
6.1	Introduction	53
6.2	Interfacing Over a Terminal Program	53
6.3	Graphical User Interface	55
6.4	Summary	58
7	Conclusions	61
A	HC-SR04 Datasheet	65
B	Arduino Code	67
B.1	Version 1	67
B.2	Version 2	76
C	Consultation Meetings Attendance Form	89
	Bibliography	91

List of Figures

1.1	Project Specifications	2
1.2	Project Timeline	4
2.1	Difference in wind tunnel situation and real world situation [11]	7
2.2	How self-driving car see the road [17]	8
2.3	Lidar data before and after calibration [9]	9
2.4	Optimum Trajectory vs Hybrid A trajectory [14]	10
2.5	Image from the front camera (left), image sampled by Ralph (right) [15] . .	11
2.6	Ralph's technique for determining road curvature [15]	11
2.7	Toe-in setting of the steering wheels [21]	12
3.1	Vehicle steering dynamics [21]	16
3.2	Four different scenarios of vehicle location and orientation	18
3.3	Radius of Curvature	19
3.4	Major triangle ADC and sub-triangle ABD and ABC	20
3.5	Steering Dynamic Simulation	21
3.6	Control algorithm for the normal mode of operation	23
3.7	Control algorithm for the tolerance mode of operation	25
3.8	Complete Control algorithm incorporating the two modes of operation . . .	26
3.9	Different forces acting on the vehicle during operation	27
4.1	Concept of LDRs and laser pointer	34
4.2	Arrangement of LDRs and Laser Dot	35
4.3	Initial Position of LDR	36
4.4	Incremented Position of LDR	36
4.5	LDR values corresponding to their position increment	37
4.6	Ultrasonic Sensor Test Arrangement	38
4.7	Percentage Error in Measurement Using Ultrasound	39
4.8	Measured Distance vs Actual Distance	40
5.1	The Sensor Setup Plan	42
5.2	The model vehicle maintaining its position on the running ground	42
5.3	Circuit schematic for a single LDR	43
5.4	Digital model of the designed sensor mount	44

5.5	3D printed sensor mount	45
5.6	Refined control algorithm	46
5.7	Complete schematic	47
5.8	Model vehicle with the populated sensor mount and all components	48
5.9	Experimental procedure to get relation between steering angle and radius of curvature	49
5.10	Geometric representation for calculation of radius	50
5.11	Radius of curvature vs steering angle	51
6.1	The serial terminal during calibration	54
6.2	The serial terminal during autonomous driving	55
6.3	The idle graphical user interface	56
6.4	The graphical user interface during normal mode of operation	57
6.5	The graphical user interface during tolerance mode of operation	58
A.1	66

List of Tables

3.1	Control system response for small change in vehicle's position	24
4.1	Results of criteria based sensor research [19] [5] [8] [7] [22] [20] [10]	33

Chapter 1

Introduction

Wind tunnel testing to understand the aerodynamics of a vehicle has long been a part of the automotive industry. Having a body that is aerodynamic, such that the air resistance or drag faced by the vehicle is reduced to a minimum where needed and maximized where needed, can drastically improve the performance of a vehicle. In order to understand the aerodynamics of a vehicle, often model vehicle are subjected to wind tunnel testing. The model is placed on a moving ground inside the wind tunnel and subjected to various wind forces. The most common method to maintain the models position on the moving ground is to use metal struts to hold the model in position. The tension experienced by this structures is used to measure the drag experienced by the vehicle [13]. However, due to the absence of these struts in real life, the actual aerodynamics of the car are not obtained [21]. In order to obtain more realistic data, a solution that enables the model to maintain its position in the tunnel and measure drag subsequently without the presence of any physical object within the range of the model that might affect the airflow around the model needs to be implemented. This thesis project looks into the development of an autonomous control system with advanced mathematical algorithm that will enable a model vehicle to maintain its position in a wind tunnel while measuring the drag that the model experience.

1.1 Project Overview

This section discusses the overview of the project including the particular project specifications. The aims and goals of the project that were set in order to meet the project specifications are listed and discussed. Finally, a time-line of completion of expected stages of the project is illustrated by the use of a gantt chart.

1.1.1 Project Specification

The overall aim of this project is to build an autonomous control system capable of maintaining a model vehicle position inside a wind tunnel and measure the drag force experienced by the model vehicle. The model vehicle needs to be at the center of the

moving ground inside the wind tunnel. There is a tolerance range of $\pm 5mm$ making a total of 10mm or 1cm length of total variance that can be present. This is illustrated in figure 1.1 below.

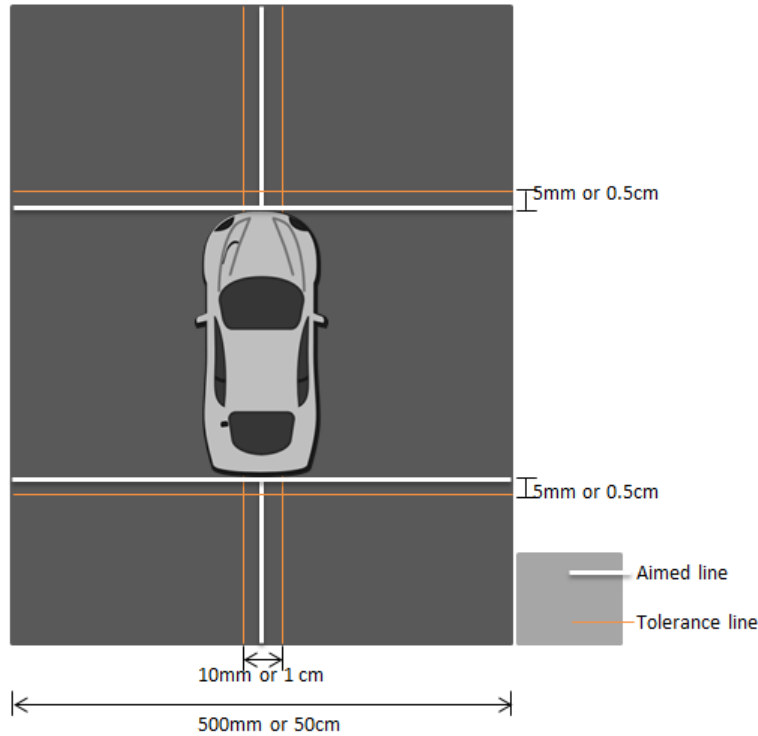


Figure 1.1: Project Specifications

In order to achieve the desired specifications the project was broken down into several stages and goals were set of each of these stages. These goals are discussed in the next section.

1.1.2 Aims and Goals

There were several aims that were set in order to establish a systematic approach to the design process which will lead up to the final goal of this project. The aims are given below.

- Develop a mathematical equation to realize vehicle steering dynamic and form a control algorithm based on the mathematical analysis.
- Research and list sensors that can be used to realize the input parameters to the mathematical algorithm.
- Implement the control system such that the vehicle aligns itself with the central target line from either side of the target line.

- Develop a technique to measure the drag that the model vehicle experience during operation.

1.1.3 Project Timeline

The project is to be carried out over a thirteen week long semester as part of an undergraduate degree. A project schedule was developed in order to maintain and track progress of the project. The project schedule is best illustrated by the aid of a gantt chart. This is given below in figure 1.2.

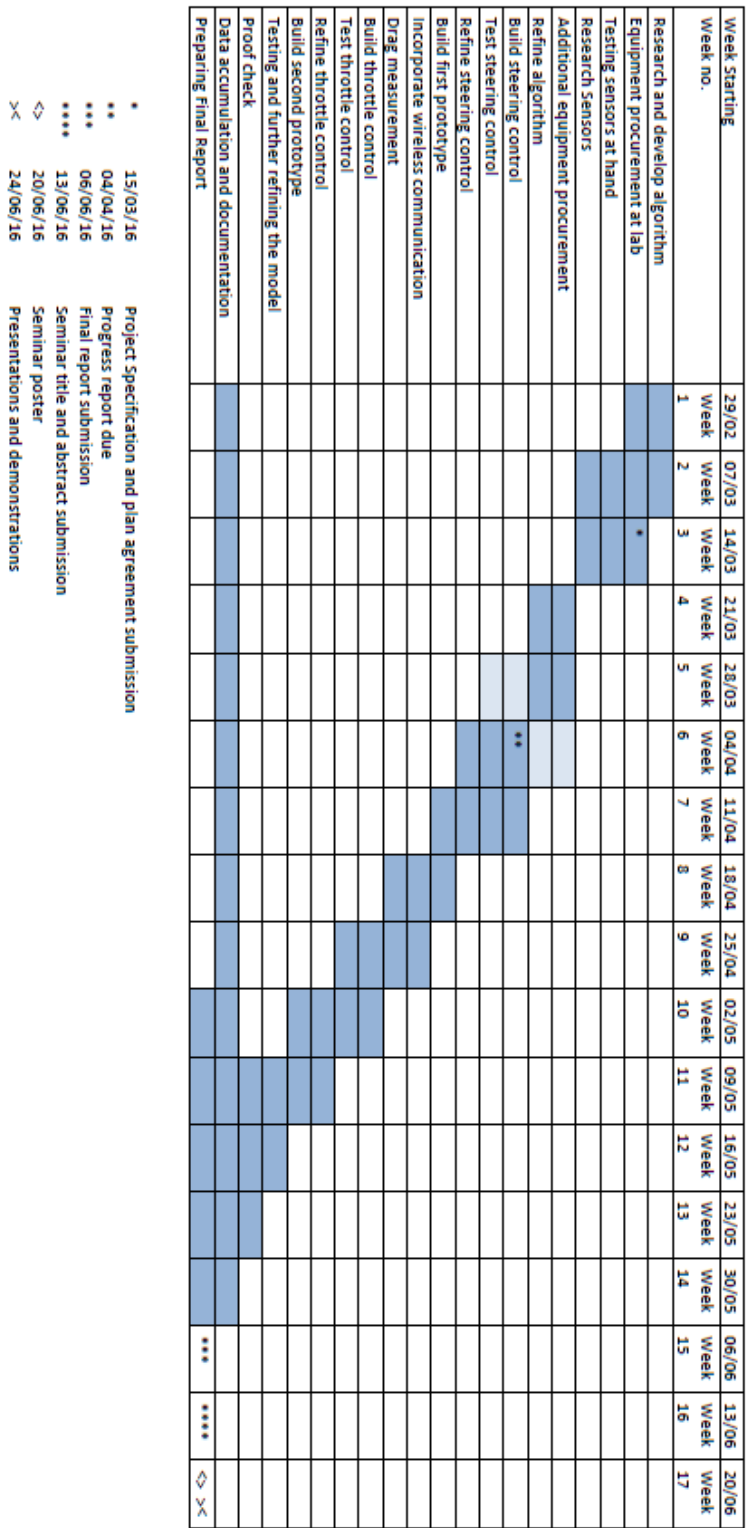


Figure 1.2: Project Timeline

Chapter 2

Literature Review

2.1 Background

This section discusses the reasons behind the aim of this project and what have been done so far. To understand the need of the proposed control system in chapter 1, the history behind aerodynamic research and necessary equipment development needs to be realized. This dates back to the 18th century when Benjamin Robbins, an English mathematician first developed the whirling arm [1] [3]. The whirling arm was truly the first machine used for aerodynamic data accumulation. The first machine had an arm 4 feet long. The arm was spun by a falling weight acting on a pulley and spindle arrangement, however the arm tip could only reach velocities of a few feet per second.

The whirling arm could be used to gather most of the systematic aerodynamic data. However, its flaws were noticed. The working principal of the whirling arm actually set the air in the vicinity in rotary motion which adversely affected the test results. Aircraft models on the end of an arm in effect flew into their own wakes [1]. Due to such strong turbulence, the true relative velocity between the model and air could not be determined. Furthermore, it was extremely difficult to mount instruments and measure the small forces exerted on the model when it was spinning at high speeds [1]. In pursuit of more accurate data something better needed to be developed. It was not until the 19th century that something better had emerged. The first wind tunnel was developed. Frank H. Wenham (1824-1908), a Council Member of the Aeronautical Society of Great Britain, is generally credited with designing and operating the first wind tunnel in 1871 [1] [3]. Wenham had used a whirling arm for his experiments, but was not satisfied with them and found motive to develop something better and with the council backing him with funds, he was able to build the first wind tunnel [1]. The first wind tunnel had a trunk 12 feet long and 18 inches square. It had a steam engine powered fan-blower which propelled air through the tube to the model being analyzed. The wind tunnel managed to direct the air current horizontally and parallel to the course [1]. Tests using these wind tunnels allowed for researchers to understand the aerodynamic forces such as lift, drag and their ratios but these wind tunnels were capable of analyzing only scale models as analyzing full-size models were simply too expensive [1]. So a question still remained whether these

scale model results were applicable to real sized aircrafts until Osborne Reynolds from the University of Manchester managed to demonstrate that the airflow pattern over a scale model would be the same for the full-scale vehicle if a certain flow parameter were the same in both cases. This factor is known as the Reynolds number and it is a basic parameter in the description of all fluid-flow situations [1].

In 1901 a better wind tunnel was developed by the Wright brothers [3]. Their first tunnel consisted of a two-element balance mounted in the airstream, one was a calibrated plane surface and the other was a cambered test surface inclined at the same angle but in the opposite direction. When the wind tunnel was brought up to speed, the vane-type balance turned one way or the other, thereby indicating the relative lifting forces [1]. Improvements in aerodynamic testing and equipment used for these tests continued and the subjects being tested were not limited to airplanes anymore. They were expanded to cars as well when it was realized that proper aerodynamic design affected its performance [6]. In the automobile industry, especially in motor sports such as F1, aerodynamics of vehicles is important down to the point where meticulous precision in aerodynamic design will result in the car being faster by crucial seconds thus making a difference in the final result of a race or so. In 1972, Colin Chapman showed the way ahead for Formula 1. The legendary designer and team boss equipped his Lotus 72 with revolutionary aerodynamics. This resulted in Emerson Fittipaldi in winning the World Championship for Lotus. According to Steven de Groote from F1 Technical, aerodynamics are the most important factor in the design of a Formula One car [6]. Now, when determining the aerodynamics of a vehicle, wind tunnel testing for these vehicles is essential.

2.1.1 Current Wind Tunnels and Testing Methodology

Many types of wind tunnels have been developed, but they all have the purpose of creating a uniform stream of air which passes through a test chamber in which a scale model is mounted. The model is mounted on struts or wires fastened to a system of balances which enable the tunnel operator to measure the forces and moments acting on the model [13]. However the current wind tunnels are more advanced. One major advancement was the use of moving ground planes when analyzing cars [18]. A stationary ground plane causes a boundary layer to build up under the car, and can interfere with the boundary layer of the lower components of the car [18]. Thus influencing the test results. The ground boundary layer can be removed by using a moving belt under the car which causes the wheels to rotate with the belt [18]. This was an effective way of simulating rotating wheels and this simulation was important as cars in Indy and Formula 1 have exposed wheels and neglecting these aerodynamic effects will alter the performance. The moving belt approach is mandatory for vehicles with very low ground clearance or low drag coefficients [18]. Now on a moving belt, the model needs to be held stationary and the existing method of using struts connected to system of balances is applied to hold the vehicle and measure aerodynamic forces. However during actual operation of the vehicle on tracks these struts will not be present and using them in the wind tunnel fails to co-relate the wind tunnel environment to the actual environment. An example of such a

situation is shown below in figure 2.1.

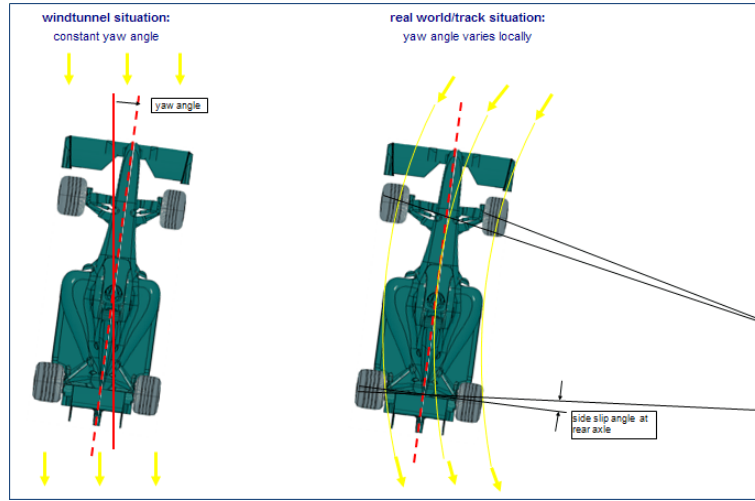


Figure 2.1: Difference in wind tunnel situation and real world situation [11]

From figure 2.1 it can be seen that the yaw angle of attack vary locally [11]. Having a structure holding the vehicle in place does not allow for the dynamic changes in the angle of attack to occur. However if the vehicle could hold its position without the presence of support structures, the dynamic changes would occur and the aerodynamic test results would be more accurate. The presence of support structures also alter the air-flow around the model and therefore affect the results obtained [4] [12]. Therefore, in order to obtain better results, further improvement to the testing equipment or alternative methods have been researched.

2.1.2 Alternative Support and Testing Methodology

In wind tunnel testing of models, interference due to support structure is a problem as old as wind tunnels themselves, and a lot of research has been conducted to establish alternative methods that eliminate errors due to support structure interference [12]. An alternative that have been considered a potential solution for many support system problems for several decades is the use of magnetic suspension and balance systems, or MSBS [12]. In the past, there have been applications of MSBS in small wind tunnels, however the technologies necessary to construct large wind tunnel systems were not available until recent years [12]. While MSBS does appear to be a potential solution, it has its limitations. Some problems and limitations of existing MSBS include roll control, size limitations, reliability, position sensing and calibration. Some further problems specific to large-scale magnetic suspension and balance systems are requirements of high-power electromagnets, high-capacity power supplies, highly sophisticated control systems and high costs [2].

A less expensive proposed alternative that could eliminate support structures in wind tunnel testing is an autonomous control system within the model [21]. The focus of this

project is the autonomous control system that will allow for the model vehicle to control itself on the running belt inside the wind tunnel and also allow us to measure the drag the vehicle is experiencing. The control system will be implemented in a model car. The control system will be able to control the vehicle and maintain its position with a high degree of accuracy and measure the drag the vehicle is experiencing. Since the control system will be implemented on a model vehicle, therefore self-driving cars are reviewed.

2.2 Autonomous Driving

Self-driven car has been an interesting phenomenon in the past years and technological giants and automobile giants such as Google and Mercedes respectively and certain other companies have managed to establish such cars [17]. Since the control system being researched will essentially result in a self-driven car therefore existing technologies, sensors and control algorithms in the field of autonomous driving is reviewed.

In order for a car to be able to perform autonomous driving, it needs to know where it is and what are in it's surroundings and where it needs to go. The car obtains these information by means of a combination of advanced sensors. Figure 2.2 below shows a pictorial representation of how a car capable of autonomous driving sees the road.

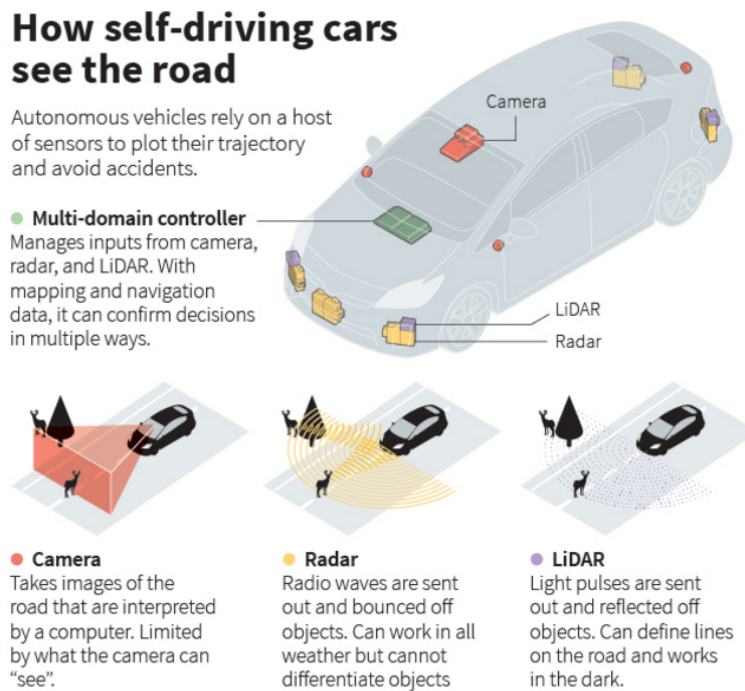


Figure 2.2: How self-driving car see the road [17]

From figure 2.2 above, it can be seen that these car use the following list of sensors.

- Camera

- Radar
- Lidar

A self-driving car uses these sensors along with Global Positioning System (GPS) to sense its surroundings and plan its trajectory [9]. The GPS is used to navigate from an initial location to a final destination and the other set of on-board sensors are used to localize the vehicle with its surroundings. These sensor data are then fed back to a central processing unit where the data is analysed and instructions are outputted to the vehicle for autonomous driving [17]. Autonomous maneuvers such as obstacle detection, obstacle avoidance, maintaining position and adjusting speed are amongst the common tasks that a self-driving vehicle would do. Considering the maneuvers that a self driving vehicle would do, some sensors that could be used to input data to the autonomous control system and certain control algorithms used to process these sensor data is also reviewed in this thesis.

2.2.1 Control Algorithms

This section of the thesis looks into control algorithms and techniques of how autonomous driving has been achieved so far by other institutions. In particular two techniques stand out in terms of how the techniques and research involved can be related to this project. Of the two techniques, one, called Junior [9], was developed at Stanford University as part of an urban competition held by Defense Advanced Research Projects Agency (DARPA) in the United states and the other called Ralph (Rapidly Adapting Lateral Position Handler) [15] was developed by Carnegie Mellon University and Assistware Technology Inc.

Junior

A research vehicle, named Junior, uses a trio of unsupervised algorithms to calibrate a high accuracy, 64-beam rotating LIDAR. The LIDAR is then used to scan the surroundings and the sensor data is used to generate high-resolution maps of the environment which are used used for localization purposes accurate to a centimeter. An example of the effect of the calibration and map generated is shown below in figure 2.3

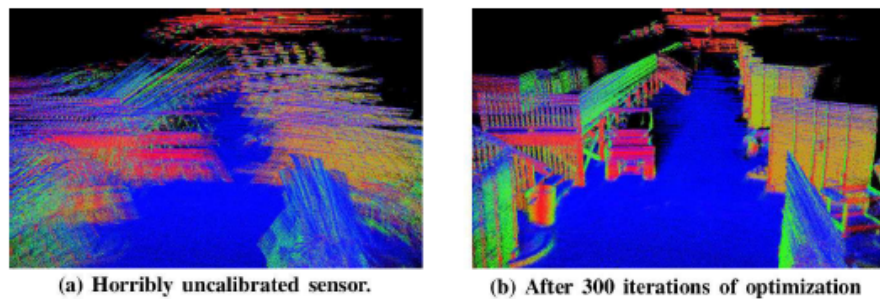


Figure 2.3: Lidar data before and after calibration [9]

Then specialized algorithms for perception and recognition is used to track and classify different obstacles as cyclists, pedestrians, and vehicles. The system is capable of detecting traffic lights as well [9]. The incoming sensor data is used to generate thousands of candidate trajectories per second and the optimal path is chosen dynamically. An example of such a planned trajectory is shown in figure 2.4 below. Once the trajectory has been determined, the controller constantly adjusts the throttle, brake, and steering actuation through proportional, integral and differential (PID) control for lower-level feedback control tasks that allows for the vehicle to maintain the planned trajectory [9] [14].

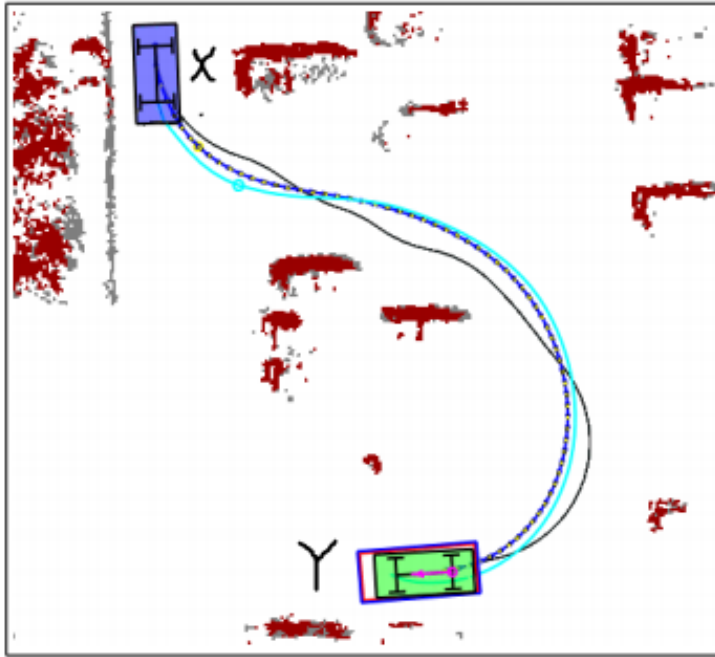


Figure 2.4: Optimum Trajectory vs Hybrid A trajectory [14]

In figure 2.4 above, the vehicle is traveling from location X to location Y. The blue and cyan traces show the optimum trajectory of the front and rear axle respectively. The black trace, also referred to as the Hybrid A trace, however represents the dynamically adjusted trajectory that the vehicle traveled upon considering the obstacles in the path that have been sensed by the sensors and map data [14]. It is worthy to notice that the optimum trajectory can be represented by multiple small arcs. The arc representation of the vehicle's path of travel is later used in the development of the control algorithm for this project.

Rapidly Adapting Lateral Position Handler (RALPH)

Ralph is a technique that employs an advanced control algorithm that allows for the vehicle to maintain its position in a driving lane. Ralph achieves autonomous vehicle steering in three steps. First it samples the image captured by a camera. Second, using

the sampled images it determines the road curvature and finally assesses the lateral offset of the vehicle relative to the lane center [15]. An example of the sampled image is shown in figure 2.5 below. The results of the latter two steps is then combined by Ralph into a steering command, which is then sent to the steering motor to achieve autonomous steering, hence autonomous driving. This control algorithm was implemented on a test bed vehicle called Navlab 5 for autonomous steering control [15].

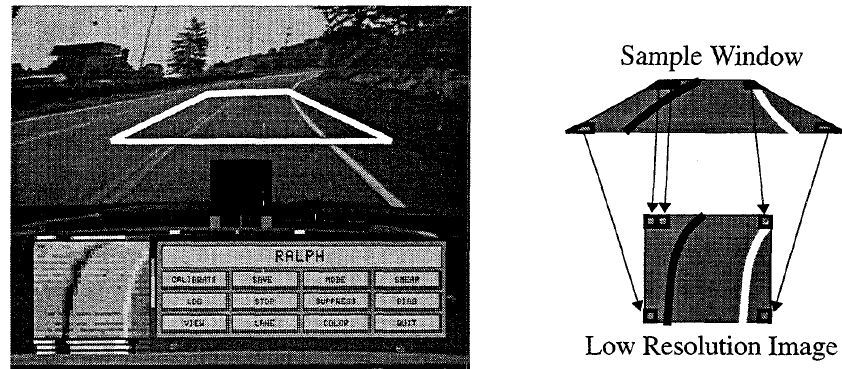


Figure 2.5: Image from the front camera (left), image sampled by Ralph (right) [15]

From the image obtained, the background is irrelevant and only the region within the white trapezoid, as shown in figure 2.5(left), is of interest and the length of the upper and lower boundaries of the trapezoid vary with speed [15]. This section of the sampled image is then parallelized as shown in figure 2.5(right) and analyzed to understand the curvature of the road. The analysis consists of a "hypothesize and test" strategy [15]. RALPH hypothesizes a possible curvature for the road ahead. This hypothesized curvature is then subtracted from the parallelized low resolution image of the road ahead and tested to see how well the hypothesized curvature has straightened the image [15]. This process is repeated multiple times until the best result is obtained and the best radius of curvature is chosen. A visual representation of the determination of radius of curvature is shown in figure 2.6 below.

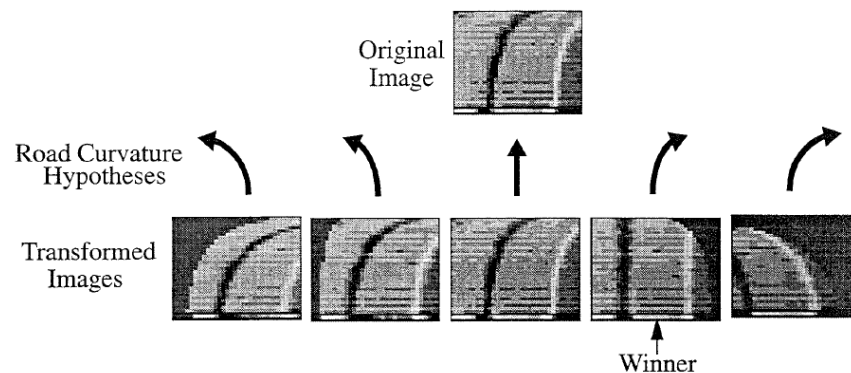


Figure 2.6: Ralph's technique for determining road curvature [15]

2.2.2 Autonomous Control System for Wind Tunnel Model

An autonomous control system for a wind tunnel model was researched by Nicholas Todesco at Macquarie University in 2014 [21]. However the project needed additional work in terms of the development of a control algorithm that used mathematical representation of vehicle steering dynamics. The control system that was implemented was not fully autonomous and also lacked stability in terms of performance. The vehicle was not able to maintain its position with the desired accuracy. The control system was implemented on a 1/10th scale model. It used the micro-controller, Arduino Uno as the central processing unit. An arduino is great in terms of affordability and ease of use. It used a Hitec HS422 servo to steer the vehicle and a Leopard V2 electronic speed controller (ESC) in conjunction with a brushless DC motor to drive the vehicle. For sensors, it used multiple proximity sensors. It used lithium polymer (LiPo) batteries to power the arduino and the ESC.

The working principle of the control system was such that it uses the proximity sensors on both sides of the vehicle to sense the distance of the vehicle from the side walls and steered the vehicle so that the distance measured on either side of the vehicle is the same. Hence maintaining the vehicle's position on the center of the moving ground inside the wind tunnel. It also used proximity sensors at the back of the vehicle to measure the distance of the vehicle from a panel at the end of the moving ground and using this measured distance it controlled the speed of the vehicle to maintain a specific distance. Although this technique was implemented but the operation of the control system was not fully autonomous and needed operator intervention. Also an important aspect of the research that was adopted into this project was the 'toe-in' setting of the steering wheels. Figure 2.7 below shows a 'toe-in' setting for steering wheels.

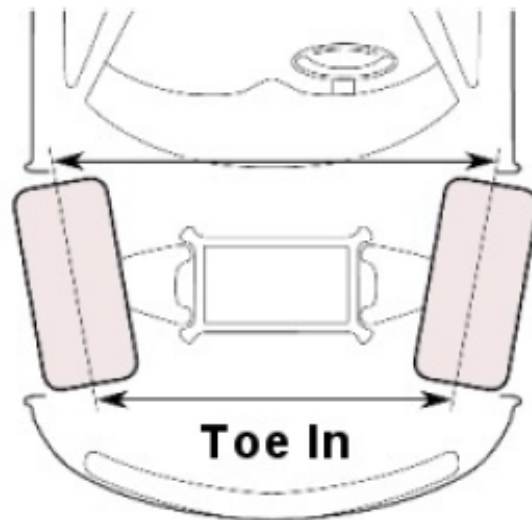


Figure 2.7: Toe-in setting of the steering wheels [21]

The toe-in setting increases the stability of the system when the vehicle is steering. This setting makes the turn less sharp thus contributing to the stability of the control system [21]. This project took prior research and the model vehicle and components used in Todesco's [21] thesis and advanced further into the development of the control system for this project. Since an arduino was being used as the micro-controller, therefore arduino compatible sensors that can be used in this project are also reviewed.

2.3 Sensor Review

This section of the document reviews sensors that can be used to detect the presence of physical objects and measure it's distance from the sensor. In reviewing these sensors, the working principal of these sensors is also reviewed.

2.3.1 Arduino Compatible Sensors

- Proximity sensors : Proximity sensors use infrared rays to sense object. Typically these sensors have an infrared ray emitter and a receiver. The emitted ray reflects of an object present in front of it within range of sensing and the reflection is received by the receiver. The intensity of the reflection received is used to determine the distance of the object from the sensor. Infrared proximity sensors are usually not expensive and their range of sensing is roughly from 10cm to 80cm. However the accuracy of these sensors are not great. It has a tolerance rating that ranges from 20% to 40% [19].
- Ultrasonic distance sensors : These distance sensors use ultrasound to measure distance. These sensors usually contain an ultrasound emitter and a receiver. A short pulse of ultrasound is emitted and the time taken for the emitted wave to reflect form an object within range of detection and come back to the receiver is measured. Using the known speed of travel of the wave, the distance of the object from the sensor is calculated. The arduino compatible ultrasonic sensors that are readily available generally have a sensing range of a few centimeters to three or four meters and the measurements can be as accurate to 3mm [5].
- Laser range finders : Laser range finders are generally more accurate amongst the three types of distance sensors reviewed in this section. Laser range finders use laser light pulse emitter and optical sensors arranged in a compact sensor. The working principle is similar to ultrasonic sensors but different in terms of the wave emitted. It emits a pulse of light and measures the time taken for the emitted pulse to reflect of a surface and come back to the sensor. Laser range finders work in conjunction to optical sensors to sense the reflected light pulse. With the time of flight measured and speed of travel of light known the distance is calculated. Laser range finders have variable range of sensing. Some laser range finders can sense

upto 40m whereas other can sense only upto several meters. The higher the range of sensing the accuracy reduces and vice versa [16].

2.4 Summary

In summary, the evolution of wind tunnel testing facility and how the scale model testing principle is valid was reviewed. Then existing wind tunnel testing facilities was also reviewed and this revealed the areas of testing where there can be improvements hence the motivation for this project is highlighted. Since the implementation of this project will essentially result in a self driving car, therefore control algorithms for self driving cars from existing research were reviewed and the working principle of these algorithms was studied so that it can be related to this project and it was found that the trajectory planning in order to steer the vehicle involved formation of arcs to represent the path of travel. Finally, an autonomous control system for wind tunnel model vehicle was reviewed. In reviewing the control system, the sensors used in the implementation of such a control system were also reviewed along with other potential sensors.

Chapter 3

Development of an Advanced Control System Algorithm

3.1 Introduction

The most crucial part of this project is the final control algorithm that will be used in the autonomous control system. In order to develop this control algorithm, vehicle steering dynamics was analyzed and an approach to realize these dynamics in terms of a single mathematical equation was undertaken. The control algorithm needs to be such that it will be able to align the model vehicle with a target line which will be in the center of the moving ground, the exact specification can be found in chapter 1. When it is aligned with the target line and operating within the desired specifications, it needs to be able to take power consumption measurements and transmit the data to the operator for analysis of aerodynamic drag. Incorporating the results of the steering dynamics analysis, it was observed that the control algorithm would best perform if there were to be two modes of operation, one when the vehicle is not operating within the acceptable tolerance range, (the normal mode) and the other when the vehicle is operating within the tolerance range, (tolerance mode). This section of the thesis discusses the steering dynamic analysis, mathematical approach to finding a method of steering the vehicle and the two modes of operation i.e. the normal mode and the tolerance mode.

3.2 Vehicle Steering Dynamics

In order to steer a vehicle, the control system needs to know that the path the it needs to steer the vehicle along and from section 2.2.1 and 2.2.1 above, it was observed that vehicle trajectory can be represented by a combination of arcs of circles. The arc characteristics can be determined by analyzing the circle.

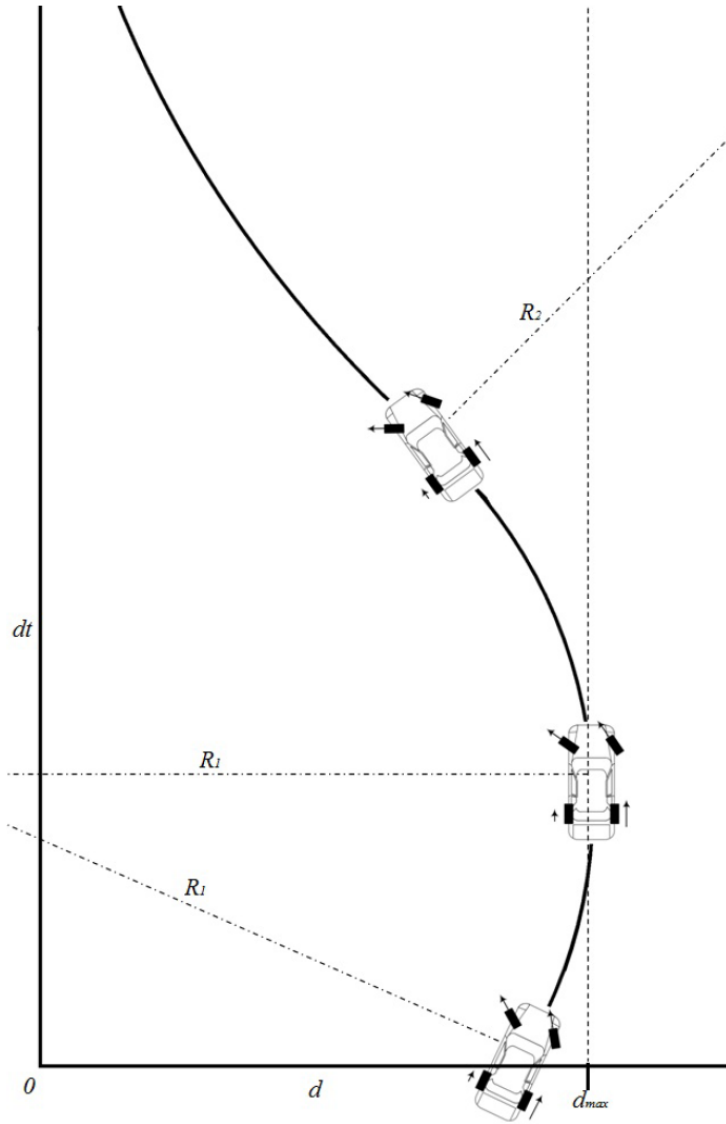


Figure 3.1: Vehicle steering dynamics [21]

From figure 3.1 above it can be observed that the arc can be determined by determining the radius of the circle that is formed by the arc. Initially this radius is labeled R_1 and when the direction of the vehicle has changed by a certain amount of degrees, the center of the circle changes and there is a new radius R_2 which steers the vehicle accordingly. From this analysis it was observed that in order to determine the arc, it will be essential to determine the position of the center of the imaginary circle and the radius of the circle. Therefore further analysis was conducted to form a mathematical equation to represent this arc.

3.3 Mathematical Approach to Realizing Steering Dynamics

In determining how the arc should be, the control algorithm essentially determines a radius of curvature that can be used to represent the arc. These autonomous vehicles operate at velocities often much lower than the velocities that a wind tunnel model vehicle might be subjected to. Operation at lower velocities mean availability of more time to analyze the mass amount of data that can be gathered. However, if a wind tunnel model which will be operating at much higher velocities, is to perform autonomous driving inside the tunnel the amount of data that needs to be sensed has to be minimized as much as possible in comparison to traditional autonomous driving vehicles. The smaller the amount of data that needs to be sensed, the quicker it can be sensed. The faster the input data is sensed, it can be analyzed to output a radius of curvature that the control system can then use to steer the vehicle at high velocities. This suggests that the input parameters to the mathematical equation should be as less as possible. A systematic analysis of the vehicles position on the moving ground revealed that any time during the operation of the normal mode, there can be four different types of scenarios. The scenarios are as follows.

1. The vehicle is to right of the target line, pointing towards the target line.
2. The vehicle is to left of the target line, pointing towards the target line.
3. The vehicle is to right of the target line, pointing away from the target line.
4. The vehicle is to left of the target line, pointing away from the target line.

These scenarios are best illustrated with the aid of a diagram given below.

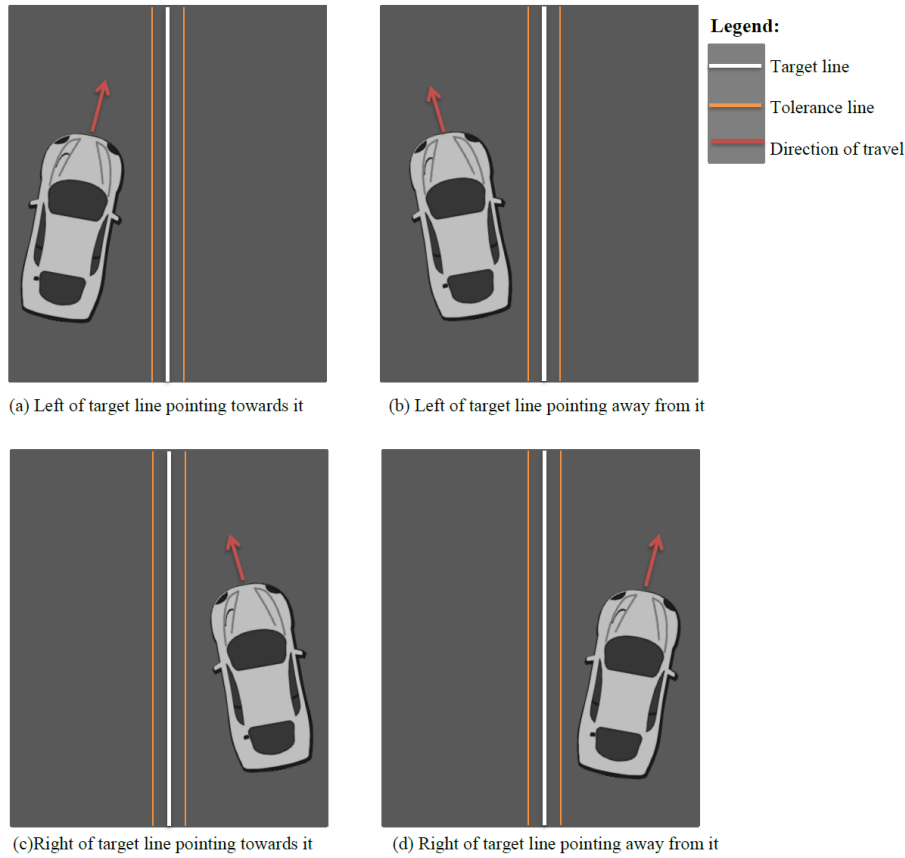


Figure 3.2: Four different scenarios of vehicle location and orientation

Apart from the four scenarios there is one other scenario that can occur. However, this fifth scenario can be incorporated within the four scenarios by making an assumption. For this fifth scenario, the vehicle could be on either side of the target line and it is neither pointing towards it or away from it but traveling parallel to it. In such a case an assumption was made that the vehicle is pointing towards the target line with a very small angle of deviation from the parallel. This assumption puts the fifth scenario within scenario one and two mentioned above. Further analyzing the vehicles position and orientation from the four scenario types, it can be observed that they can be distinguished by the two parameters mentioned below.

1. The distance from the target line.
2. The angle of deviation from the aimed direction of travel.

It was realized that these two parameters can be used as variables in a mathematical equation to calculate a radius of curvature that would represent the arced path corresponding to the vehicle steering dynamic. The mathematical illustration of calculation of the radius of curvature is shown in figure 3.3 below followed by the calculations.

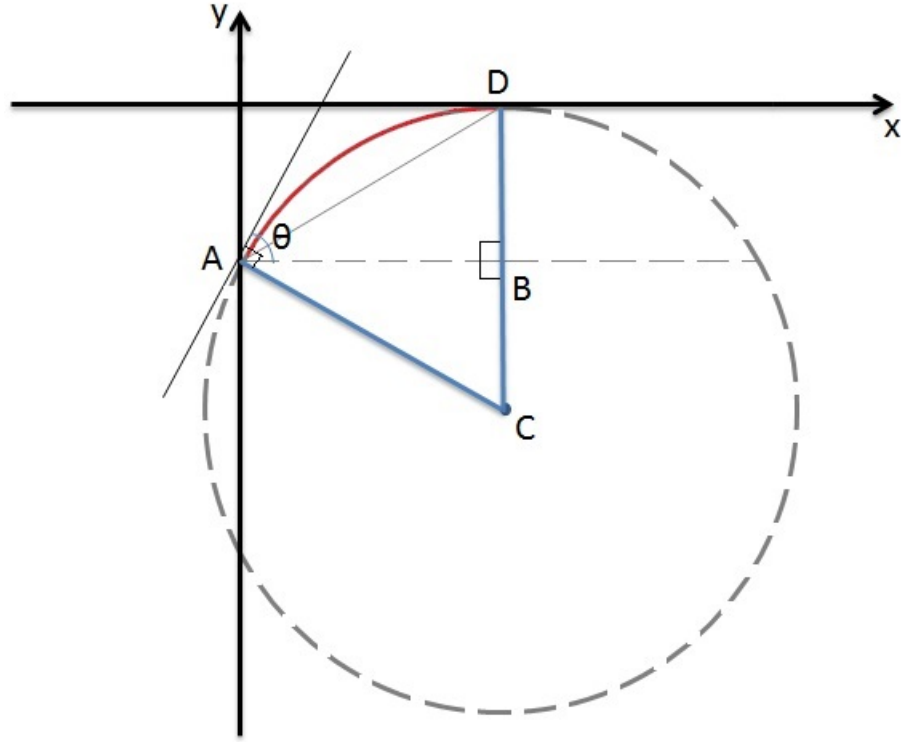


Figure 3.3: Radius of Curvature

In figure 3.3, the broken grey circle represents an imaginary circle and the red arc represents the path of the vehicle. The x axis is the target line that the model vehicle has to travel along with a tolerance of $\pm 5mm$. The inputs to the control system algorithm are two parameters

1. The horizontal displacement of the vehicle from the central target line, ' h '.
2. The angle of deviation from the aimed direction of travel, ' θ '.

Taking these inputs and applying the rules from circle theorem geometry and simple trigonometry, the coordinates of the center of the circle can be worked out and using that the radius of curvature of the circle can be worked out.

Referring to figure 3.3, it was considered that the vehicle is at location A. If a tangent to the circle at A is to be drawn, the angle subtended by the radius and the tangent at A would be a right angle. Therefore:

$$\angle BAC = 90 - \theta \quad (3.1)$$

The angle subtended by the radius and the x axis is also 90° and since the line AB is parallel to the x axis, $\angle ABC = 90^\circ$. As the sum of angles in a triangle equal 180° , and $\angle ABC = 90^\circ$ This makes $\angle ACB = \theta$. Now from $\triangle ADC$, it can be seen that it is an isosceles triangle as line AC and DC are radii. This is shown in figure 3.4 below.

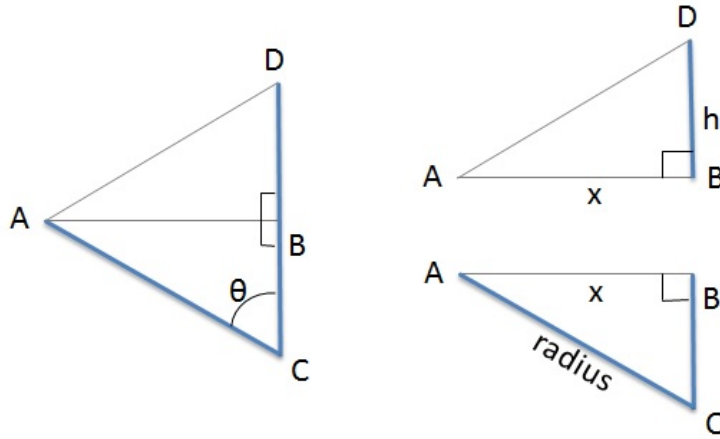


Figure 3.4: Major triangle ADC and sub-triangle ABD and ABC

Using the properties of an isosceles triangle,

$$\angle ADC = \frac{180 - \theta}{2} \quad (3.2)$$

$$\angle ADB = \angle ADC \quad (3.3)$$

Therefore,

$$\tan(\angle ADB) = \frac{x}{h} \quad (3.4)$$

where x is the x coordinate of the imaginary circle and h is the horizontal displacement from the target line

The x -coordinate of the center of the circle is given by:

$$x = h \tan\left(\frac{180 - \theta}{2}\right) \quad (3.5)$$

Therefore looking at $\triangle ABC$ in figure 3.4, the radius, which is in fact length AC is given by:

$$radius = \frac{x}{\sin(\theta)} \quad (3.6)$$

Combining equations 3.5 and 3.6 a single equation that takes the displacement and angle as its input and returns the required radius of curvature was obtained. This equation is given below.

$$radius = \frac{h \tan\left(\frac{180 - \theta}{2}\right)}{\sin(\theta)} \quad (3.7)$$

Once the equation was obtained, it was used in a simulation program to understand if the equation was able to realize the required trajectory that a vehicle would steer by.

3.3.1 Simulation of Steering Dynamics

The simulation was conducted for different combinations of horizontal displacement and angle of deviation to realize the four different types of scenarios. The results of the simulation are shown below in figure 3.5.

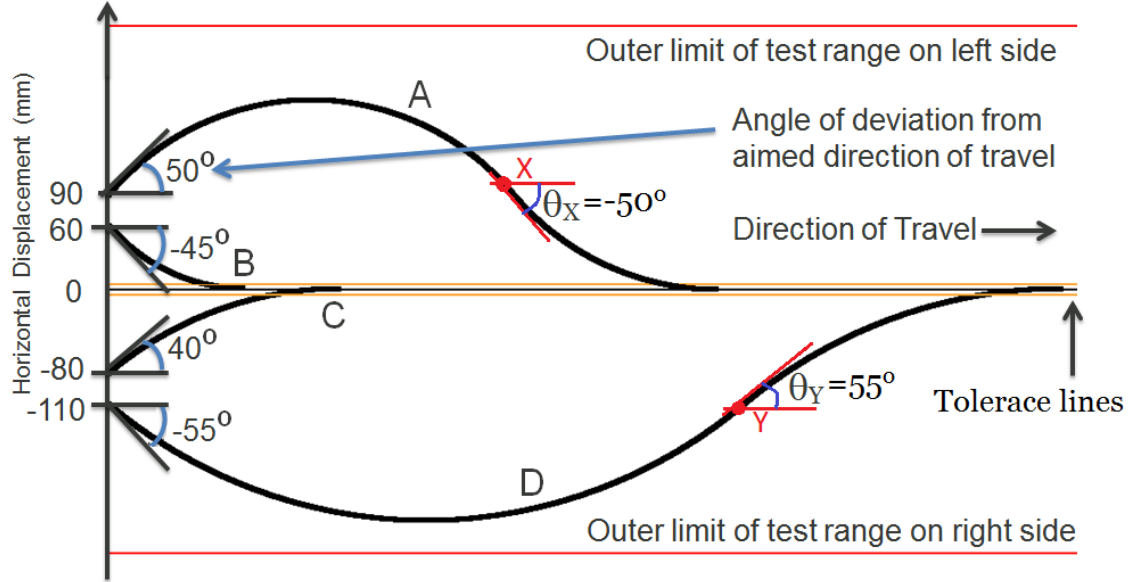


Figure 3.5: Steering Dynamic Simulation

In figure 3.5 above there are four different trajectories illustrated each corresponding to the four scenarios mentioned earlier in this section. However, in the calculation above only the case where the vehicle is pointing towards the target line was used to obtain the equation for radius of curvature i.e. equation 3.7. In the simulation, in order to project the vehicle's trajectory in the case where the vehicle is pointing away from the target line a recursive method of using equation 3.7 was adopted. In this case, the simulation program uses equation 3.7 to compute a radius of curvature that will smoothly steer the vehicle in an arc until the vehicle is pointing towards the target line but with a horizontal displacement. At this point, equation 3.7 is again used, in a recursive manner, to compute a new radius of curvature according to the calculations shown above. This is illustrated in figure 3.5 by the trajectory marked 'A' and 'D' and the point on the trajectory where the recursive use of equation 3.7 occurs is marked by 'X' and 'Y' respectively.

In the figure the orange lines represent the tolerance region. It can be noticed that the equation is able to determine a radius of curvature that can realize the trajectory needed for the model vehicle to align itself within acceptable tolerance. This technique of calculating a radius of curvature and using it to steer the vehicle into alignment is essentially applicable to the normal mode of operation where the vehicle is not within acceptable tolerance as once the vehicle is inside the tolerance region it simply needs to maintain the position by correcting any locational drift in any direction. Therefore, for

the tolerance mode, a more sensitive control algorithm was developed. The two modes of operation are further discussed in the following sections.

3.4 The Normal Mode

When the vehicle is operating in the normal mode, the autonomous control system needs to control the vehicle to bring it within the tolerance region very steadily due to the high operational velocities of the wind tunnel. Attempting to steer the vehicle rapidly will compromise the stability of the system. Therefore, to avoid the risk of losing stability, the four different types of scenarios as shown in figure 3.2 was further analyzed. Out of these four types of scenarios, they can further be classified as two different types in order to make the control algorithm for normal mode of operation more optimized. The two types are as follows.

1. The vehicle is pointing towards the target line.
2. The vehicle is pointing away from the target line.

The two scenarios mentioned above will have slightly different control algorithms applied to them in order to achieve smooth autonomous driving. In the case of scenario one mentioned above in this section, relating the simulation to the actual scenario is straight forward. On the other hand, in case of scenario two when the vehicle is pointing away from the target line, it was decided that the algorithm uses a fixed radius to steer the vehicle and make it point towards the target line and then apply the control algorithm for scenario one. This is best illustrated with the aid of a flowchart in terms of block diagrams. This is given in figure 3.6 below.

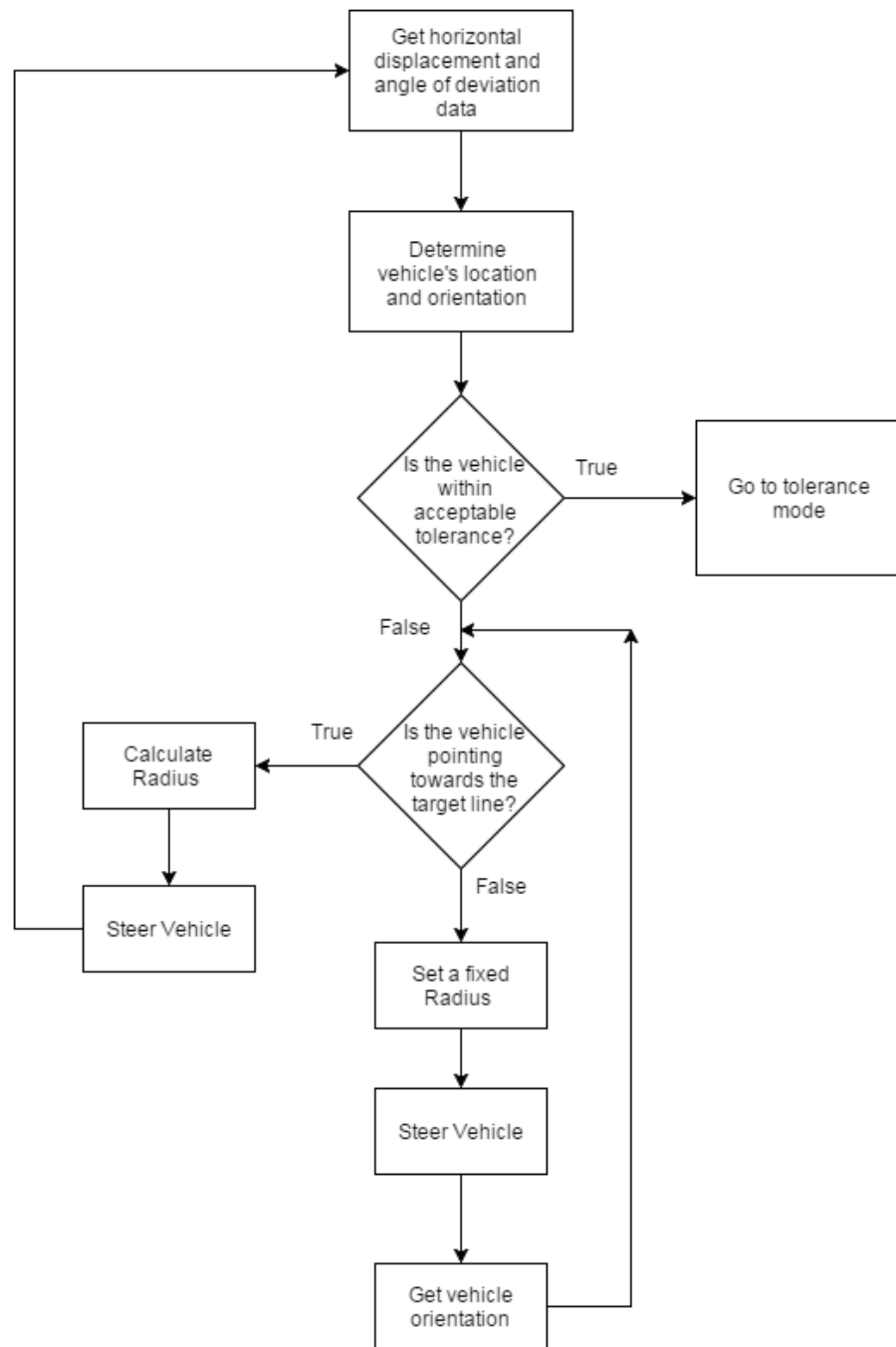


Figure 3.6: Control algorithm for the normal mode of operation

The fixed radius mentioned in the control algorithm was set to be 200mm. This was set to be 200mm because, as per the specification of the wind tunnel test facility available in Macquarie university, the test section is 500mm wide. With the target line at the center of this test section, this leaves a horizontal range of 250mm on either side. Therefore the fixed radius was capped at 200mm as this would ensure that the vehicle does not steer

outside the bounds of the test section. It is worth noticing that in the case where the vehicle is not pointing towards the target line, the orientation of the vehicle is updated constantly to check for the instance when it is pointing towards the target line. In the complete execution of the algorithm, the data regarding vehicle's location and orientation is updated and checked in every iteration to determine whether the vehicle has entered the tolerance mode or not.

3.5 The Tolerance mode

The second mode of operation is the tolerance mode. In this mode the vehicle is already within acceptable tolerance and now it needs to maintain the vehicle's position with a high degree of accuracy. According to the project specification the control system needs to hold the vehicle in position accurate to $\pm 5mm$. The first step within the operation of the tolerance mode is to check whether the vehicle has lost its position from the tolerance region or not in that case the algorithm autonomously shifts to the normal mode. If the vehicle is still within the tolerance mode, a method needed to be devised that can realize any small incremental change in the position of the car in any direction. Upon detection of a movement of approximately 5mm, the control algorithm needs to steer or adjust throttle or do both according to the movement of the vehicle.

The response of the control system according to the possible changes in position of the vehicle is best described by tabular representation. This is shown below in table 3.1

5mm change in vehicle's position				Corresponding required action
forward	backward	leftward	rightward	
True	X	X	X	Reduce throttle
X	True	X	X	Increase throttle
X	X	True	X	Short steer right
X	X	X	True	Short steer left
True	X	True	X	Reduce throttle and short steer right
True	X	X	True	Reduce throttle and short steer left
X	True	True	X	Increase throttle and short steer right
X	True	X	True	Increase throttle and short steer left
X	X	X	X	Hold steering value and throttle value

Table 3.1: Control system response for small change in vehicle's position

In table 3.1 above, the first four columns represent the directions in which the vehicle might move and the fifth column represents the required action of the control system. A tabular approach was taken to represent this as using a table it would be easier to realize the possible combinations of change in vehicle's position that might occur and how they can be related to the required action of the control system. From the table it can be seen that there are nine possible combinations and the required action of the

control algorithm outcomes of these nine possible cases are outlined. When the vehicle is performing within the tolerance region and is stable, the control system needs to measure the power consumption of the vehicle and transfer this information to the operator for analysis of aerodynamic drag. This flow of operation is again best described with the aid of a flow chart as shown in figure 3.7 below.

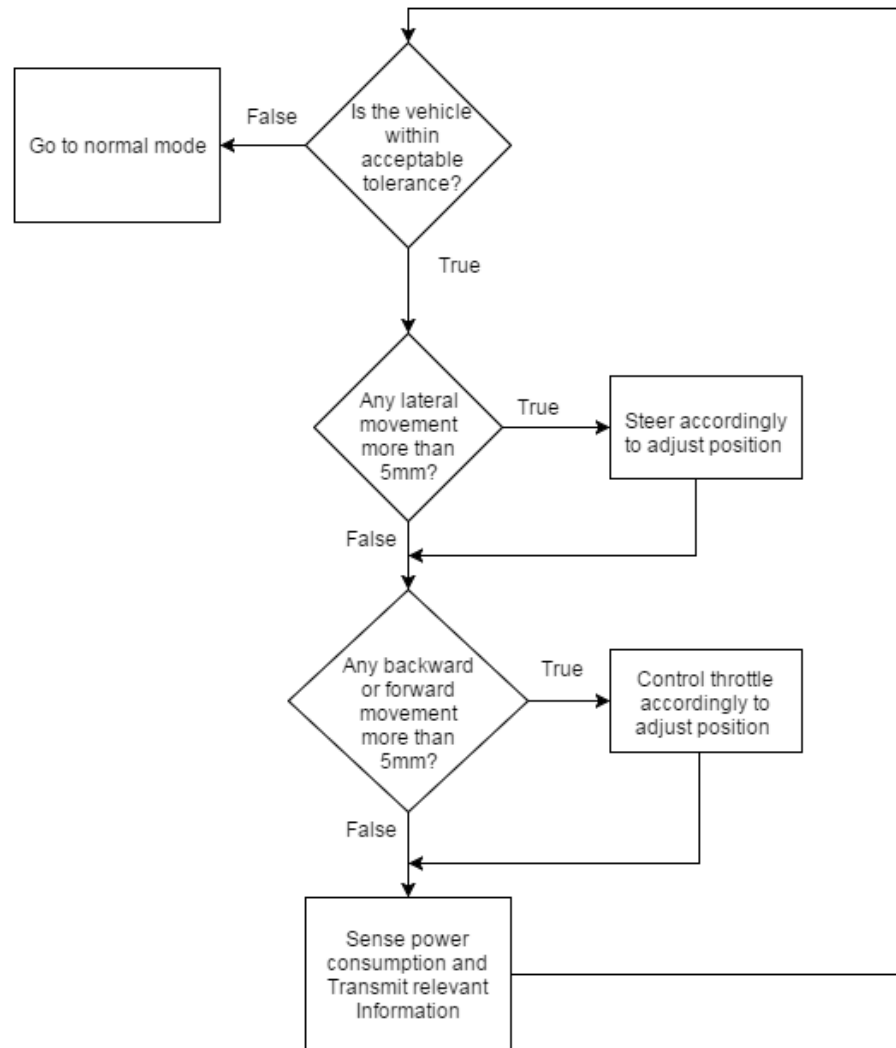


Figure 3.7: Control algorithm for the tolerance mode of operation

Now that the control algorithm for both the modes of operation has been determined they can be connected to obtain a complete final algorithm that represents the total operational flow of the autonomous control system.

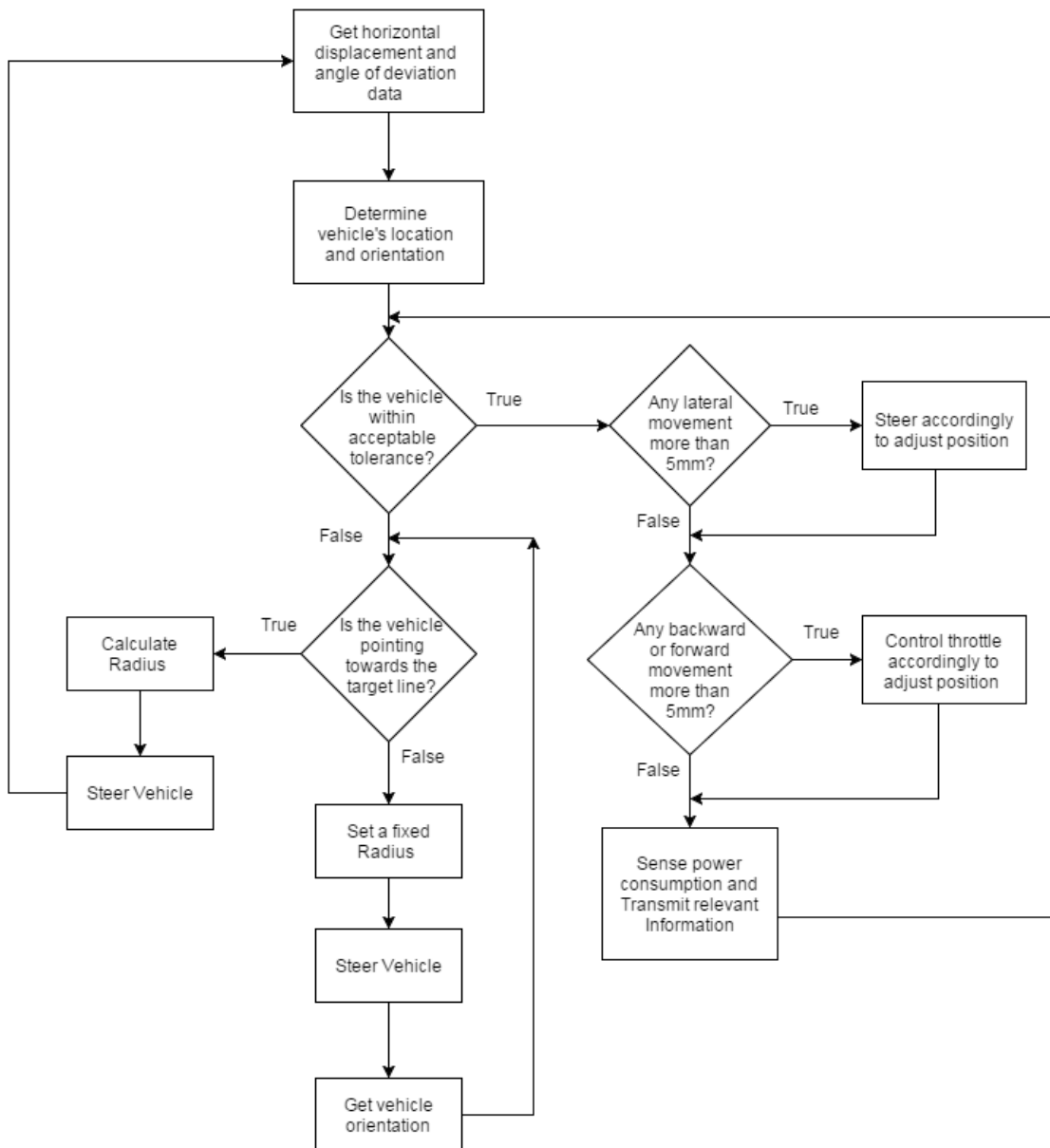


Figure 3.8: Complete Control algorithm incorporating the two modes of operation

Figure 3.8 above represents the complete control algorithm. It is worth noticing that the control block that checks if the vehicle is in tolerance or not is accessed in every iteration of the control algorithm. This ensures that the control algorithm always operates in the right mode of operation. Another important aspect worth mentioning is that, during the implementation of the control algorithm, the control algorithm was slightly altered. While the control principle remains the exact same, few additional control blocks were added such that the sensors that can be used to realize the two modes of operation are incorporated. This is further discussed in chapter 4.

3.6 Determination of the Drag Force

One of the aim of this project is such that the control system has to be capable of measuring the drag fore that is experienced by the vehicle. This section discusses the technique of how the control system was given drag measurement capability.

In fluid dynamics, drag sometimes called air resistance or fluid resistance, is a force that acts opposite to the relative motion of any object moving with respect to the surrounding fluid. The drag experienced is proportional to the velocity of the moving object. In other words the power produced by the vehicle to maintain its velocity is proportional to the drag. By measuring the power consumption of the vehicle the drag that the vehicle is experiencing can be determined. A mathematical analysis to calculate the actual drag excluding other resistive forces involved is given below.

During the operation of the vehicle inside the wind tunnel, it will be subjected to various forces Figure 3.9 below shows all the forces acting on it.

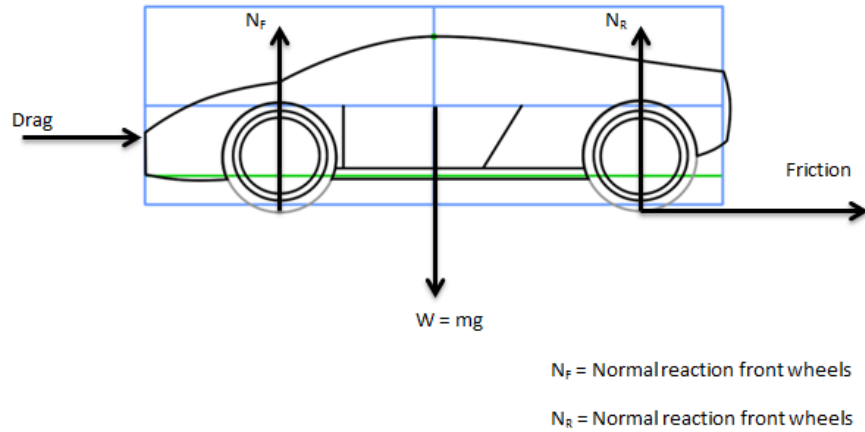


Figure 3.9: Different forces acting on the vehicle during operation

This total resistive force (Drag + Friction) now needs to be overcome by the power produced by the vehicle. The power produced by the vehicle is the work done per unit time and work done is given by:

$$work = Force \times distance \quad (3.8)$$

Therefore,

$$power = \frac{Force \times distance}{time} \quad (3.9)$$

and distance moved per unit time is the velocity and in this case the force is the total resistive force. Therefore equation 3.9 can now be written as

$$P = (D + Fr) \times v \quad (3.10)$$

where P is the power, v is the velocity, D is the drag and Fr is the friction experience by the vehicle.

From equation 3.10, ' D ' is the desirable variable and ' Fr ' needs to be calculated separately in order to obtain the actual drag. The friction can be calculated using the relation between normal reaction force and friction.

$$Fr = \mu \times N \quad (3.11)$$

where N is the normal reaction force and

$$N = mg \quad (3.12)$$

where ' m ' is the mass of the vehicle and ' g ' is the acceleration due to gravity. Combining equations 3.11 and 3.12 and substituting it into equation 3.10, the following equation can be obtained.

$$P = (D + \mu \times mg) \times v \quad (3.13)$$

The power consumed by the vehicle in order to maintain its position can be measured by taking appropriate voltage and current measurements. The mass of the vehicle can be easily measured and the velocity will be known from the velocity of the moving ground. ' g ' is a constant and ' μ ' will vary from surface to surface but for a known surface it will be a constant. Therefore, rearranging equation 3.13 to make ' D ' the subject, the drag can be calculated or if a graph of P vs v were to be plotted, the constant ' μmg ' subtracted from the gradient would also give the drag.

3.7 Chapter Summary

In summary, this chapter looks into the development of the advanced control system algorithm. In this chapter, vehicle steering dynamics was analyzed and a mathematical approach to realize the steering dynamics was undertaken. The different types of scenarios that the model vehicle might be subjected to during its operation was speculated and the distinguishing parameters were identified. These parameters were found to be the horizontal displacement of the vehicle relative to the target line and the angle of deviation of the vehicle from a line parallel to the target line. Using these two input parameters, circle theorem geometry and trigonometry was applied to obtain a single mathematical equation that outputs a radius of curvature which can be used to steer the vehicle. The equation was simulated to observe whether it projected the correct steering trajectory or not. Analysis of the simulations revealed that for best results, there needed to be two modes of operation, the normal mode and the tolerance mode. Later in the chapter, the normal mode and tolerance mode was analyzed and control algorithms for each of these modes were developed. The control algorithms for the two modes were combined to obtain a final complete control algorithm. Finally, a mathematical equation to measure the drag

experienced by the vehicle was developed and the technique to calculate the drag from the equation was discussed.

Chapter 4

Specific Criteria Based Sensor Analysis

4.1 Introduction

This chapter in the thesis looks into the sensors that were used to sense the necessary data that can be used as the input parameters for the control algorithm for both the modes of operation. There are several different combinations of sensors that can be used to achieve this, however there were several criteria involved in the choosing of sensors which limited the available options. These criteria used in the decision making process along with the final choice of sensors, are discussed below. Once the sensors of choice were procured they were tested in order to check their performance and to see if they were accurate enough for the purposes of this project. The methods used in conducting these tests and the test results are also discussed in this chapter.

4.2 Sensor Research Criteria

In order to establish the criteria for choosing sensors, it is necessary to understand what is it that needs to be sensed. From section 3.3 and section 3.4 above, it can be determined that in the normal mode of operation there are two parameters that needs to be sensed. They are the horizontal displacement from target line and angle of deviation from aimed direction of travel. So essentially the control system needs to take distance measurements and orientation measurements. When taking distance measurements, the range of sensing is an important aspect worth considering. The control system is to perform with a certain degree of accuracy. The control system also needs to operate at high speeds which indicate high frequency of sensing. Two additional factors that were considered when choosing the sensors were the cost and lead time for procurement as the project needed to be conducted within a set budget and duration. The selection criteria is discussed further below.

- Range: The range of sensing was the first criteria that was considered. The maximum required range of sensing was obtained through analysis of the technique that

will be adopted in implementing the control algorithm. The control algorithm needs the distance of the vehicle from the target line. This was achieved making use of the known dimensions of the wind tunnel test section. The test section is 500mm wide, this makes the target line to be 250mm from either of the side walls. If the vehicle were to sense the distance of one of these walls and subtract it from 250, the magnitude of the resulting figure would be the distance of the vehicle from the target line and the sign of the figure can be used to deduce the vehicle's position on either side of the wall. Considering this technique the maximum distance that needed to be sensed was 400mm as the sensors were to be mounted on the side of the vehicle and the model vehicle used in this project was only 118mm wide.

- Accuracy: Sensors are a key component of this project when it comes to the accuracy of the deliverable. The specified accuracy for this project is $\pm 5mm$. Sensors are the first point of source of error as the data sensed by the sensors will be used to compute the output of the control system and if the input data is erroneous then the computed output will be erroneous as well. Thus resulting in inaccurate position maintenance for the model vehicle. Since the desired accuracy on the control system was $\pm 5mm$, this allows a total variance of 10mm therefore the sensors were desired to be accurate to 10mm or better.
- Frequency: The sensor needed to have a high frequency of operation. Since the model vehicle will be traveling at variable high speeds inside the tunnel, therefore the higher the operating frequency for the sensors the better.
- Cost: The cost had to be as minimum as possible, the allocated fund for the project was three hundred Australian Dollars. The sensors along with other components had to be procured from this fund. Some components cost much more than others, therefore, the cost per item was minimized as much as possible to include everything within the allocated fund.
- Lead time: In researching for the sensors and other components, the lead time was attempted to be kept as short as possible. The stage of the project when the order for parts was placed the maximum amount of time that could be allocated for the arrival of parts was three weeks. This factor was also incorporated in the choice of sensors.

These were the criteria that were considered in the research for sensors. The different sensors that were considered and the ones that were finally selected are discussed in the next section.

4.3 Sensor and Component Selection

According to the criteria mentioned above a research was conducted to find suitable sensors. The results of the research are best described in the form of a table. This is given below in table 4.1.

Name	Range	Accuracy	Frequency	Cost	Lead time
Parallax Ultrasonic Distance Sensor	2cm - 3m	11% - 12 %	5kHz	Approximately AU40.53+ AU10.99 (shipping)	9 days - 26 days
HC-SR04 Ultrasonic Sensor	2cm - 400cm	3mm (0.075% - 15%)	40Hz	AU\$1.90	3 days - 9 days
IR Proximity sensors	10cm - 80cm	20-40%	25Hz	AU\$13.20	<6 days
Parallax Laser Range finder	15cm - 122cm	3% avg 5% max	1Hz	USD 99 + (shipping)	10 days - 3 months
Adafruit 9-DOF Absolute Orientation IMU Fusion BNO055 Breakout Sensor Module		60 mg	100Hz	AU46.51+ AU34.36 (shipping)	9 days - 19 days
Adafruit 9-DOF Accel/Mag/Gyro+ Temp Breakout Board - LSM9DS0 (ADA: 2021)		60mg	100kHz - 400kHz	AU41.95+ AU7.95	<5 days
DC Voltage Detector & Sensor Module For Arduino ADC	DC 0-25V	0.1%	X	AU\$5.75	9 days - 10 days
1 Pc 5A Range Current Sensor Module ACS712 for Arduino	0-5A	1.5%	X	AU1.88+ AU0.26	3 weeks - 5 weeks
30A Range Current Sensor Module ACS712ELC-30A Chip Module (Arduino)	DC 0-30A	1.5%	X	AU\$7.87	<7 days

Table 4.1: Results of criteria based sensor research [19] [5] [8] [7] [22] [20] [10]

From table 4.1 the desired sensors and other required components were chosen. To sense the vehicle's distance from the side wall the HC-SR04 Ultrasonic sensor was chosen as its performance specifications best suited the projects need. To obtain the orientation of the vehicle, the Adafruit 9 Degree of Freedom Absolute Orientation IMU Fusion BNO055 Breakout Sensor Module was chosen. However, due to the unavailability of the item the choice had to be changed to Adafruit 9 Degree of Freedom Accelerometer, Magnetometer, Gyroscope and Temperature Breakout Board. In order to measure the power consumption of the control system a current sensor and a voltage sensor was chosen. To sense the

current a 30A Range Current Sensor Module using ACS712ELC-30A Chip Module was chosen and for the voltage sensor the 0-25VDC voltage detector sensor module for arduino ADC was chosen.

The distance and orientation sensor chosen are to be used in the normal mode of operation. During the tolerance mode of operation, in order to sense very small drift of less than 5mm in the vehicles position, a technique using a combination of multiple components was conceptualized. It was proposed that a light dependent resistor (LDR) array, in conjunction with a laser pointer could be used to detect positional drift of a range less than 5mm. An LDR is a variable resistor whose resistance varies with the amount of light being shone on it. The laser beam shone on the LDR array will focus on one LDR at any time and identifying the LDR that has the strongest intensity of laser light shone upon, the position on the vehicle relative to the laser point can be identified. Holding the laser pointer on top of the target line a reference position for the vehicle can be obtained and using the relative distance of the center of the vehicle from the reference point can be used to steer the vehicle such that it always minimizes the relative distance. The LDR and laser arrangement is shown below with the aid of a diagram.

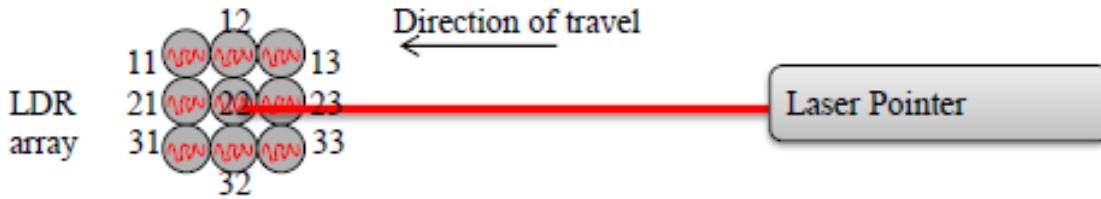


Figure 4.1: Concept of LDRs and laser pointer

According to figure 4.1 above, considering the direction of travel to be leftward, if the laser pointer is shining on the central LDR i.e. LDR 22, the vehicle is traveling along the target line. If the laser points on any of the LDRs numbered 11, 12 or 13 the vehicle has drifted left and needs to steer right. Similarly, laser pointing on any of the LDRs numbered 31, 32 or 33 means the vehicle needs to steer left. If the laser points on LDRs numbered 11, 21 or 31 the vehicle has drifted towards the rear and needs to accelerate. Similarly, if any of the LDRs numbered 13, 23 or 33 are illuminated by the laser, it indicates the vehicle is moving too fast and needs to slow down.

In terms of components the micro-controller that was chosen as the central processing unit of the control system is the Arduino Uno. The number of input and output (I/O) pins on this micro-controller is limited. In order to cater for the proposed LDR array the number of (I/O) pins needed to be increased. Therefore a multiplexer was also incorporated in the list of parts to be procured.

4.4 Sensor Test

The LDR arrangement mentioned in previous section needed to be tested so that it could be confirmed that the arrangement can sense minute changes in vehicles position and act

on it.

4.4.1 LDR Array and Laser Dot Combination

In testing the LDR, a linear translator was made out of the available equipment in the laboratory. A set of LDRs were attached side by side and a red laser dot was shone on one of the LDRs. The linear translator was used to translate the LDR arrangement until the laser was shining onto the second LDR. The change in output due to change in resistance of the LDRs upon shining the laser light was recorded corresponding to the linear translation of the LDRs and a graph was plotted to identify the maximum distance of translation that can occur before the sensors outputs completely change their values. The arrangement of the equipment is shown in figure 4.2, 4.3 and 4.4 below.

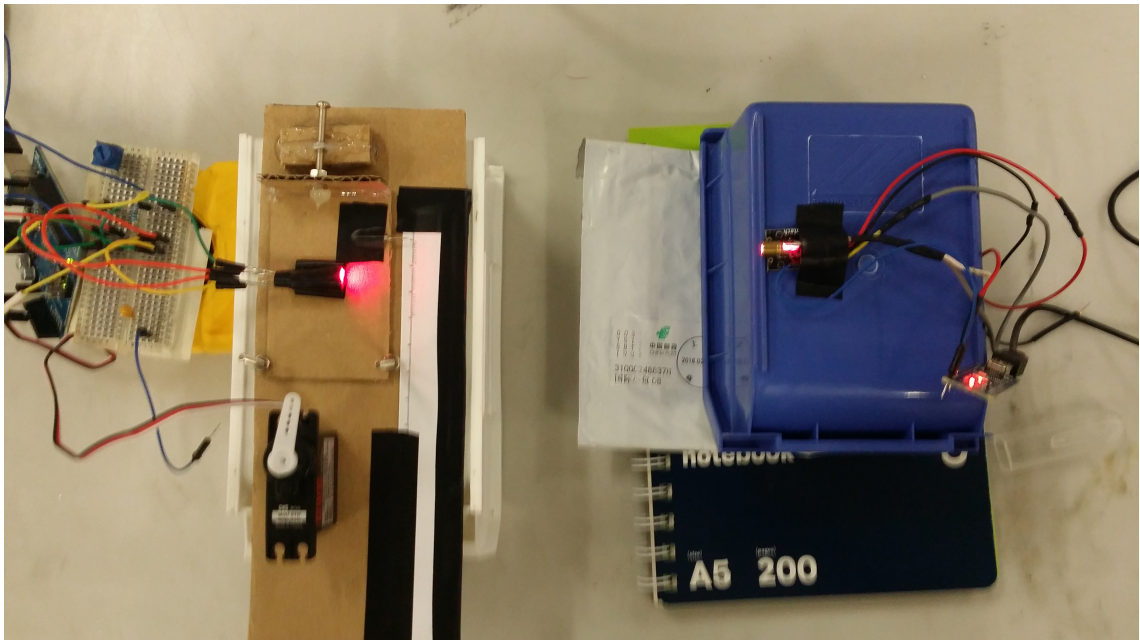


Figure 4.2: Arrangement of LDRs and Laser Dot

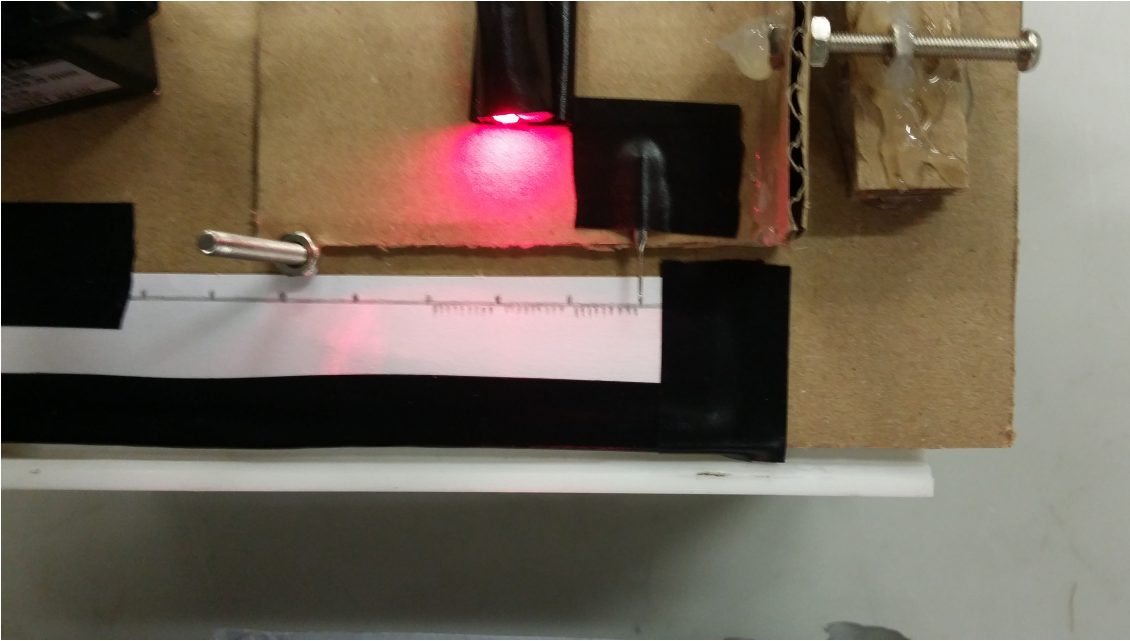


Figure 4.3: Initial Position of LDR

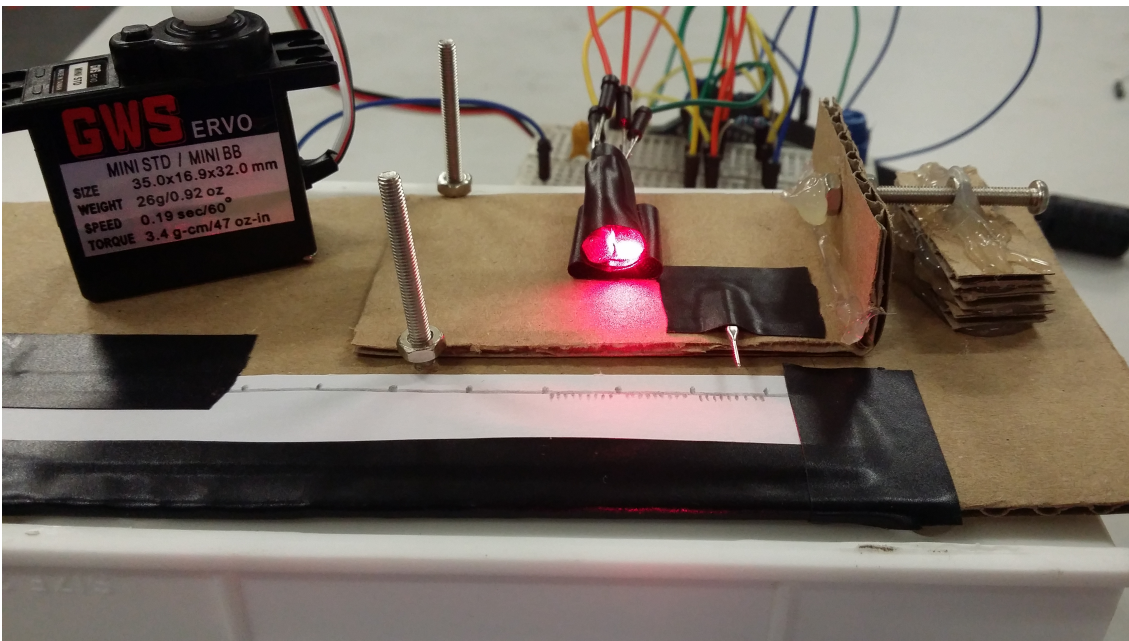


Figure 4.4: Incremented Position of LDR

The graph that was obtained is shown in figure 4.5 below and it is seen that as the position of the LDRs is incremented, one sensor value decreases and the other increases and at an increment of 3mm the value of the second sensor becomes greater than the first sensor. Hence the difference curve changes quadrant on the plot.

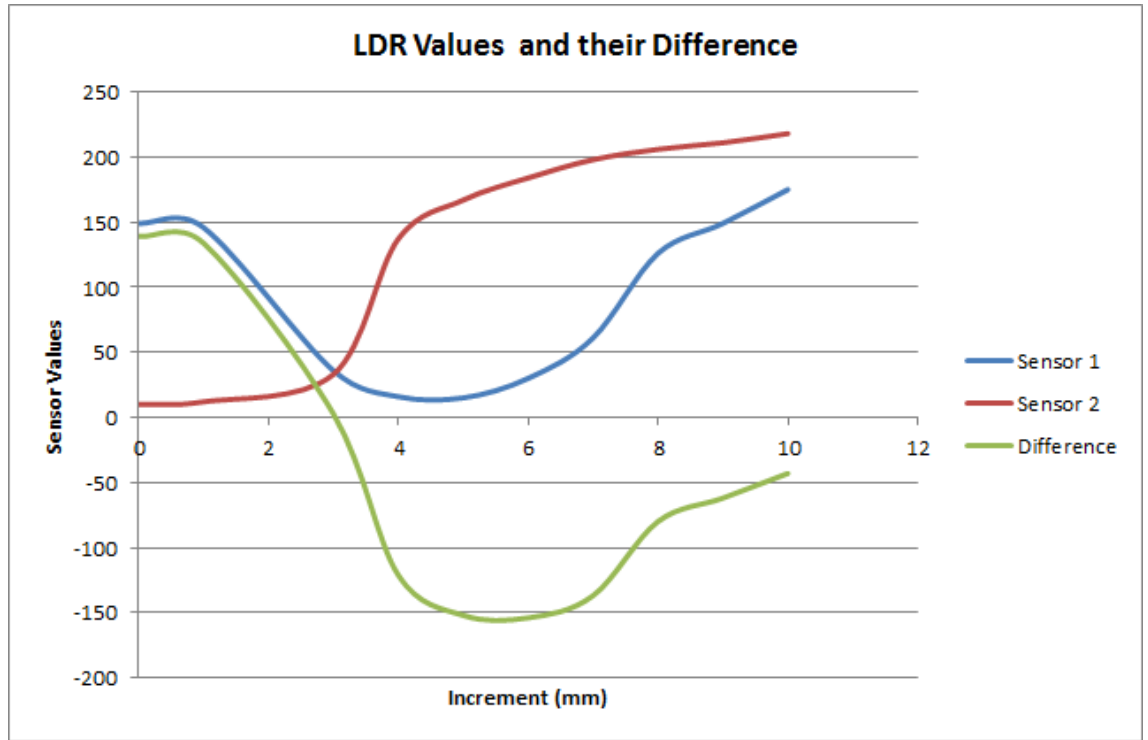


Figure 4.5: LDR values corresponding to their position increment

This confirms that the LDR arrangement can be used to accurately identify a movement of 3mm. Since the specification of the project is such that the vehicle maintains its position with an accuracy of 5mm, a sensor arrangement that can detect a movement accurate to 3mm is desirable.

4.4.2 Ultrasonic Sensor

The ultrasonic sensor was also tested to determine the accuracy of the sensor. The distance to an object from the sensor was measured using both the ultrasonic sensor and a ruler to get the real distance. The distance of the object was varied and recorded. For each distance value a hundred distance reading samples were taken and the average was calculated. Then a graph was plotted to understand the difference in measured values and actual values. The equipment arrangement is shown in figure 4.6 below.

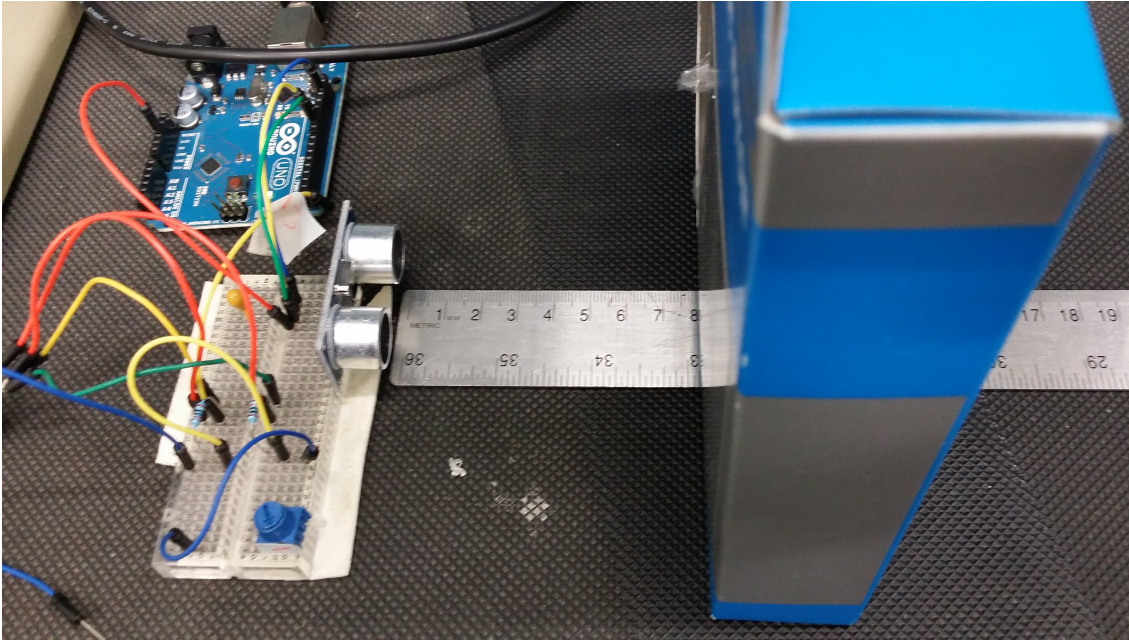


Figure 4.6: Ultrasonic Sensor Test Arrangement

Two graphs were plotted and are shown in figure 4.7 and 4.8 below. It was seen that the percentage error between the actual values and the measured values fluctuated between 0 and 0.14

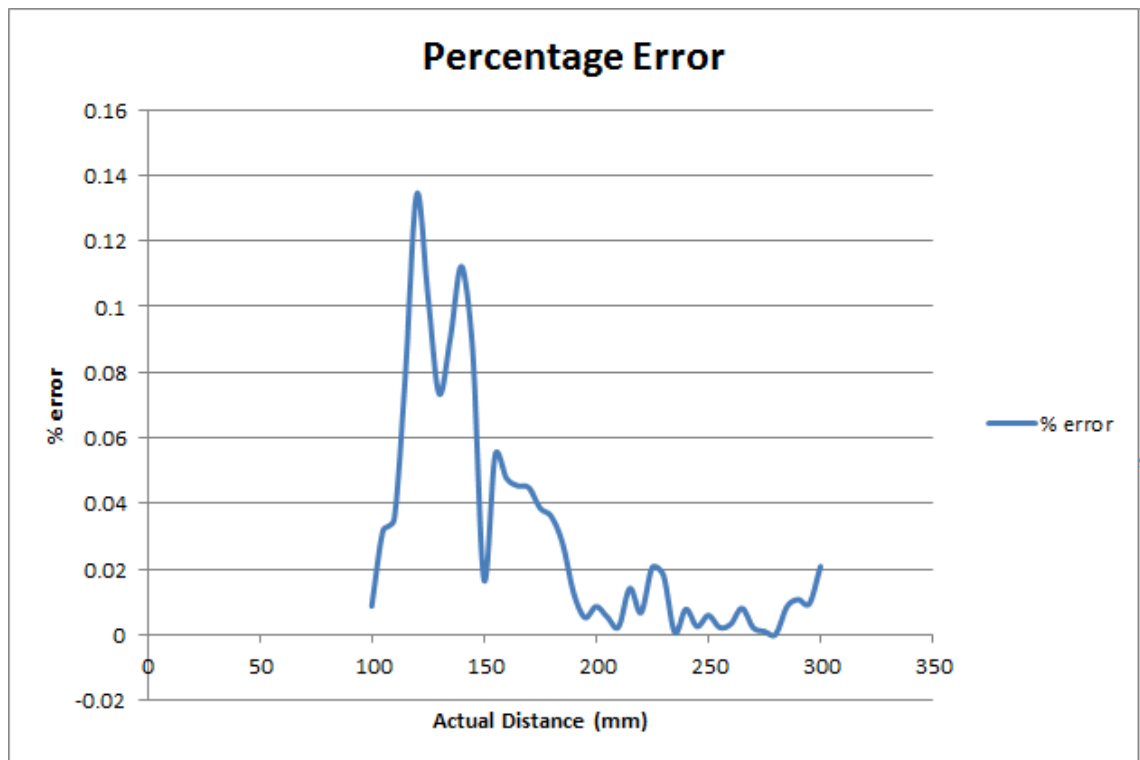


Figure 4.7: Percentage Error in Measurement Using Ultrasound



Figure 4.8: Measured Distance vs Actual Distance

From both the graphs it can be seen that the distance the sensor has an optimum performance range when it is measuring distance greater than 200mm. This is actually suitable for the purposes of this project.

4.5 Summary

To summarize a specific research criteria was formed and sensors were researched according to the criteria. The criteria involved were range of sensing, accuracy of measurement, frequency of operation, cost and lead time. According to the criteria, the best fit sensors were the HC-SR04 ultrasonic distance sensor and LSM9DSO accelerometer, magnetometer and gyroscope. These sensors were applicable for the normal mode of operation and for the tolerance mode of operation, an arrangement of LDRs in conjunction with a laser pointer was developed and tested. The LDR arrangement and ultrasonic sensor was tested for their accuracy of performance and it was found that the ultrasonic sensor had a optimum operating range of more than 200m which is appropriate for this project and the LDR arrangement was accurate to 3mm.

Chapter 5

Implementation of the Control System

5.1 Introduction

The control algorithm developed in section 3.5 and the sensors researched in section 4.3 were implemented on a 1/10th scale model vehicle that was provided from the department. Taking the project further from prior development, the arduino, electronic speed controller and brushless motor was kept. The sensors were replaced and additional sensors were added. The proximity sensors were changed to the combination of sensors researched in section 4.3. In order to mount all the sensors appropriately a sensor mount was designed according to the dimensions of the model vehicle and the sensor mount was 3D printed. The micro-controller from the prior project was reprogrammed such that it would realize the new sensors and control algorithm. This section discusses the design of a sensor mount to house all the sensors and necessary components and the technique that was adopted in order to implement a prototype autonomous control system.

5.2 The Technique of Implementation

The control system will have two modes of operation as discussed earlier and these two modes need different set of sensors. As mentioned in section 4.2, for normal mode of operation, the distance of the vehicle from the target line will be obtained by measuring the distance of the vehicle from the side wall and subtracting 250mm from it. Therefore the ultrasonic sensor chosen in section 4.3 will be mounted on one side of the vehicle and the accelerometer will be mounted on top of the vehicle such that it lays flat. For the tolerance mode of operation, the LDR array will also be mounted on the top of the vehicle so that the laser pointer can point onto the array. This technique of implementing the control algorithm is illustrated below with the aid of a diagram.

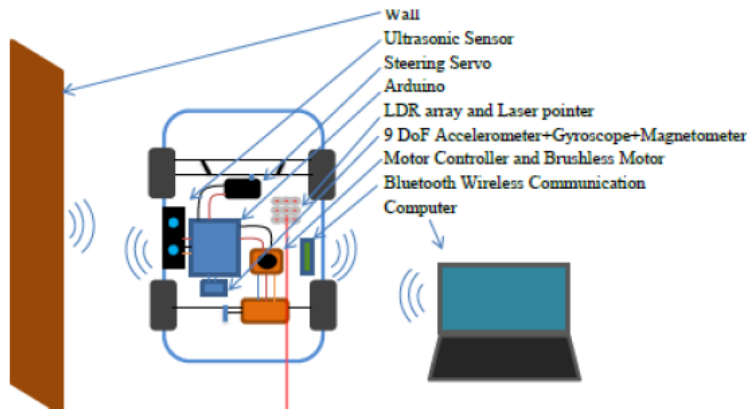


Figure 5.1: The Sensor Setup Plan

Figure 5.1 above shows the plan that was adopted to develop the control system. According to this plan a test bench prototype control system was developed with the sensors mounted accordingly to test the functionality of the plan. This is shown in figure 5.2 below.

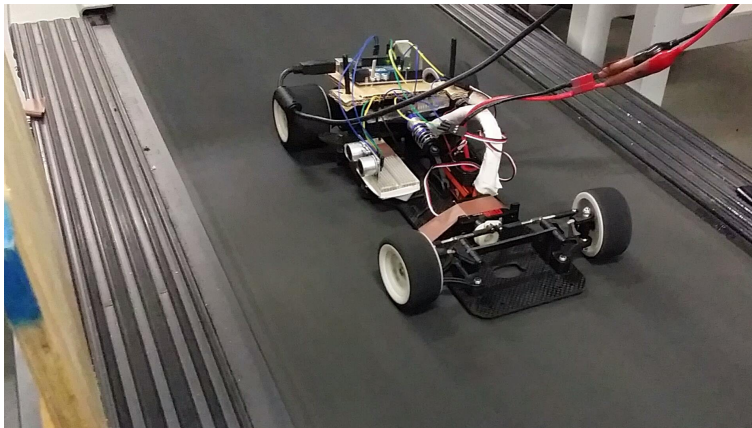


Figure 5.2: The model vehicle maintaining its position on the running ground

It was observed that the control system was being able to measure the vehicle's distance from the side wall and steer the vehicle to maintain the measured distance. However it was not being able to maintain the vehicle position according to the desired specification of $\pm 5mm$ accuracy. Therefore the tolerance mode needed to be implemented to meet the project specification. In order to implement the tolerance mode of the control system, the LDR array needed to be developed. Therefore a schematic was developed which incorporated the LDR array.

An LDR was used as a resistor in a voltage divider and when the laser light is shone onto the LDR, the resistance decrease rapidly which alters the voltage across the voltage divider and this change in voltage can be used to identify which LDR is the laser pointing at. This voltage divider circuit is shown below for a single LDR.

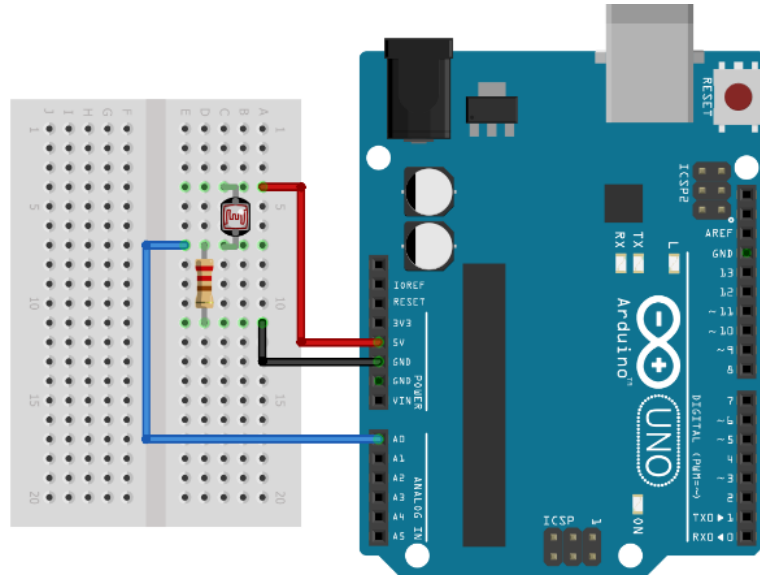


Figure 5.3: Circuit schematic for a single LDR

While figure 5.3 shows the working principle of a single LDR voltage divider, the project needed an array of LDRs to operate successfully. With careful analysis of the accuracy of the ultrasonic sensor in section 4.4 it was determined that a 3x3 array would be sufficient enough. A 3x3 array of LDRs would mean for nine analog pins on the Arduino, however the number of analog pins available were not sufficient. Therefore a multiplexer selector was chosen to increase the number of analog pins in the Arduino Uno. The LDR array was assembled on a breadboard for initial testing and then it was soldered onto a silicon board. With all the sensors procured, the sensor mount was designed by taking exact measurements of the sensor, components and vehicle dimensions.

5.3 Sensor Mount Design

The sensor mount was initially designed by hand drawing a three dimensional model. It was then transformed into a digital design using computer aided design software. The design has holes on the right side of the body for the ultrasonic sensor's transmitter and receiver to protrude and it has holes at the bottom so that it can be screwed onto the car. The top surface also has holes where the micro-controller and other components were screwed on. The sensor mount design is such that it has some empty space between the top surface of the mount and the base of the model vehicle's chassis. This space was intentionally incorporated in the design to make room for parts of the chassis to pass through and to place the LiPo battery. Figure 5.4 below shows the digital model of the sensor.

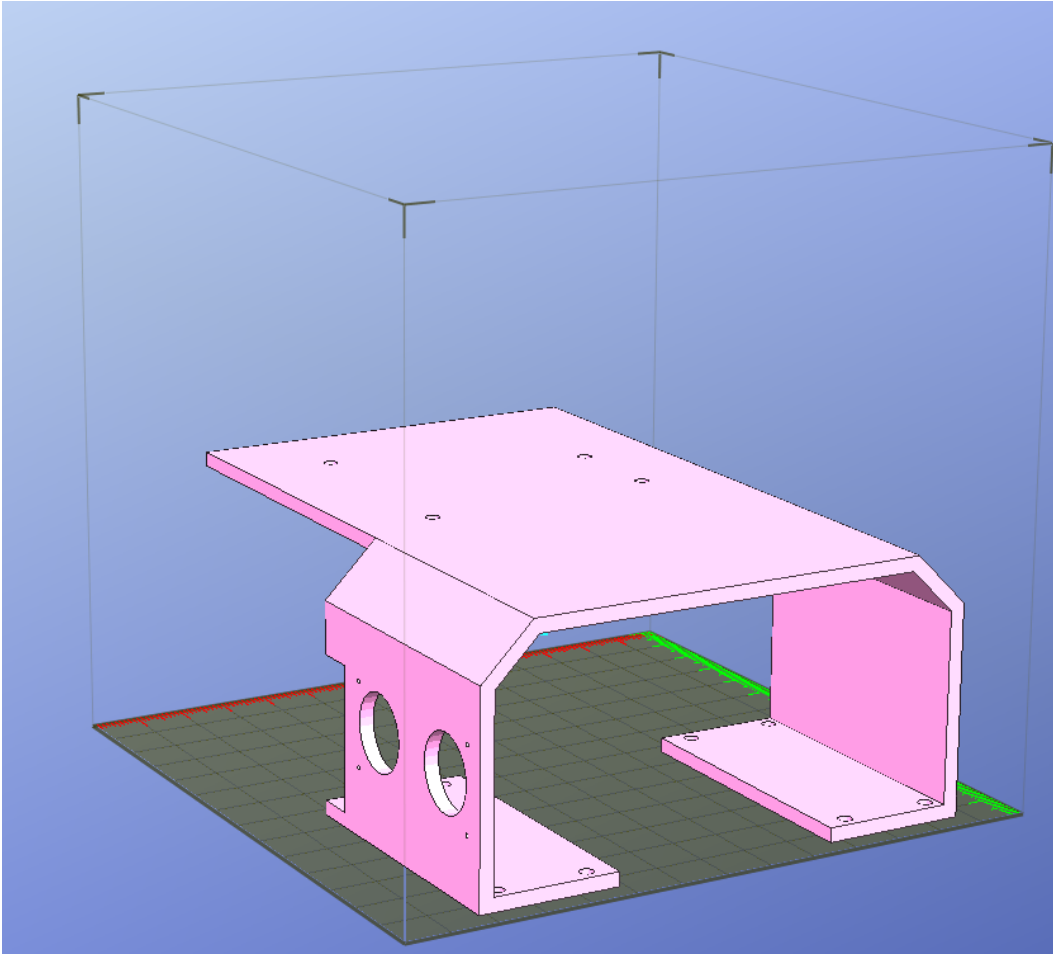


Figure 5.4: Digital model of the designed sensor mount

The digital model was then 3D printed using the available equipment at the laboratory. The print was successful. Figure 5.5 below shows the results of the print.

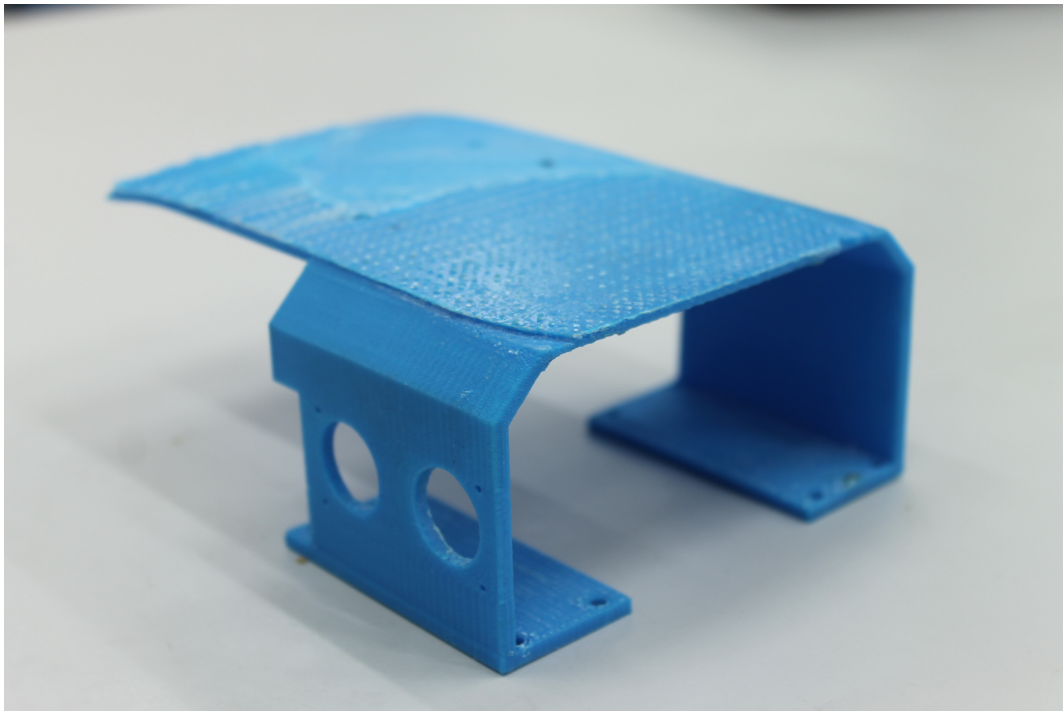
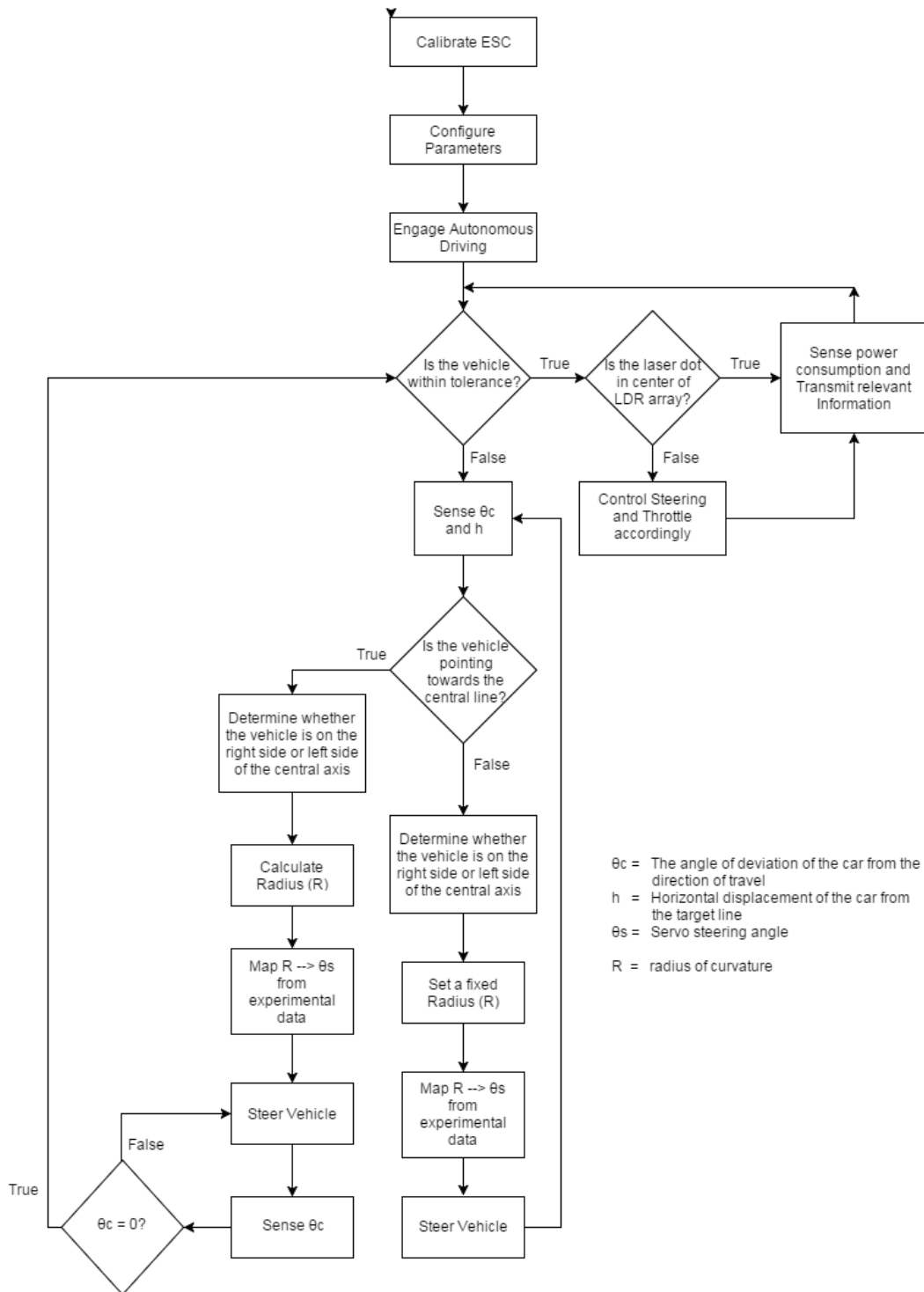


Figure 5.5: 3D printed sensor mount

5.4 The Final Control System

With the sensor mount printed and the appropriate sensors mounted on the model vehicle, the micro-controller needed to be programmed. The complete control algorithm that was developed at the end of section 3.5 is refined here so that the appropriate use of sensors, the calibration process and autonomous driving is implemented in the programming of the micro-controller. The refined control algorithm which in essence is the pseudo code for the micro-controller is given below in figure 5.6.

**Figure 5.6:** Refined control algorithm

Every time the electronic speed controller is disconnected from the battery it needs recalibration and at the beginning of the operation of the control system after the ESC

calibration the parameters such as default steering angle and the default PWM frequency for throttle response is configured. This configuration is a manual process where the operator will have to control the vehicle remotely from the PC. Once these parameters are configured, the autonomous driving mode can be engaged. Once the autonomous driving mode has been engaged the micro-controller controls the vehicle according the control algorithm shown in figure 5.6. At this stage of the project the model vehicle was finally built and the micro-controller was programmed accordingly. The complete schematic including every single components used in the implementation of the control system is given below in figure 5.7.

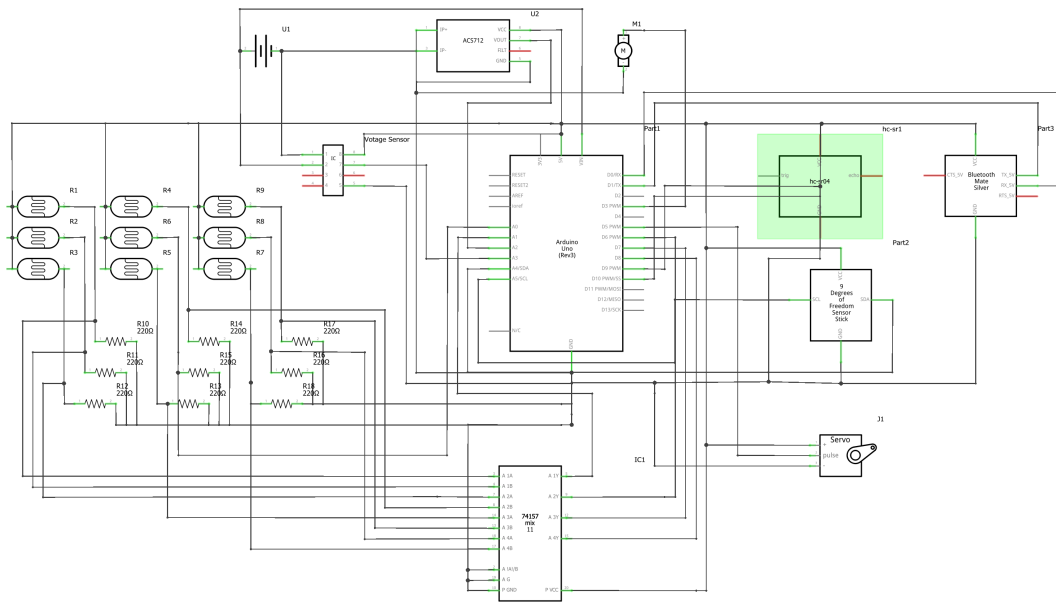


Figure 5.7: Complete schematic

Following the schematic the sensor mount was populated with the HC-SR04 ultrasonic distance sensor, the nine degree of freedom LSM9DS0 accelerometer magnetometer gyroscope, the LDR array, the HC06 bluetooth module and the Arduino Uno micro-controller itself. The current sensor and voltage sensor was not mounted on the sensor mount as they needed to be placed close to the LiPo battery. The populated sensor mount was then attached to the model vehicle and the final result of the model vehicle is shown below in figure ??.

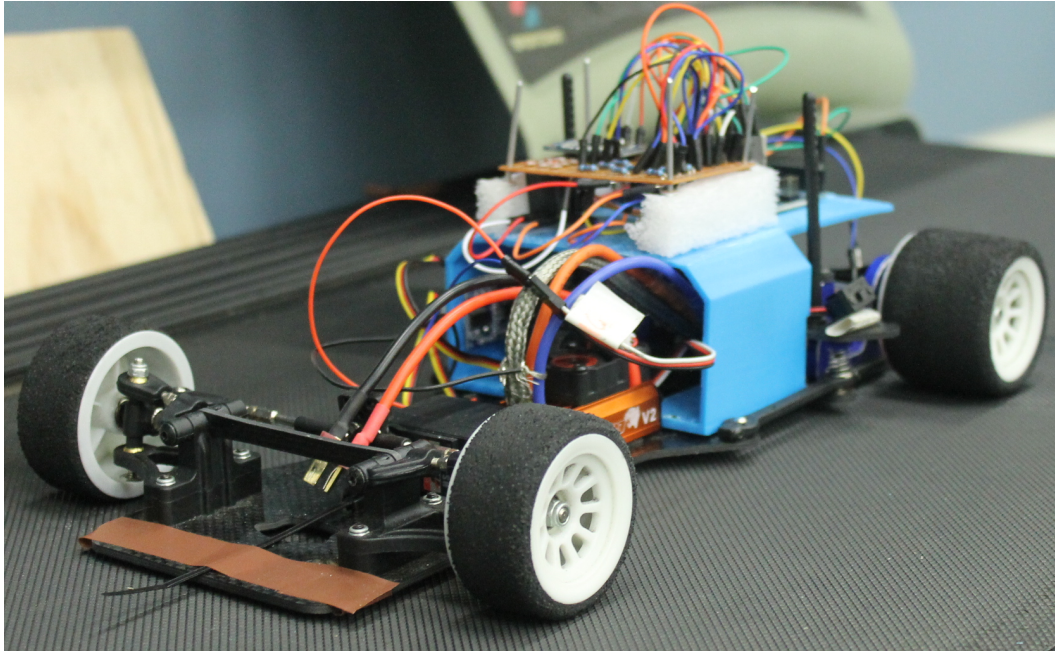


Figure 5.8: Model vehicle with the populated sensor mount and all components

5.4.1 Mapping The Calculated Radius to a Steering Angle

In the control algorithm, there is a control block that states the calculated radius needs to be mapped onto the steering angle. This block is essential in terms of programming the micro-controller. When programming the micro-controller, there were no means of calculating a servo steering angle from a radius of curvature without relevant reference data. In order to do understand the relationship between a radius of curvature and a steering angle the reference data was obtained experimentally. The experiment involved varying the steering angle and letting the vehicle drive itself for a known distance and measuring the resultant horizontal offset. Using these two lengths and applying geometric and trigonometric analysis the radius of curvature could be calculated. The experimental procedure is better illustrated with the aid of a diagram. This is shown below in figure 5.9 followed by the calculations.

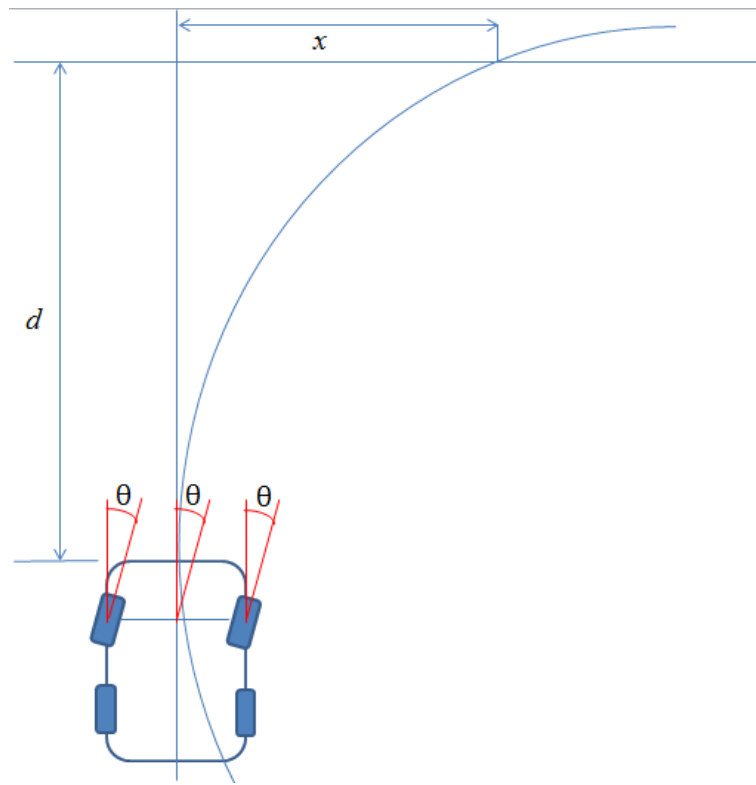


Figure 5.9: Experimental procedure to get relation between steering angle and radius of curvature

In figure 5.9 above 'd' is the known distance and 'x' is the measured horizontal offset. This is represented in geometric form in figure 5.10 to aid with the calculations.

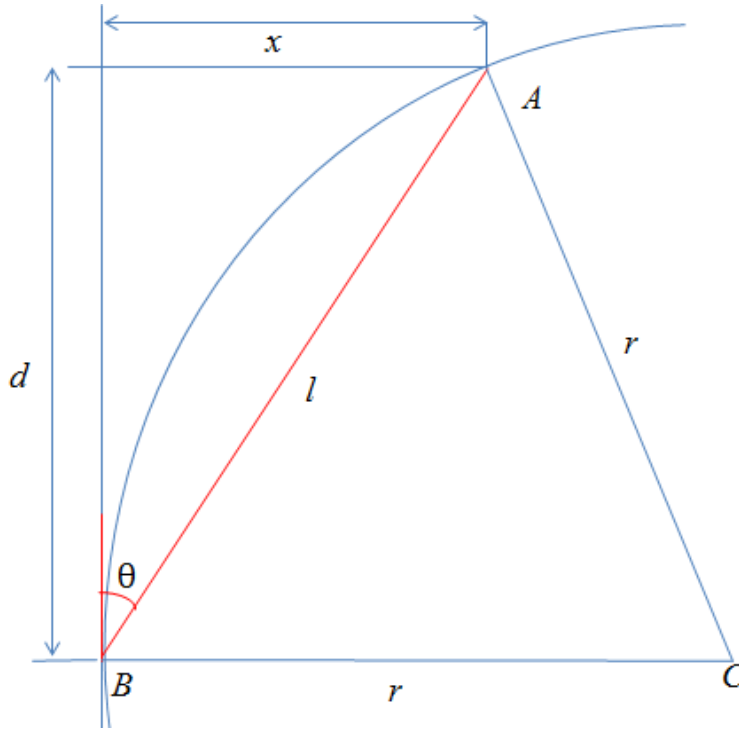


Figure 5.10: Geometric representation for calculation of radius

The length AB or ‘ l ’ can be calculated by pythagoras theorem

$$l = \sqrt{x^2 + d^2} \quad (5.1)$$

$$\theta = \arctan \frac{x}{d} \quad (5.2)$$

In $\triangle ABC$, $\angle ABC$ equals $90 - \theta$ as per rules from the circle theorem.

$$\cos(90 - \theta) = \frac{\frac{l}{2}}{r} \quad (5.3)$$

Therefore, rearranging equation 5.3 and substituting equations 5.1 and 5.2 in it the radius of curvature can be obtained by the following equation.

$$r = \frac{l}{2 \cos(90 - \arctan(\frac{x}{d}))} \quad (5.4)$$

The equation was used to calculate the radius for several different steering angle and a graph of radius vs steering angle was plotted. This is given below.

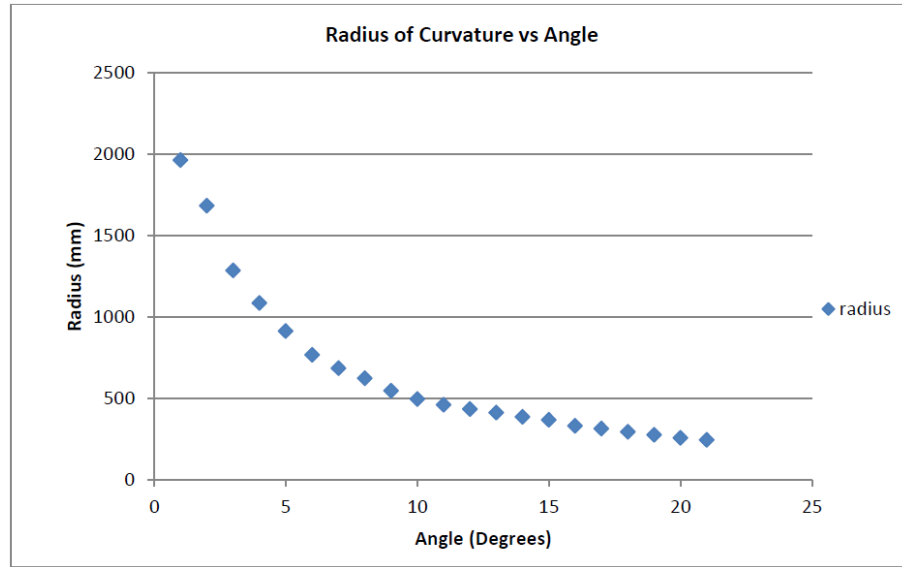


Figure 5.11: Radius of curvature vs steering angle

An analysis of the graph suggests that the relation is accurate. The steering angle and the radius of curvature should be inversely proportional and that is what the graph suggest as well. If the steering angle is small the the radius of curvature should be large and from the graph it can be seen that as the steering angle approaches zero the radius of curvature tends towards infinity and the y-axis in the graph is asymptotic to the graph. Using this relation, during the execution of the control algorithm, the calculated radius is mapped to a steering angle.

5.5 Summary

To summarize, this chapter discusses the implementation of the control system on a 1/10th scale model vehicle. The complete control algorithm developed in section 3.5 was refined in this chapter so that the control algorithm can be related to the new components that were researched for this project. The pre-existing project was further developed to build the final model of this project. In the process, the additional sensors were added to the model and to house these sensors, a sensor mount was designed and 3D printed. The sensor mount was then populated appropriately with the sensors and the final model vehicle was built. In order to execute the complete control algorithm the data needed for the block which maps calculated radius to steering angle was accumulated experimentally.

Chapter 6

Interfacing with the Control System

6.1 Introduction

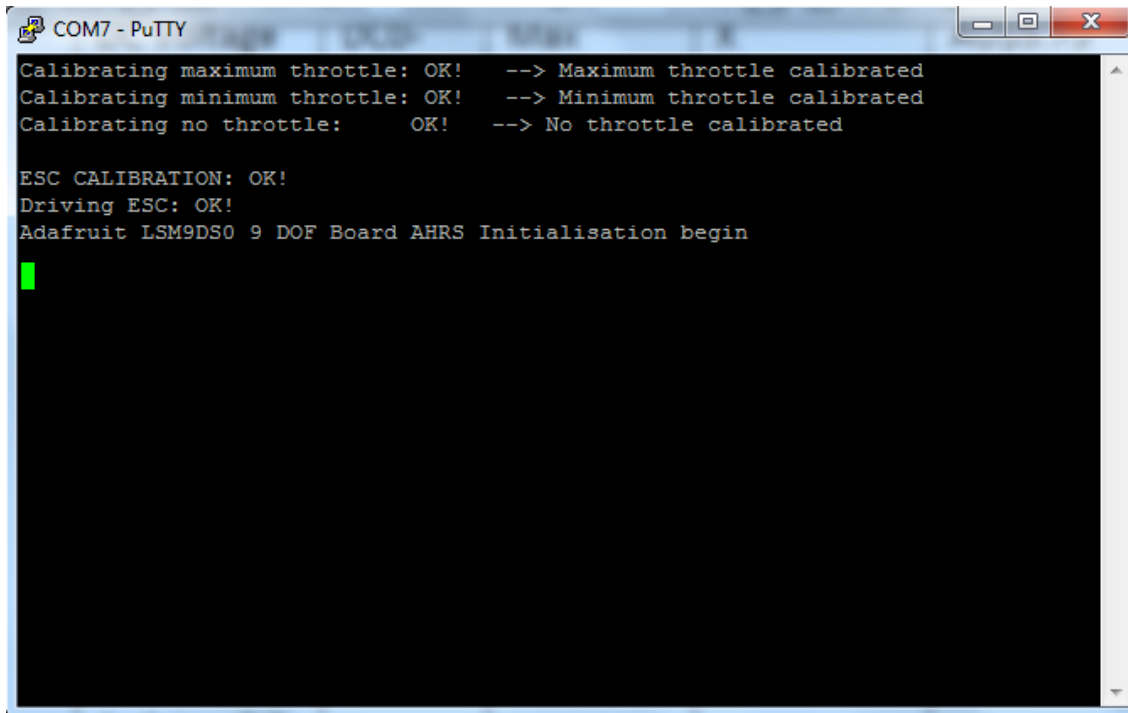
This chapter discusses the steps that were taken to interface with the control system from the computer. There were two means developed in order to interface with the control system. The first is a simple interfacing technique over a terminal program with a set baudrate and the second is a graphical user interface that can be used to represent the data accumulated while the control system is operational.

6.2 Interfacing Over a Terminal Program

In order to interface with the control system, the operator would essentially have to establish a serial connection between the computer and the arduino on the vehicle. A wireless method of communication was chosen by the use of bluetooth modules. A bluetooth dongle connected to the computer and paired with the bluetooth module on the vehicle can be used to establish a stable serial communication channel between the PC and the control system. Once the bluetooth module is connected and paired, the com port that it will communicate through can be found from the device manager. Once the COM port is identified, a terminal program needs to be launched with the correct COM port number and baudrate. For this project, PuTTY was used as the terminal program and the communication baudrate was set to be 9600 as per the arduino code.

The first step in the operation of the control algorithm is the calibration of the electronic speed controller (ESC). The calibration process is done in three steps. The first step is to set the maximum throttle. At this point the arduino outputs a PWM signal with a frequency of 180Hz. Once the maximum throttle is adjusted, the ESC makes a beep sequence of four beeps. Then pressing any key on the keyboard advances the calibration process to the next step. In this step the minimum throttle is set. Here the PWM signal frequency is set to 0Hz. A similar beep sequence is produced once the calibration is completed. The last step in the calibration process is the recognition of the neutral throttle value. In this case the PWM signal frequency value is set to be halfway between the max-

imum value and minimum value i.e. 90Hz. Once the maximum, minimum and neutral throttle values are set, the ESC is calibrated and at this stage the ESC confirms this by producing a beep sequence of four beeps followed by an additional two beeps making a total of six beeps. After the ESC is calibrated, the accelerometer is initialized. During the whole calibration process, the control system was programmed to send strings of text over the serial connection established to communicate with the operator. An illustration of this communication terminal is shown below in figure 6.1



```
COM7 - PuTTY
Calibrating maximum throttle: OK! --> Maximum throttle calibrated
Calibrating minimum throttle: OK! --> Minimum throttle calibrated
Calibrating no throttle: OK! --> No throttle calibrated

ESC CALIBRATION: OK!
Driving ESC: OK!
Adafruit LSM9DS0 9 DOF Board AHRS Initialisation begin
```

Figure 6.1: The serial terminal during calibration

Upon successful completion of the calibration process, the reference steering angle needed to be configured. This is done by adjusting the vehicle's position on the center of the moving ground by manually incrementing or decrementing the steering angle. The arduino was programmed to recognize the transmission of numeric key '8' and '9' as commands to increment and decrement the steering angle. The orientation in which the steering servo was mounted was as such that an increment in the steering angle would steer the vehicle left and a decrement would steer it right. Once the operator is satisfied with the vehicle's position on the moving ground, pressing the the numeric key '7' initializes the autonomous driving. Once the autonomous driving is engaged, the data that is being sensed by the sensors are transmitted over the wireless serial communication established at the beginning. An image of the terminal screen during the operation of the autonomous driving is given below followed by a discussion of the transmitted data.

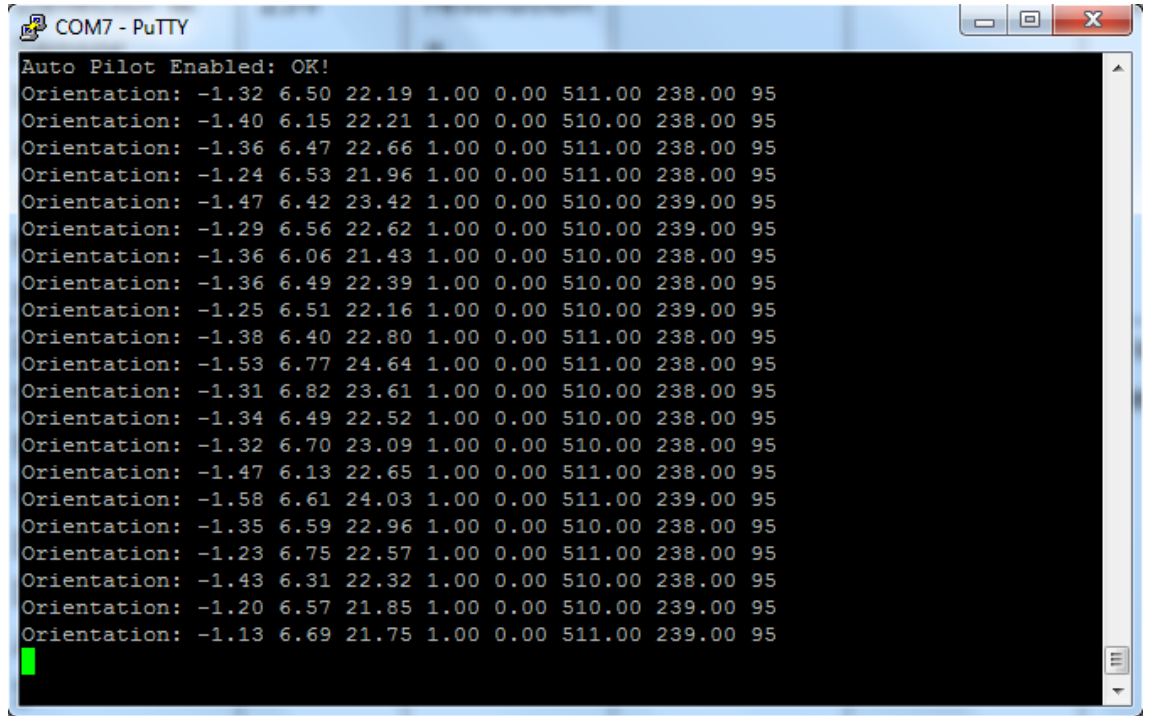


Figure 6.2: The serial terminal during autonomous driving

The data transmitted is the orientation of the vehicle along the X, Y and Z axis followed by five floating values. The X, Y and Z axis gyroscope readings and the five floating values are each separated by a 'space' character. Out of the five floating values, the first two can only take values of either zero or one and these two characters indicate the activation of autonomous driving and operation in tolerance mode respectively. The next two figures are the current and voltage readings respectively. The last figure is the PWM frequency value.

6.3 Graphical User Interface

In an attempt to represent the data in a more visual manner rather than just continuous strings of text an attempt was made to develop a graphical user interface (GUI). However this graphical user interface can only be accessed once the calibration and configuring steps has been completed and the autonomous driving mode has been activated.

As discussed in the previous section, the data transmitted consists of the vehicle's orientation, indication of autonomous driving activation and tolerance mode operation and the power consumption details. In order to represent this data, the vehicle's orientation was best illustrated by modeling the vehicle as a cube that would mimic the orientation of the vehicle in real time. The rest of the data was best represented in an information dialog box beside the cube. The indication of autonomous driving mode and tolerance mode was illustrated with the aid of colored blocks, where a red block indicates deacti-

vated stage and a green block indicates active stage. The GUI is shown below in a series of three images.

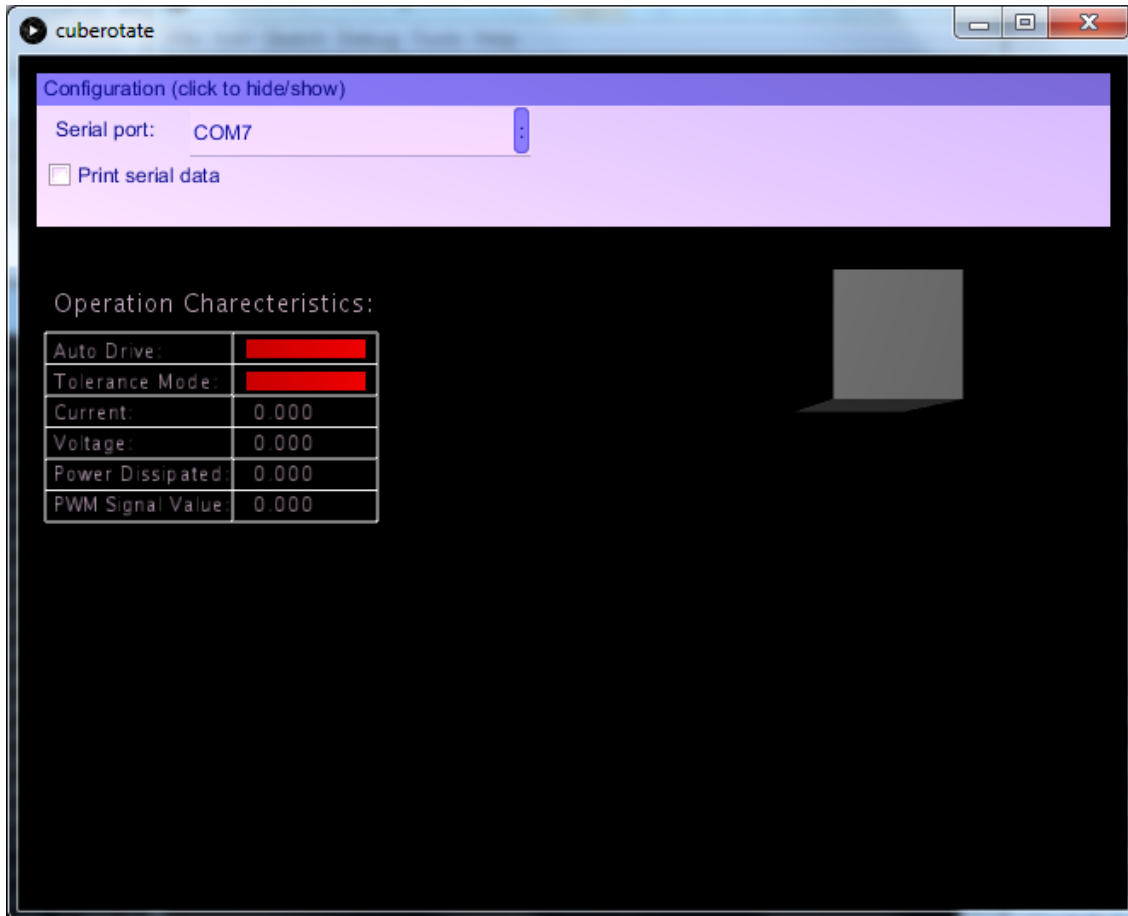


Figure 6.3: The idle graphical user interface

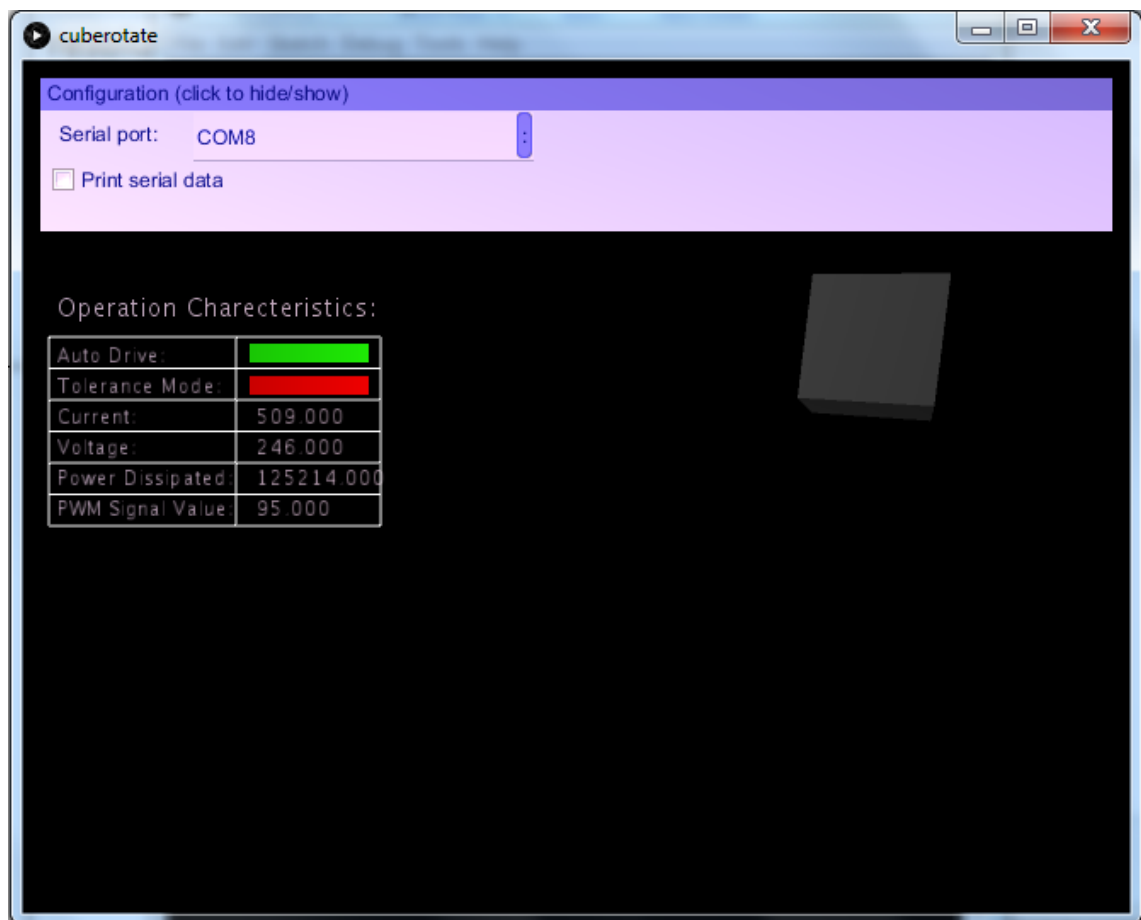


Figure 6.4: The graphical user interface during normal mode of operation

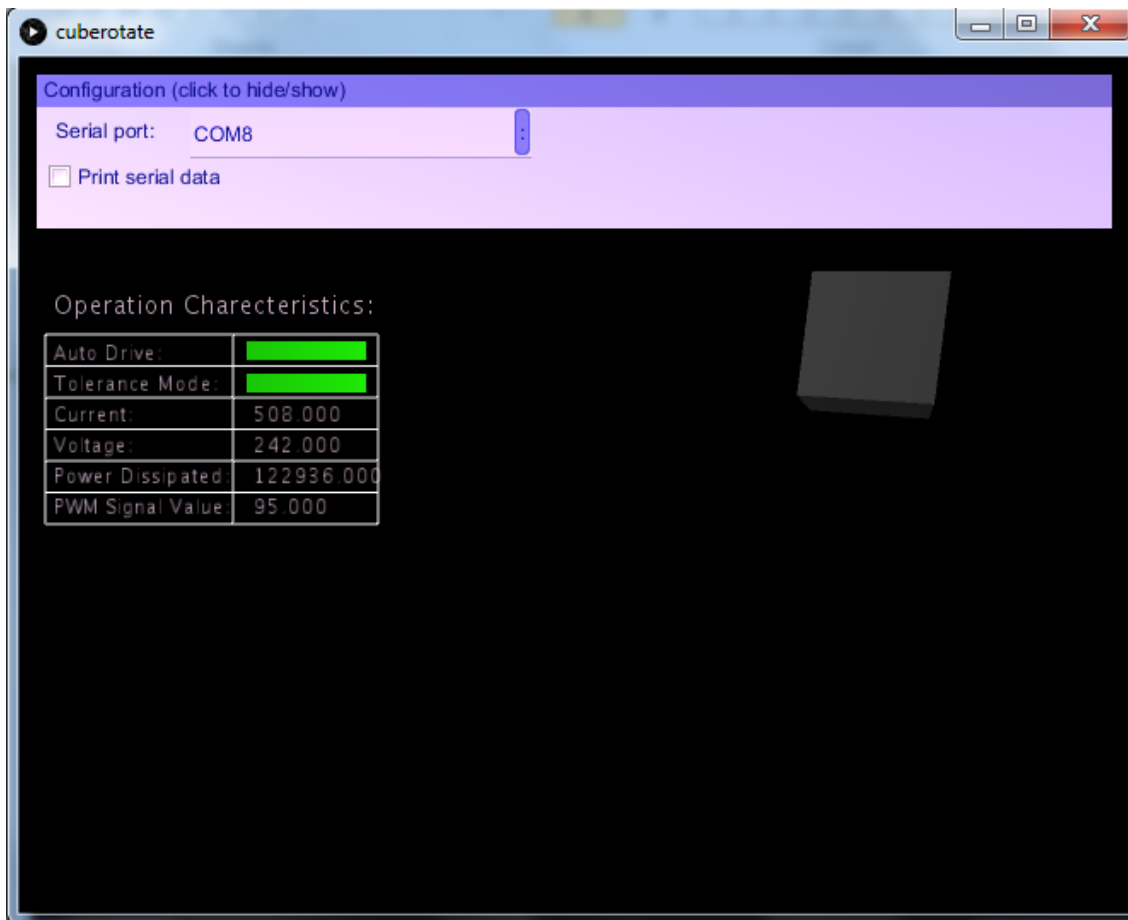


Figure 6.5: The graphical user interface during tolerance mode of operation

Figure 6.3, 6.4 and 6.5 above show three stages of the GUI. Figure 6.3 shows the GUI when there is no data being transmitted. Here the dialog box is empty and the cube is in its default orientation. Figure 6.4 above shows the GUI when the control system is operational and autonomous driving has been enabled but it is not in the tolerance mode and figure 6.5 shows both autonomous driving and tolerance mode engaged. In these figures the autonomous driving block and the tolerance mode block is green and red accordingly and the dialog box consists of the current reading, voltage reading and PWM signal frequency.

6.4 Summary

This chapter is about the communication protocol used and methods of interfacing with the control system. In summary the communication between the control system and PC is done wirelessly over bluetooth. Two different methods of interfacing with the control system was developed. The first method was interfacing over a terminal program and the second was a graphical user interface. The terminal program is used to calibrate and

configure the control system's default parameters and shows the quantitative data that is transmitted from the control system. The second interfacing method is the graphical one where the vehicle is modeled as a cube and a dialog box is present displaying all the accumulated data.

Chapter 7

Conclusions

In conclusion, this thesis project took an intuitive approach to develop a control system that can maintain a model vehicle's position in a wind tunnel with high degree of accuracy and measure the drag the vehicle is experiencing. Existing wind tunnel facilities and methods of measuring drag was reviewed. Relevant autonomous driving algorithms were reviewed and the concept of determining the vehicles trajectory by obtaining a radius of curvature was adopted. A mathematical approach was taken to realize the vehicle steering dynamics and a single equation was obtained that takes two input parameters of a horizontal displacement and angle of deviation from the direction of travel and outputs a radius of curvature. An experiment was done to map the calculated radius to a steering angle. After the mathematical analysis it was observed that the control system would perform best if there were to be two modes of operation, one where the vehicle is not in tolerance, the normal mode and the other when the vehicle is within tolerance, tolerance mode. Separate control algorithms were developed for each of the modes and then combined to obtain one complete algorithm. As part of the control algorithm, a technique to measure the drag the vehicle experiences was also developed. Sensors that could realize the inputs to the control algorithm were researched. The sensors were researched under a specific criteria and the ones that best fitted the criteria were chosen. The chosen sensors were also tested and finally the control system was implemented on a 1/10th scale model. The control system performed according to the project specifications and was able to measure power consumption for drag determination.

Future Work

Whilst the control system performed within acceptable tolerance, there is room for improvement. The program running in the micro-controller can be further optimized. As part of the optimization process, the LDR array and multiplexer arrangement that is being used in the tolerance mode needs to be further developed such that they have their own computational capacity and the new arrangement of the tolerance mode sensors allow for quicker sensing of the minute movement in the tolerance region. The drag measurement can also be further improved. Currently the power consumption is measured by measuring

the power input to the electronic speed controller and there is question of efficiency. This efficiency factor needs to be further analyzed. Another area of improvement is the smooth launching ability. At the moment the operation of the control system is such that the increase in the speed of the brushless motor system going from no throttle to minimum throttle is very rapid. So the operator needs to place the vehicle on the moving ground when the ground is moving above a certain speed. A technique needs to be developed so that this increase in speed is gradual and not rapid.

Appendix A

HC-SR04 Datasheet



Tech Support: services@elecfreaks.com

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

- (1) Using IO trigger for at least 10us high level signal,
- (2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
- (3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time × velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

Appendix B

Arduino Code

B.1 Version 1

```
#include <Servo.h>

// Accelerometer package code incorporation begin
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS0.h>
#include <Adafruit_Simple_AHRS.h>

// Create LSM9DS0 board instance.
Adafruit_LSM9DS0 lsm(1000); // Use I2C, ID #1000

// Create simple AHRS algorithm using the LSM9DS0 instance's accelerometer
Adafruit_Simple_AHRS ahrs(&lsm.getAccel(), &lsm.getMag());

// Function to configure the sensors on the LSM9DS0 board.
// You don't need to change anything here, but have the option to select
// range and gain values.
void configureLSM9DS0(void)
{
    // 1.) Set the accelerometer range
    lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_2G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_4G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_6G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_8G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_16G);

    // 2.) Set the magnetometer sensitivity
```

```

lsm.setupMag(lsm.LSM9DS0_MAGGAIN_2GAUSS);
//lsm.setupMag(lsm.LSM9DS0_MAGGAIN_4GAUSS);
//lsm.setupMag(lsm.LSM9DS0_MAGGAIN_8GAUSS);
//lsm.setupMag(lsm.LSM9DS0_MAGGAIN_12GAUSS);

// 3.) Setup the gyroscope
lsm.setupGyro(lsm.LSM9DS0_GYROSCALE_245DPS);
//lsm.setupGyro(lsm.LSM9DS0_GYROSCALE_500DPS);
//lsm.setupGyro(lsm.LSM9DS0_GYROSCALE_2000DPS);
}
//Accelerometer package code incorporation end

Servo driveServo;
Servo steerServo;

const int trigPin = 9;
const int echoPin = 10;

long duration;
int distance;

const int analogInPin0 = A0; // Analog input pin that the potentiometer is att
const int analogInPin1 = A1; // Analog input pin that the potentiometer is att
//const int analogInPin2 = A2; // Analog input pin that the potentiometer is a

const int currentPin = A2; // Analog input pin that the potentiometer is attac
const int voltagePin = A3; // Analog input pin that the potentiometer is attac

int sensorValue[3];
int outputValue[3];

int minVal;
int LDR;

int driveServoPin = 3;
int steerServoPin = 5;
int instruction = 0;
int steerAngle = 87;
int manualAdjustedAngle = 87;
int bluetoothPower = 12;

float currentDrawn = 0;
float voltageDrop = 0;

```

```
int currentThrottle = 95;

bool autoPilotUSND = false;
bool autoPilotLDR = false;
float tolerance = 0;
float autoDrive = 0;

void setup() {
    // put your setup code here, to run once:
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    pinMode(bluetoothPower, OUTPUT);
    digitalWrite(bluetoothPower, HIGH);

    driveServo.attach(driveServoPin);
    steerServo.attach(steerServoPin);

    Serial.begin(9600);
    while(!Serial){}
    calibrateDriveServo();
    Serial.println("");
    Serial.println("ESC CALIBRATION: OK!");
    driveServo.write(95);
    steerServo.write(steerAngle);
    Serial.println("Driving ESC: OK!");

    //Accelerometer initilisation begin
    Serial.println(F("Adafruit LSM9DS0 9 DOF Board AHRS Initialisation be

    // Initialise the LSM9DS0 board.
    if(!lsm.begin())
    {
        // There was a problem detecting the LSM9DS0 ... check your connect
        Serial.print(F("No LSM9DS0 detected ... Check wiring or I2C ADDR!"));
        while(1);
    }

    // Setup the sensor gain and integration time.
    configureLSM9DS0();
    //Accelerometer initialisation end
}
```

```
void loop() {
  // put your main code here, to run repeatedly:
  int temp = instruction;
  if(Serial.available())
    instruction = Serial.read() - 48;

  /*
  if(instruction == 0)
    driveServo.write(95);
  else if(instruction == 1)
    driveServo.write(100);
  else if(instruction == 2)
    driveServo.write(105);
  else if(instruction == 3)
    driveServo.write(110);
  //else
    //driveServo.write(90);
  */
  if(instruction == 1)
  {
    currentThrottle++;
    instruction = temp;
  }
  else if(instruction == 2)
  {
    currentThrottle--;
    instruction = temp;
  }

  if(currentThrottle < 180)
  {
    driveServo.write(currentThrottle);

    Serial.print("Throttle Value: ");
    Serial.print(currentThrottle);
    Serial.print(" units");
    Serial.print(" \t");
  }

  if(instruction == 7)
  {
    autoPilotUSND = true;
  }
}
```



```
    autoDrive = 1;
    instruction = temp;
    Serial.println("Auto Pilot USND Enabled: OK!");
}
else if(instruction == 6)
{
    autoPilotUSND = false;
    autoDrive = 0;
    instruction = temp;
    Serial.println("Auto Pilot USND Disabled: OK!");
}

if(instruction == 5)
{
    autoPilotLDR = true;
    instruction = temp;
    Serial.println("Auto Pilot LDR Enabled: OK!");
}
else if(instruction == 4)
{
    autoPilotLDR = false;
    instruction = temp;
    Serial.println("Auto Pilot LDR Disabled: OK!");
}

if(!autoPilotUSND && !autoPilotLDR)
{
    if(instruction == 9)
    {
        steerAngle++;
        instruction = temp;
        manualAdjustedAngle = steerAngle;
    }
    if(instruction == 8)
    {
        steerAngle--;
        instruction = temp;
        manualAdjustedAngle = steerAngle;
    }
    steerServo.write(steerAngle);
    Serial.print("Manually Adjusted Angle: ");
```

```
    Serial.print(manualAdjustedAngle);
    Serial.println(" degrees");
}

while(autoPilotUSND)
{

    temp = instruction;
    if(Serial.available())
        instruction = Serial.read() - 48;

    if(instruction == 6)
    {
        autoPilotUSND = false;
        autoDrive = 0;
        instruction = temp;
        Serial.println("Auto Pilot USND Disabled: OK!");
    }

    //int currentDistance = distanceMeasured;
    //autoDrive = 1;
    //_____
    //Ultrasonic position maintaining code
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);

    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);
    distance = duration*0.034/2.0;
    //Serial.print(" Current Distance: ");
    //Serial.print(distance);
    //Serial.println("cm");

    if(distance < 25)
    {
        steerAngle++;
        steerServo.write(steerAngle);
        //delayMicroseconds(3000);
        delay(50);
    }
}
```

```

        steerAngle--;
        //instruction = temp;
    }
    else if (distance > 25)
    {
        steerAngle--;
        steerServo.write(steerAngle);
        //delayMicroseconds(3000);
        delay(50);
        steerAngle++;
        //instruction = temp;
    }
    else
    {
        steerAngle = manualAdjustedAngle;
    }
    if (abs(distance - 25) <= 5)
        tolerance = 1;
    else
        tolerance = 0;
    //-----
    currentDrawn = analogRead(currentPin);
    voltageDrop = analogRead(voltagePin);

    //Accelerometer code incorporated begin
    sensors_vec_t    orientation;

    // Use the simple AHRS function to get the current orientation.
    if (ahrs.getOrientation(&orientation))
    {
        /* 'orientation' should have valid .roll and .pitch fields */
        Serial.print(F(" Orientation: "));
        Serial.print(orientation.roll);
        Serial.print(F(" "));
        Serial.print(orientation.pitch);
        Serial.print(F(" "));
        Serial.print(orientation.heading);
        Serial.print(F(" "));
        Serial.print(autoDrive);
        Serial.print(" ");
        Serial.print(tolerance);
        Serial.print(" ");
        Serial.print(currentDrawn);
    }

```

```

    Serial.print(" ");
    Serial.print(voltageDrop);
    Serial.print(" ");
    Serial.print(currentThrottle);
    Serial.println(" ");
}

//delay(100);
//Accelerometer code incorporated end

}
if(autoPilotLDR)
{
    //*****
    // LDR position maintaining code
    // read the analog in value:
    // and
    // map it to the range of the analog out:
    sensorValue[0] = analogRead(analogInPin0);
    outputValue[0] = map(sensorValue[0], 0, 1023, 0, 255);

    sensorValue[1] = analogRead(analogInPin1);
    outputValue[1] = map(sensorValue[1], 0, 1023, 0, 255);

    //sensorValue[2] = analogRead(analogInPin2);
    //outputValue[2] = map(sensorValue[2], 0, 1023, 0, 255);

    minVal = outputValue[1];
    LDR = 1;
    for(int i = 0; i < 2; i++)
    {
        if(outputValue[i] < minVal)
        {
            minVal = outputValue[i];
            LDR = i;
        }
    }
}

if(LDR == 0)
{
    Serial.println("Steer Right");
    steerAngle--;
    steerServo.write(steerAngle);
}

```

```
        delayMicroseconds(3000);
        //delay(50);
        steerAngle++;
    }
    else if (LDR == 2)
    {
        Serial.println("Steer Left");
        steerAngle++;
        steerServo.write(steerAngle);
        delayMicroseconds(3000);
        //delay(50);
        steerAngle--;
    }
    else
    {
        Serial.println("Steer Straight");
    }
    //*****
}

}

void calibrateDriveServo()
{
    int maxThrottle = 180;
    int minThrottle = 0;
    int noThrottle = 90;
    int discard;

    //Outputting Maximum throttle value and calibrating the ESC
    Serial.print("Calibrating maximum throttle: ");
    while(!Serial.available())
    {
        driveServo.write(maxThrottle);
    }
    discard = Serial.read();
    Serial.println("OK!    —> Maximum throttle calibrated");

    //
    Serial.print("Calibrating minimum throttle: ");
    while(!Serial.available())
    {
        driveServo.write(minThrottle);
```

```

}
discard = Serial.read();
Serial.println("OK!    —> Minimum throttle calibrated");

Serial.print(" Calibrating no throttle: ");
while(!Serial.available())
{
    driveServo.write(noThrottle);
}
discard = Serial.read();
Serial.println("    OK!    —> No throttle calibrated");
}

```

```

int distanceMeasured()
{
    long duration;
    int distance;

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);

    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);
    distance = duration*0.034/2.0;
    Serial.print(" Current Distance: ");
    Serial.print(distance);
    Serial.print("cm");

    return distance;
}

```

B.2 Version 2

```

#include <Servo.h>
//#include <SoftwareSerial.h>

//Req. Acc lib begin
#include <SPI.h>

```

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS0.h>
#include <Adafruit_Simple_AHRS.h>

// Create LSM9DS0 board instance.
Adafruit_LSM9DS0 lsm(1000); // Use I2C, ID #1000

// Create simple AHRS algorithm using the LSM9DS0 instance's accelerometer
Adafruit_Simple_AHRS ahrs(&lsm.getAccel(), &lsm.getMag());

// Function to configure the sensors on the LSM9DS0 board.
// You don't need to change anything here, but have the option to select
// range and gain values.
void configureLSM9DS0(void)
{
    // 1.) Set the accelerometer range
    lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_2G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_4G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_6G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_8G);
    //lsm.setupAccel(lsm.LSM9DS0_ACCEL_RANGE_16G);

    // 2.) Set the magnetometer sensitivity
    lsm.setupMag(lsm.LSM9DS0_MAGGAIN_2GAUSS);
    //lsm.setupMag(lsm.LSM9DS0_MAGGAIN_4GAUSS);
    //lsm.setupMag(lsm.LSM9DS0_MAGGAIN_8GAUSS);
    //lsm.setupMag(lsm.LSM9DS0_MAGGAIN_12GAUSS);

    // 3.) Setup the gyroscope
    lsm.setupGyro(lsm.LSM9DS0_GYROSCALE_245DPS);
    //lsm.setupGyro(lsm.LSM9DS0_GYROSCALE_500DPS);
    //lsm.setupGyro(lsm.LSM9DS0_GYROSCALE_2000DPS);
}

//Req. Acc lib end

//SoftwareSerial Serial(7, 8); // RX, TX

float h;
float angleValue;
float angle_calc;
```

```
float theta_n;
float theta_calc;

float xCoor;
float radius;

Servo driveServo;
Servo steerServo;

const int trigPin = 9;
const int echoPin = 10;

long duration;
int distance;

const int centroLDR = A0; // Analog input pin that the potentiometer is attach
const int toleranceLDR = A1; // Analog input pin that the potentiometer is attach
const int currentPin = A2; // Analog input pin that the potentiometer is attach
const int voltagePin = A3; // Analog input pin that the potentiometer is attach

float currentDrawn = 0;
float voltageDrop = 0;
float theta;

//int ldrValue[3];
//int ldrOutputValue[3];

int minVal;
int LDR;

int driveServoPin = 3;
int steerServoPin = 5;
int instruction = 0;
int steerAngle = 90;//87
int manualAdjustedAngle = 90;//87
int bluetoothPower = 12;
int tolCheckCount = 0;

int currentThrottle = 95;

bool autoPilotUSND = false;
bool autoPilotLDR = false;
```



```
bool vehicleInTolerance = false;
bool autonomousDriving = false;

void setup() {
    // put your setup code here, to run once:
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    pinMode(bluetoothPower, OUTPUT);
    digitalWrite(bluetoothPower, HIGH);

    driveServo.attach(driveServoPin);
    steerServo.attach(steerServoPin);

    //----- Accelerometer Setup begin
    Serial.begin(115200);
    Serial.println(F("Adafruit LSM9DS0 9 DOF Board AHRS Example")); Serial

    // Initialise the LSM9DS0 board.
    if(!lsm.begin())
    {
        // There was a problem detecting the LSM9DS0 ... check your connect
        Serial.print(F("Oops, no LSM9DS0 detected ... Check your wiring on
        while(1);
    }

    // Setup the sensor gain and integration time.
    configureLSM9DS0();

    //----- Accelerometer Setup end

    //Serial.begin(9600);
    while(!Serial){}
    calibrateDriveServo();
    Serial.println("");
    Serial.println("ESC CALIBRATION: OK!");
    driveServo.write(95);
    steerServo.write(steerAngle);
    Serial.println("Driving ESC: OK!");
}

void loop() {
```

```
// put your main code here, to run repeatedly:
int temp = instruction;
if(Serial.available())
    instruction = Serial.read() - 48;

//Throttle control code begin
if(instruction == 1)
{
    currentThrottle++;
    instruction = temp;
}
else if(instruction == 2)
{
    currentThrottle--;
    instruction = temp;
}

if(currentThrottle < 180)
{
    driveServo.write(currentThrottle);

    Serial.print("Throttle Value: ");
    Serial.print(currentThrottle);
    Serial.print(" units");
    Serial.print(" \t");
}
//Throttle control code end

if(instruction == 7)
{
    autoPilotUSND = true;
    instruction = temp;
    Serial.println("Auto Pilot USND Enabled: OK!");
}
else if(instruction == 6)
{
    autoPilotUSND = false;
    instruction = temp;
    Serial.println("Auto Pilot USND Disabled: OK!");
}

if(instruction == 5)
{
```

```

    autoPilotLDR = true;
    instruction = temp;
    Serial.println("Auto Pilot LDR Enabled: OK!");
}
else if(instruction == 4)
{
    autoPilotLDR = false;
    instruction = temp;
    Serial.println("Auto Pilot LDR Disabled: OK!");
}

//8/6/16
while(autonomousDriving)
{
    checkTolerance();
    while(vehicleInTolerance)
    {
        if(analogRead(centroLDR) < 50)
        {
            currentDrawn = analogRead(currentPin);
            voltageDrop = analogRead(voltagePin);
            Serial.println("Current and Voltage Measured");
            Serial.print(currentDrawn);
            Serial.print(voltageDrop);
            Serial.println();
        }
        else
        {
            if(tolCheckCount == 3 || tolCheckCount == 5 || tolCheckCount == 7)
            {
                steerAngle--;
                steerServo.write(steerAngle);
            }
            else if(tolCheckCount == 1 || tolCheckCount == 4 || tolCheckCount == 6)
            {
                steerAngle++;
                steerServo.write(steerAngle);
            }
            if(tolCheckCount == 1 || tolCheckCount == 2 || tolCheckCount == 3)
            {
                currentThrottle++;
            }
            else if(tolCheckCount == 4 || tolCheckCount == 5 || tolCheckCount == 6)
            {
                currentThrottle--;
            }
        }
    }
}

```

```

    {
        currentThrottle--;
    }

    driveServo.write(currentThrottle)
    steerServo.write(manualAdjustedAngle);
}
checkTolerance();
}
while(!vehicleInTolerance)// else
{
    h = abs(getDistance() - 250);
    angleValue = abs(getAngle());
    //need to sense distance here

    angle_calc = (180 - angleValue)/2;
    theta = (angleValue*PI)/180;
    theta_calc = (angle_calc*PI)/180;

    if((theta > 0 && distance > 25) || (theta < 0 && distance < 25))
    {
        //calculate Radius
        xCoor = h*tan(theta_calc);
        radius = (xCoor/tan(theta)) + h;

        //map radius to steering angle

        if(radius > 1000)
            steerAngle+=2;
        else
            steerAngle+=5;

        steerServo.write(steerAngle);
        delayMicroseconds(50);
        steerServo.write(manualAdjustedAngle);

        while(abs(theta) >= 0.3)
        {
            theta = getAngle();
        }
    }
    else
    {

```

```
        if(distance > 25)
        {
            steerAngle++; //steer by fixed radius
        }
        else
        {
            steerAngle--; //steer by fixed radius
        }
        steerServo.write(steerAngle);
    }
    checkTolerance();
}
}
//8/6/16

if(!autoPilotUSND && !autoPilotLDR)
{
    if(instruction == 8)//swapped for new servo
    {
        steerAngle++;
        instruction = temp;
        manualAdjustedAngle = steerAngle;
    }
    if(instruction == 9)//swapped for new servo
    {
        steerAngle--;
        instruction = temp;
        manualAdjustedAngle = steerAngle;
    }
    steerServo.write(steerAngle);
    Serial.print(" Manually Adjusted Angle: ");
    Serial.print(manualAdjustedAngle);
    Serial.println(" degrees");
}

if(autoPilotUSND)
{
    //int currentDistance = distanceMeasured;

    //-----
    //Ultrasonic position maintaining code
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
```

```

digitalWrite(trigPin , HIGH);
delayMicroseconds(10);
digitalWrite(trigPin , LOW);

duration = pulseIn(echoPin , HIGH);
distance = duration*0.034/2.0;
Serial.print(" Current Distance: ");
Serial.print(distance);
Serial.println("cm");

if(distance > 25) //swapped for new servo
{
    steerAngle++;
    steerServo.write(steerAngle);
    delayMicroseconds(3000);
    //delay(50);
    steerAngle--;
    //instruction = temp;
}
else if(distance < 25) //swapped for new servo
{
    steerAngle--;
    steerServo.write(steerAngle);
    delayMicroseconds(3000);
    //delay(50);
    steerAngle++;
    //instruction = temp;
}
else
{
    steerAngle = manualAdjustedAngle;
}
//_____

}
}

void calibrateDriveServo()
{

```

```
int maxThrottle = 180;
int minThrottle = 0;
int noThrottle = 90;
int discard;

Serial.println("Begin initialisation process");
while(!Serial.available()){
  discard = Serial.read();
  Serial.println("Initialisation Started");

  //Outputting Maximum throttle value and calibrating the ESC
  Serial.print("Calibrating maximum throttle: ");
  while(!Serial.available())
  {
    driveServo.write(maxThrottle);
  }
  discard = Serial.read();
  Serial.println("OK!    —> Maximum throttle calibrated");

  //
  Serial.print("Calibrating minimum throttle: ");
  while(!Serial.available())
  {
    driveServo.write(minThrottle);
  }
  discard = Serial.read();
  Serial.println("OK!    —> Minimum throttle calibrated");

  Serial.print("Calibrating no throttle: ");
  while(!Serial.available())
  {
    driveServo.write(noThrottle);
  }
  discard = Serial.read();
  Serial.println("    OK!    —> No throttle calibrated");
}

void checkTolerance()
{
  vehicleInTolerance = false;
  float ldrVal[9];
```

```
for(int i = 0; i < 2; i++)
{
    if(i == 1)
        digitalWrite(6, HIGH);
    else
        digitalWrite(6, LOW);
    for(int j = 0; i < 2; i++)
    {
        if(j == 1)
            digitalWrite(7, HIGH);
        else
            digitalWrite(7, LOW);
        for(int k = 0; i < 2; i++)
        {
            if(k == 1)
                digitalWrite(8, HIGH);
            else
                digitalWrite(8, LOW);
            ldrVal[count] = map(analogRead(toleranceLDR), 0, 1023, 0, 255);
            tolCheckCount++;
        }
    }
}

for(int i = 0; i < 9; i++)
{
    if(ldrVal[i] < 50)
        vehicleInTolerance = true;
    else
        vehicleInTolerance = false;
}

}

float getDistance()
{
    long durationN;
    float distanceN;

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
```



```
digitalWrite(trigPin , HIGH);
delayMicroseconds(10);
digitalWrite(trigPin , LOW);

durationN = pulseIn(echoPin , HIGH);
distanceN = durationN*0.034/2.0;
Serial.print(" Current Distance: ");
Serial.print(distanceN);
Serial.println("cm");

return distanceN;
}

float getAngle()
{
    sensors_vec_t    orientation;

    // Use the simple AHRS function to get the current orientation.
    if (ahrs.getOrientation(&orientation))
    {
        /* 'orientation' should have valid .roll and .pitch fields */
        Serial.print(F(" Orientation: "));
        Serial.print(orientation.roll);
        Serial.print(F(" "));
        Serial.print(orientation.pitch);
        Serial.print(F(" "));
        Serial.print(orientation.heading);
        Serial.println(F(""));
    }

    //delay(100);
    return orientation.pitch;
}
```


Appendix C

Consultation Meetings Attendance Form

Consultation Meetings Attendance Form

Week	Date	Comments (if applicable)	Student's Signature	Supervisor's Signature
1	03/03/16	Discuss mathematics behind the control system	<i>Shadell</i>	<i>[Signature]</i>
2	10/03/16	Discuss project specifications & aims	<i>Shadell</i>	<i>[Signature]</i>
3	16/03/16	Discuss sensors and testing methods	<i>Shadell</i>	<i>[Signature]</i>
4	24/03/16	Discuss sensor and equipment budget	<i>Shadell</i>	<i>[Signature]</i>
5	31/03/16	Sorting out room allocation issue	<i>Shadell</i>	<i>[Signature]</i>
7	14/04/16	Clean up and project demonstration	<i>Shadell</i>	<i>[Signature]</i>
8	19/04/16	CAD sensor mount & flow chart	<i>Shadell</i>	<i>[Signature]</i>
9	28/04/16	Flowchart & math and graphing techniques	<i>Shadell</i>	<i>[Signature]</i>
10	12/05/16	Discuss structure of document & drag calc.	<i>Shadell</i>	<i>[Signature]</i>
11	19/05/16	Discuss disruption to studies issue	<i>Shadell</i>	<i>[Signature]</i>
12	25/05/16	Discuss document preparation	<i>Shadell</i>	<i>[Signature]</i>
13	2/06/16	Discuss document preparation & drag measurement	<i>Shadell</i>	<i>[Signature]</i>

Bibliography

- [1] T. Benson, “Whirling arms and the first wind tunnels,” NASA, 2014. [Online]. Available: <https://www.grc.nasa.gov/www/k-12/WindTunnel/history.html>
- [2] R. A. Britcher, Collin P. Kilgore, *Magnetic Suspension and Balance Systems*, 1987.
- [3] G. Dimitriadis, *Experimental Aerodynamics*, 1st ed. Universite de Liege. [Online]. Available: <http://www.ltas-aea.ulg.ac.be/cms/uploads/Expaero01.pdf>
- [4] A. Donovan, H. Lawrence, F. Goddard, and R. Gilruth, *High Speed Problems of Aircrafts and Experimental Methods*. Princeton Legacy Library.
- [5] E. Freaks, *HC-SR04, Datasheet*.
- [6] S. D. Groote, “The Importance of Aerodynamics,” *F1 Technical*, 2006.
- [7] P. Inc., *Parallax Laser Range Finder 28044, Datasheet*.
- [8] —, *PING Ultrasonic Range Finder 28015, Datasheet*.
- [9] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, “Towards fully autonomous driving: Systems and algorithms,” in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, June 2011, pp. 163–168.
- [10] A. M. LLC, *Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Isolation and a Low-Resistance Current Conductor, Datasheet*.
- [11] P. W. Lucas, “Wind tunnel testing,” 2013.
- [12] P. L. l. Marie H. Tuttle, *Support Interference of Wind Tunnel Models*, 1984.
- [13] D. C. B. Millikan, “Purpose and Requirements,” *Engineering and Science*, 1945.
- [14] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, and B. e. a. Huhnke, “Junior: The stanford entry in the urban challenge,” *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.
- [15] D. Pomerleau and T. Jochem, “Rapidly adapting machine vision for automated vehicle steering,” *IEEE Intelligent Systems*, vol. 11, no. 2, pp. 19–27, 1996.

- [16] PulsedLight, *LIDAR-Lite v2 Overview, Datasheet*.
- [17] A. Sage, “Where’s the lane? Self-driving car confused by shabby U.S. roadways,” *Los Angeles*, 2016.
- [18] Seas, “Wind Tunnel,” *Formula1-dictionary.net*, 2016. [Online]. Available: http://www.formula1-dictionary.net/wind_tunnel.html
- [19] SHARP, *GP2D12, Datasheet*.
- [20] ST, *LSM9DS0, Datasheet*.
- [21] N. Todesco, “Control System For A Self-Driving Wid Tunnel Model,” 2014.
- [22] K. Townsend, *Adafruit BNO055 Absolute Orientation Sensor, Datasheet*.