# AUTONOMOUS AUTOMOTIVE WIND TUNNEL MODEL

Angus Buick

Bachelor of Engineering
Mechatronic

Department of Electronic Engineering
Macquarie University

December 4, 2017

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

## ACKNOWLEDGMENTS

I would like to acknowledge Dr. Sammy Diasinos, for his support, supervision and mentoring with this project. I would also like to acknowledge Dr. David Inglis for his enthusiasm and novel solutions for any problems I encountered. As well as Dr. Graham Town and Dr. Len Hamey for their wealth of knowledge on topics specific to the project.

## STATEMENT OF CANDIDATE

I, Angus Buick, 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 an any academic institution.

Student's Name: Angus Buick

Student's Signature: Angus Buick (electronic)

Date: 6/12/27

## ABSTRACT

A review of automotive wind tunnel testing today reveals that techniques are required that more closely replicate real world driving conditions. one such technique involves removing all external supporting structures from the model and having it "drive" by itself on a belt type system. This removal of supporting structures requires a control system to be developed that will keep a radio controlled car platform driving on the conveyor belt. The aim is to have the radio controlled car platform drive on the belt with high accuracy and to use measurements from the cars motor to determine drag coefficient. In order to do this a mathematical model of the system must be constructed and used to develop the control system. To determine the vehicles position relative to the center of the belt a computer vision algorithm is developed to track an identifying green rectangle mounted to the vehicle model. These control signals are then sent to the vehicle wireless through a bluetooth connection. The results reveal where the greatest delay in the system occurs, when position sensing and control derivation are optimized. Providing the necessary ground work to build a truly optimized system.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background and Motivation

### 1.1.1 Wind Tunnel Testing

Wind tunnels were developed towards the end of the 19th century during the early days of aeronautical research. They facilitate the flow of air around a stationary object, simulating the object moving through stationary air. This allows for various measurements to be taken on the moving air and the resulting forces produced by the object. The test section is where the object is placed and the majority of the measurements are taken. The object is then held in the air flow by a sting.

Stings are a fixture on which the models are mounted. For example, when doing F1 racing car wind tunnel testing the sting is attached to the model of the drivers helmet, or in rocket testing the sting is usually attached to the rocket engines where the exhaust plume would be. This fixture has an aerodynamic cost and introduces inaccuracies in the model due to it being a foreign object not attached to the object under normal circumstances. The elimination of these structures would result in more accurate testing of the object.

With this in mind it is easy to see why removal of these structures is desirable. One such method of doing so is proposed in this document. The proposed system is for automotive wind tunnel testing where it is common to simulate the ground being at the same velocity as the air, this is done with a *moving ground* or *moving belt* as its commonly called. The idea is to have the vehicle model "drive" itself with use of a control system, on the moving ground.

### 1.1.2 The Problem

Defining the problem more specifically we can say that we are concerned namely with the position of the car on the belt and keeping it center. Some errors that are important to formalize for the problem are:

- The cars lateral displacement error with respect to the center of the belt denoted as $e_1$.

- The cars angular error with respect to the *straight ahead* axis of the belt denoted as $e_2$
- The cars lateral velocity error with respect to the center of the belt denoted as $\dot{e}_1$.
- The cars angular velocity error with respect to the *straight ahead* axis of the belt denoted as $\dot{e}_2$
- The cars longitudinal displacement error with respect to the center of the belt.

# Chapter 2

# Background and Theory

## 2.1 Wind Tunnel Testing

### 2.1.1 Consumer Automotive Aerodynamics

Wind tunnels were originally developed for aerodynamic development in the aerospace industry and it was not until the 1970's that full scale, aerodynamic wind tunnels began to be designed and built for aerodynamic testing. Now days every major car manufacturer has multiple wind tunnels for testing and developing their products, because in order to remain competitive modern cars must be fuel efficient, safe and comfortable. All these issue are related, at least partially to aerodynamics. Another factor is that when companies are doing climatic testing they are limited to 2 seasons (summer and winter) to locally test in climate extremes. These time constraints pressure manufacturers. The advent of climatic wind tunnels allow for more flexibility during the design process. For example the same car could be tested at sea level and then at 4200m altitude in the same facility in the same day [12]. This flexibility has been more important since the evolution of passenger cars has become extremely fast due to increased competition between car companies [4].

### 2.1.2 Race Car Aerodynamics

Race cars rely heavily on the effectiveness of the aerodynamic package to produce down force necessary for keeping the car from losing traction during cornering. Optimization of these down force producing components (namely the wings, floor, diffuser barge boards and splitter plate), is extremely important for producing a competitive race car [6]. In order to assess the effectiveness of these components CFD (computational fluid dynamics) and wind tunnel testing are used to investigate force, flow structures, pressures and drag.

In order to hold an automotive model, stings are utilized which are a fixture upon which the model is mounted. An example can be seen in figure 2.1. This project aims to provide a way of eliminating these fixtures.

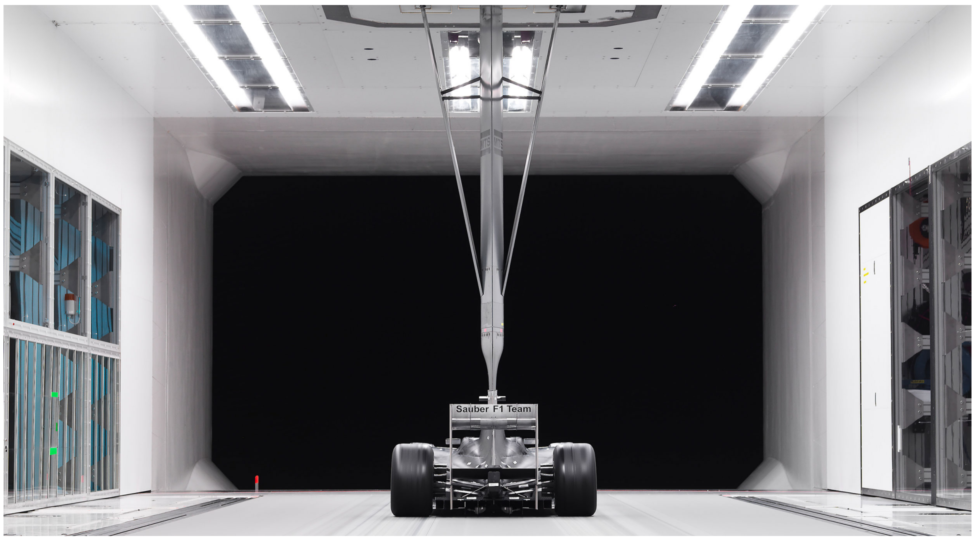


**Figure 2.1:** An example of a sting holding a scale F1 car model. [23]

### 2.1.3 Impact of Stings on Aerodynamics

When wind tunnel testing automotive vehicle models the most common configuration is to have a strut that attaches vertically to the car usually positioned on the highest point which is generally the roof on closed cars or the drivers helmet on open cars. This strut is assumed to have little aerodynamic affect but this is not always true. The introduction of this overhead support can affect the flow structures and therefore the results of the experiment. As shown in [3] when testing a race car with and without the overhead strut there was a difference in $cDa$ of $-0.016$ which is not insignifcant. This difference is generally larger in race cars because of the flow disruption for the rear wing, as compared to consumer cars where there is not as much affect due to no rear wing.

An example of the change of these flow structures has been investigated with the use of CFD in [6]. Figure 2.2 depicts the results of a CFD simulation that shows a change of flow structure around a wheel with a support structure and without. The main affect of the strut in this case due to the blockage was an increased flow into the side of the wheel.

### 2.1.4 Aerospace Stingless Aerodynamics

NASA (National Aeronautics and Space Administration) in conjunction with MIT (Massachusetts institute of technology) developed a stingless aerospace wind tunnel. By using extremely powerful electromagnets, a small model is able to be suspended in the test section "no strings attached". This technology is called MSBS (magnetic suspension and balance system), this example is a three dimensional case of what is aimed to be achieved with this project. The system without any interference from stings is able to model real world operating conditions accurately. The system uses deflection and changes in the field to measure the force and torque needed for assessment [7].

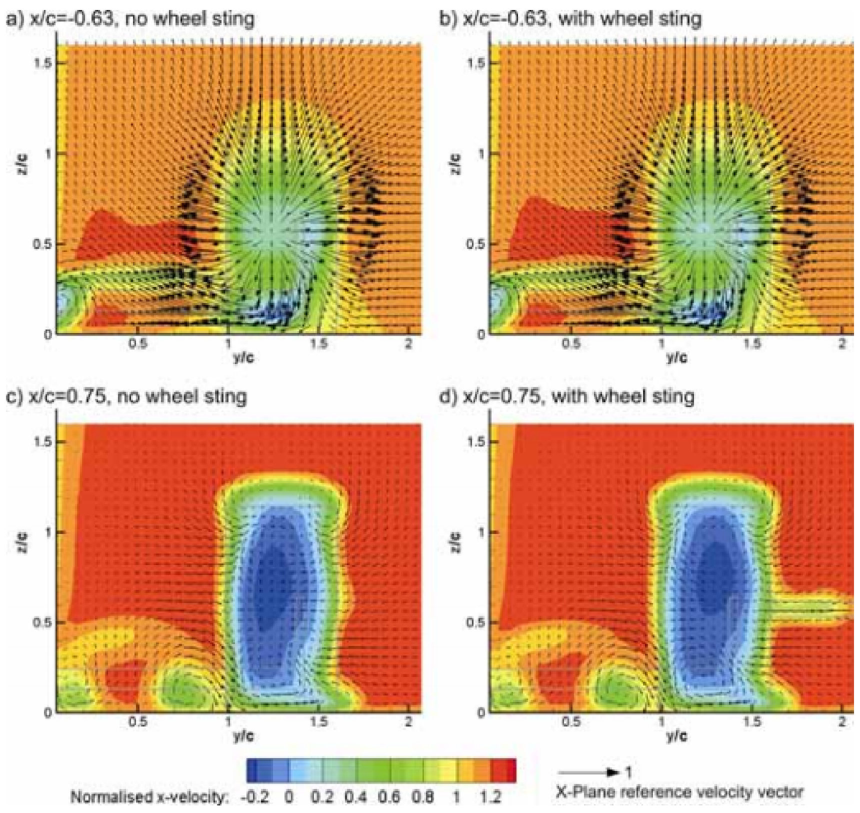


**Figure 2.2:** CFD fluid flow velocity vectors with supporting apparatus and without. [6]

## 2.2 Determination of Vehicles Position

### 2.2.1 Position Sensors

In order to determine the location of the vehicle, a positioning system must be designed. Many different systems could be employed for this task, each system will be assessed on their:

- Latency (lower is better)
- Accuracy (higher is better)
- Cost (lower is better)

Some sensors that could be employed are:

- Ultra sonic sensors: Side mounted to sense $e_1$ and derivative of $e_1$ could be used to determine $e_2$.
- Laser sensors: Side mounted to sense $e_1$ and derivative of $e_1$ could be used to determine $e_2$.

- Computer vision: Mounted for birds eye view to sense both $e_1$ and $e_2$.

Another distinction to be made is whether the processing is done on board via a micro-controller or off board and then transmitted to the vehicle. This distinction is particularly important for latency as well as ease of use due to the heavy modification of the model that is required for on board processing. Another problem with an on board type system is the sensing of the longitudinal position, this is due to there being no walls to sense due to the wind tunnel use case.

#### Ultrasonic sensors

An ultrasonic distance sensor emits an ultrasonic wave pulse and then receives a reflection of that wave, and by measuring the time this has taken, this is the same principal that bats use to "see" in the dark. The sensor produces this wave by vibrating a piezo-electric semi conductor [1].

This project was completed by a prior student that used ultrasonic sensors mounted on the side of the model to measure distance from the car to a side panel.

#### Laser Sensors

Laser sensors operate on the same time-of-flight principle as ultrasonic sensors sending out a pulse of light instead of sound, and measuring the time it takes for the reflection to return. Laser sensors have a wide variety of applications because of their wide variability of operating distances, large distances can be sensed for military applications like missile guidance. Or civilian applications like golf range finding. Or for small distances like 3D modeling tasks, where a hand held range finder constructs a 3D point mesh of the object. [2]

### 2.2.2 Computer Vision

Computer vision aims to gain higher level information from digital images and video similar how to the human visual system performs. Tasks like tracking a person against a moving background or facial recognition are made possible by computer vision. However despite these advances having a computer interpret an image at the same level as a 2 yr old child is still impossible even with today's sophistication. For example counting the number of animals in an image.

Computer vision is so difficult because it is an inverse problem in which we seek to recover some unknowns given insufficient information to fully specify the solution [22]. This means that forward models must be used to find potential solutions. However many solutions for various tasks have been solved, tasks like character recognition, motion capture, 3D modeling, and medical imaging.

### Thresholding

Thresholding is the process that turns a conventional image into a binary image. This is used to isolate the ROI (Region Of Interest). Where as a binary image is a digital image that has only two possible values 0, or 1 typically displayed where 0 is black and 1 is white. There are many possible ways to do this with the most basic being the simple threshold of a grayscale image. Figure 2.3 shows this where the leftmost image (being a linear progression from black to white of all grayscale values) being thresholded to the right most binary image. If the parameter were adjusted the point at which the image transitions from black to white would alter. [10]

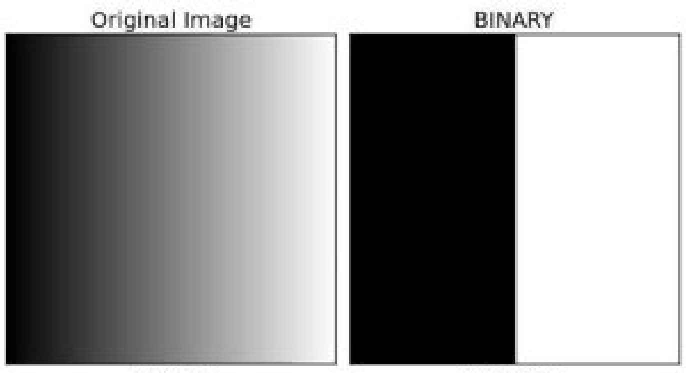


**Figure 2.3:** Threshold of grayscale image. [10]

Images can be thresholded for color as well. Depending on what color space the image file is using there are different methods. If the image is using HSV (Hue, Saturation, Value) color space which is easy to visualize using Figure 2.4. We can specify a plane or two that cuts through that color space to segment it, meaning any values on one side become ones and any on the other side become zeros [13].

### Hough Circle Transform

The circle hough transform is an algorithm used for feature extraction in computer vision. Specifically, circles. It does this by drawing a circle of a known radius on each white pixel of the binary image, where these circle intersect reveals the center. This is described as a voting process where as the highest voted pixel is discerned as the center pixel. This can understood with the help of figure 2.5. [18]

### Image Moments

Image moments work much the same way as their physics counterparts. Where as in physics a moment is the measure of the shape of a set of points. In image processing an image moment is the measure of the shape of a set of pixels. Useful information can be discovered in a computationally efficient way using this technique. It is particularly simple when the image is binary. To calculate the moments of an image equation 2.1

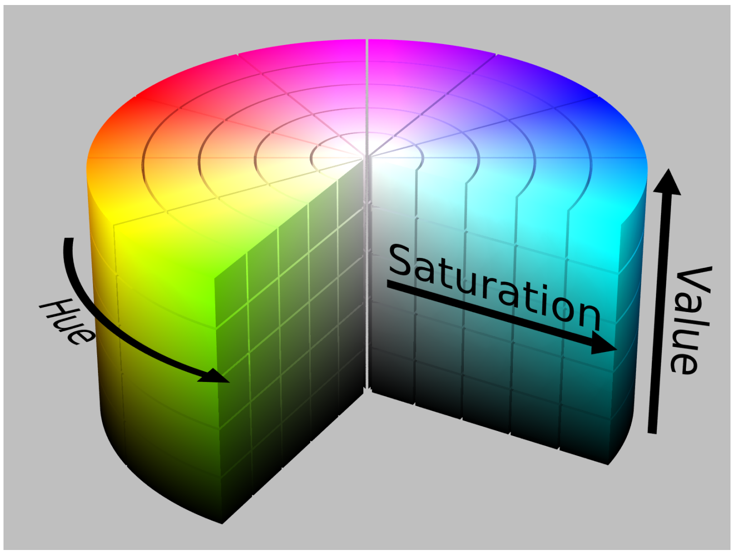


**Figure 2.4:** HSV colorspace. [20]

is used where the order of the moment is m+n. The *zeroth* order moment simplifies to equation 2.2 and 2.3. Now in a binary image each pixel is either 0 or 1 so for every white pixel, 1 is added to the moment, this calculates the area of the image.

Another valuable parameter that can easily be calculated using moments is the centroid. That is the area *center of mass* equivalent. To calculate the centroid of a binary image you need to calculate two coordinates the center of x and center of y. This is done by summing the $x$ coordinates of all white pixels and the summing the $y$ coordinates of all white pixels. Then to find the average, dividing by the area which is also the zeroth moment. This process is contained in equation 2.6. [11]

$$\mu_{m,n} = \sum_{x=0}^{\infty}\sum_{y=0}^{\infty} (x - c_x)^m (y - c_y)^n f(x,y) \tag{2.1}$$

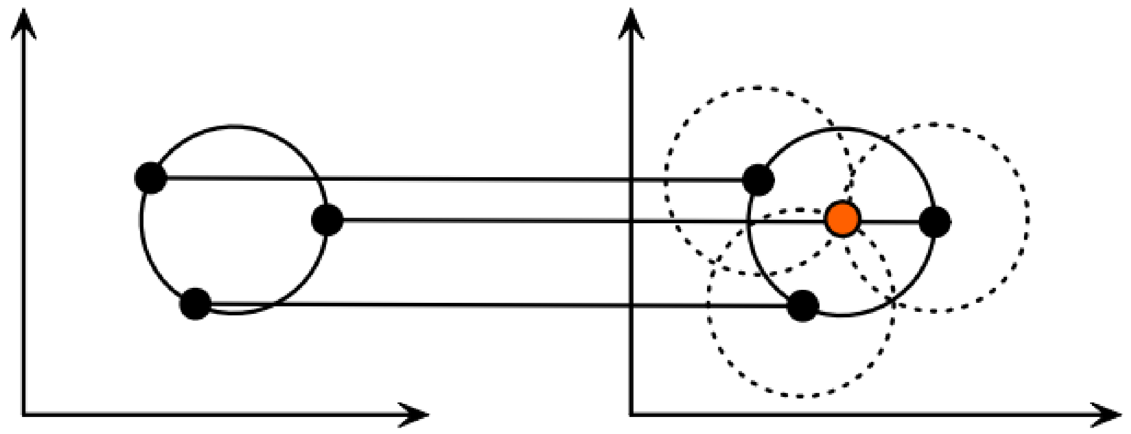
$$sum_x = \sum\sum \; xf(x,y) \tag{2.2}$$

$$sum_y = \sum\sum \; yf(x,y) \tag{2.3}$$

$$\mu_{0,1} = \frac{sum_x}{\mu_{0,0}} \tag{2.4}$$

$$\mu_{0,1} = \frac{sum_x}{\mu_{0,0}} \tag{2.5}$$

$$centroid = \left(\frac{\mu_{1,0}}{\mu_{0,0}}, \frac{\mu_{0,1}}{\mu_{0,0}}\right) \tag{2.6}$$



Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the $(a, b)$ that is the center in geometric space.

**Figure 2.5:** Hough Circle transform. [18]

## 2.3 System Models and Control Algorithms

### 2.3.1 Kinematic Model

#### Derivation

The system can be simplified and considered to be 2 dimensional, and moving about a plane. This first model is based purely on the geometric relationships that govern the system [16]. The system can simplified by making a two wheeled "bicycle" version of the system the kinematics can be seen in figure 2.7. A depiction of this model relative to the four wheel system is found at 2.6 to make this easier to understand. The bicycle model can be assumed to be true when Ackerman steering the four wheel system at low velocity where sliding does not occur. [19]

This model has these simplifications:

- Velocity at center of gravity is considered to be constant along the longitude of its trajectory.
- Lifting, rolling and pitching forces will be neglected.
- Vehicles mass only at COG.
- The front and the rear tires will be represented as one single tire on each axle.
- Torque from tire slip angle will be ignored. [19]

The equations of motion for this system are given by,

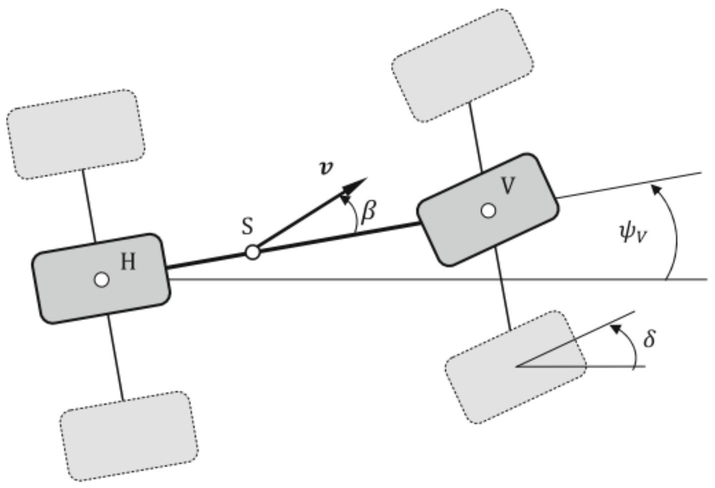


**Figure 2.6:** bicycle model [19]

$$\dot{X} = V\cos(\psi + \beta) \tag{2.7}$$
$$\dot{Y} = V\sin(\psi + \beta) \tag{2.8}$$
$$\dot{\psi} = \frac{V\cos(\beta)}{\ell_f + \ell_r}(\tan(\delta_f) - \tan(\delta_r)) \tag{2.9}$$
$$\beta = \arctan\left(\frac{\ell_f \tan\delta_r + \ell_r \tan\delta_f}{\ell_f + \ell_f}\right) \tag{2.10}$$

### 2.3.2 State-Space Model

#### Derivation

At higher speeds the assumption that individual wheel velocity is in the direction of the wheel (no-slip) can not be made. For this a dynamic model must be developed [16]. This model inherits many of the same parameters shown in figure 2.7. This model can be written in a state space representation as follows:

$$\frac{d}{dt}\begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mV_x} & 0 & -V_x - \frac{2C_{\alpha f}\ell_f + 2C_{\alpha r}\ell_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2\ell_f C_{\alpha f} - 2\ell_r C_{\alpha r}}{I_z V_x} & 0 & -\frac{2\ell_f^2 C_{\alpha f} - 2\ell_r^2 C_{\alpha r}}{I_z V_x} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2\ell_f C_{\alpha f}}{I_z} \end{bmatrix} \delta \tag{2.11}$$

The parameters of which are (the rest are inherited from the previous bicycle model):

- $C_{\alpha f}$ is the cornering stiffness of each front tire

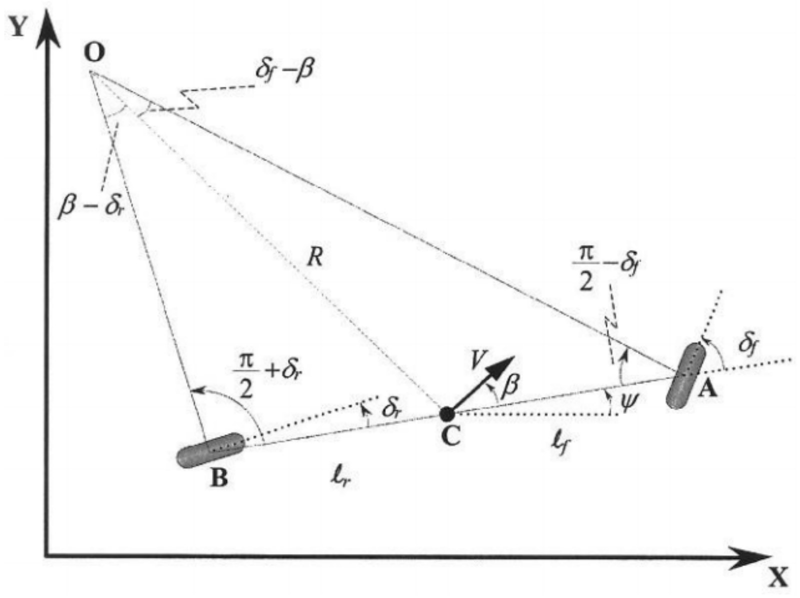


**Figure 2.7:** Kinematics of bicycle model [16]

- $C_{\alpha r}$ is the cornering stiffness of each rear tire
- $\delta$ is the steering angle.
- The longitudinal velocity of the vehicle at CG is denoted by $V_x$.

This model take in account the slip angle of the tires and the cornering stiffness of the vehicle and is thus more representative of the actual vehicle. This model can be made even more useful for this specific task by some reassignment of variables. The two following error variables:

- $e_1$ will be the distance of the CG of the vehicle from the center of the track.
- $e_2$ will be the orientation error (angle) with respect to the belt.

These two values are what we are trying to make zero in order to have the model drive straight at the center of the belt. The state space model can be re-written for the state space vector 2.12

$$\begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} \tag{2.12}$$

This is done with equations 2.13 and 2.14

$$\ddot{e}_1 = \ddot{y} + V_x(\dot{\psi} - \psi_{des}) \tag{2.13}$$

$$e_2 = (\psi - \psi_{des}) \tag{2.14}$$

and the entire system can be written terms of the new state vector as equations 2.15

$$\frac{d}{dt}\begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mV_x} & 0 & \frac{2C_{\alpha f}+2C_{\alpha r}}{m} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2\ell_f C_{\alpha f}-2\ell_r C_{\alpha r}}{I_z V_x} & \frac{2\ell_f C_{\alpha f}-2\ell_r C_{\alpha r}}{I_z} & -\frac{2\ell_f^2 C_{\alpha f}+2\ell_r^2 C_{\alpha r}}{I_z V_x} \end{bmatrix} \begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2\ell_f C_{\alpha f}}{I_z} \end{bmatrix} \delta + \begin{bmatrix} 0 \\ -\frac{2\ell_f C_{\alpha f}-2\ell_r C_{\alpha r}}{mV_x} \\ 0 \\ -\frac{2\ell_f^2 C_{\alpha f}+2\ell_r^2 C_{\alpha r}}{I_z V_x} \end{bmatrix} \tag{2.15}$$

This complete state space model of the system can be used for more accurate simulation of the system. and more advanced MIMO algorithms like state variable feedback that take into account multiple error signals rather than just one.

### 2.3.3 Vehicle Control

In order to control this vehicle on the moving ground, a control system will have to be constructed. Conveniently a vast amount of research has been conducted on autonomous car control systems. Cars as a vehicle are an under-actuated system meaning that there are less actuators than degrees of freedom. The classic approach for trajectory following (what is aimed to be solved with this project) on under actuated vehicles uses local linearization and decoupling of the multivariate model to steer the same number of degrees of freedom as the number of control inputs (throttle, steering). This can be done using classical linear control, these control methods include PID (Proportional Integral Derivative) and LQR (Linear Quadratic Gaussian) these are well known and well documented control methods. Their disadvantages are that stability is only guaranteed around the operating point and that if the vehicle performs a movement that accentuates its non-linearity, then control may breakdown.

- Proportional Integral Derivative (PID) - The proportional, integral derivative control system is the most widely used control method in industry. Thus it is well studied in its applications to a large amount of problems as well as autonomous car control. PID control can be attributed to Nicholas Minorsky, a Russian born in 1885, it was his theory and application to the automatic steering of ships that led to what we now know as PID control [15].

- Optimal Control; The LQR (linear quadratic regulator) control method operates by minimizing a cost function. That is applied to a linearized state space model of the system of the form: [9]

  $$\dot{x} = (A - BK)x + BKx_{desired}$$

  This control method is a MIMO (multiple input multiple output) control method which means it allows control of $n$ inputs and outputs, it does so via the state space matrix. The algorithm aims to minimize the deviations from the required outputs, therefore finding the *optimal* solution. The cost function however is human specified and is therefore not always representative of the best output.

- Intelligent systems; Recently systems such as machine learning and fuzzy logic controllers, have been implemented for autonomous vehicles that enable them to take in real world surroundings and produce path plans. A genetic algorithm is a type of searching algorithm that searches a solution space for an optimal solution to a problem. It works by creating a population of possible solutions, each achieves a fitness score, usually the higher the better. Each generation is mutated and bred into the existing population, selecting for the best scores. In this way the possible solutions *evolve* through a process of natural selection where the weakest with the worst fitness scores are not selected for *breeding*, whilst the fittest are selected for and survive. [17]

### 2.3.4 Pure Pursuit Control

The pure pursuit algorithm was developed in 1992 for use when tracking a planned trajectory of early autonomous vehicles. Namely the "NavLab" an extraordinary full size automotive autonomous vehicle that could accurately follow a road at high speed using a computer vision based system that analyzed each frame using a neural network image classifier that decided on a straight or curved trajectory and then used pure pursuit control to control the vehicle relative to that trajectory. It was extremely sophisticated for its time. [5]

The algorithm works by calculating a curvature that will move a vehicle from its current position to some goal position which is a specific distance *ahead* of the vehicle [5]. The process is repeated always looking forward a specific distance and steering to get to that point. This is very similar to way that humans drive looking ahead of the road and pointing the vehicle where we desire it to be. The derivation of this control method is simple and is described as follows.

The goal point (gx, gy) is illustrated in Figure 2.8. The vehicles steering angle $\delta$ can be determined using only the goal point location and the angle $\alpha$ between the vehicles heading vector and the look-ahead vector. Applying the law of sines to Figure 2.8 results in: [21]

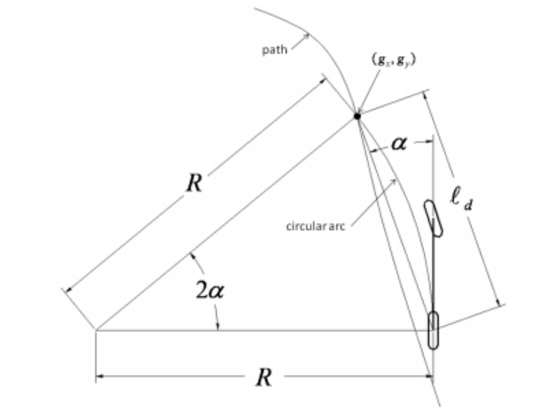


Figure 10: Pure Pursuit geometry

**Figure 2.8:** Pure pursuit geometry. [21]

$$\frac{l_d}{sin(2\alpha)} = \frac{R}{sin(\frac{\pi}{2} - \alpha)}$$
$$\frac{l_d}{2sin(\alpha)cos(\alpha)} = \frac{R}{cos(\alpha}$$
$$\frac{l_d}{sin(\alpha)} = 2R$$
$$k = \frac{l_d}{sin(\alpha)}$$

Then assuming Ackerman steering, the kinematic model steering angle from section 2.3.1 can be written as,

$$\delta = tan^{-1}(kL)$$

substituting in, we get the pure pursuit control law:

$$\delta = tan^{-1}\left(\frac{2Lsin(\alpha)}{l_d}\right)$$

To be implemented in a digital system this algorithm works as follows [5]:

1. Determine the location of vehicle.
2. Find the path point closest to the vehicle.

3. Find the goal point.
4. Transform the goal point to vehicle coordinates.
5. calculate the curvature and request the vehicle to set the steering to that curvature.
6. update the vehicles position.

## 2.4 Transmission of Control Signals

### 2.4.1 Radio Control

Radio control was first demonstrated by Nikola Tesla in 1898 at the first electrical exhibition in Madison Square Garden [24]. However during the 1990's the technology became miniaturized and widely available. It was used mainly to control amateur models. Modernly RC technology is still widely used for amateur model control and persists in many military and aerospace technology such as using UHF radio communication for the mars exploration rover.

The most common type of RC is where a pulse for that channel is varied between $920\mu s$ and $2120\mu s$ that is representative of a 0-255 PWM signal. Depending on frequency the latency of a system like this can be between $10ms$ and $30ms$. Although modern alternatives exist utilizing pulse code modulation (PCM) and a $2.4GHz$ carrier wavelength that can achieve latencies of $3ms - 6ms$.

### 2.4.2 Bluetooth

Bluetooth is a wireless short range communication protocol that is used all over the world for low-cost, low-power communication systems. It was designed in 1989 by Nils Rydbeck the CTO of Ericsson mobile at the time for use in wireless headsets [8].

It operates at frequencies between 2400 and 2483.5 MHz and depending on the class and power consumption it has a range that varies from 0.5m to 100m. The maximum theoretical asymmetric (only one device is transmitting) data rate of bluetooth devices is 723kb/s.

## 2.5 Previous Attempts

In 2016 an attempt was made at this problem in [14] the student solution used ultrasonic sensors that measured the distance from a side plate to determine the vehicles position, and for control the student used a unique arc combining system. If the cars angle was pointing away from from the target line it would require two arcs to be drawn. These two angles where determined geometrically. Figure 2.9 assists in visualization of this concept. Once the model was deemed in a "tolerance range" the arc control method relinquished control and another control method based on a matrix of nine light dependent resistors

(LDR) took over that measured the cars position with respect to a laser that shone at the center of the belt.

Since the computation was then done on board the vehicle model itself transmission of the control signals was not needed. Although bluetooth was used for basic control of the model. This attempt proved successful in the sense that it achieved a self driving vehicle model on the belt however it was only stable at low speed and tended to drift of the center an unsatisfactory amount.

This attempts main downfall was the control method as it has been demonstrated already in [5] that the two arc control method attempted is inherently unstable, a more stable method now called pure pursuit was developed in that paper. The LDR array attempted to remedy this but added complexity to the system and required fine control not provided by the resistors.

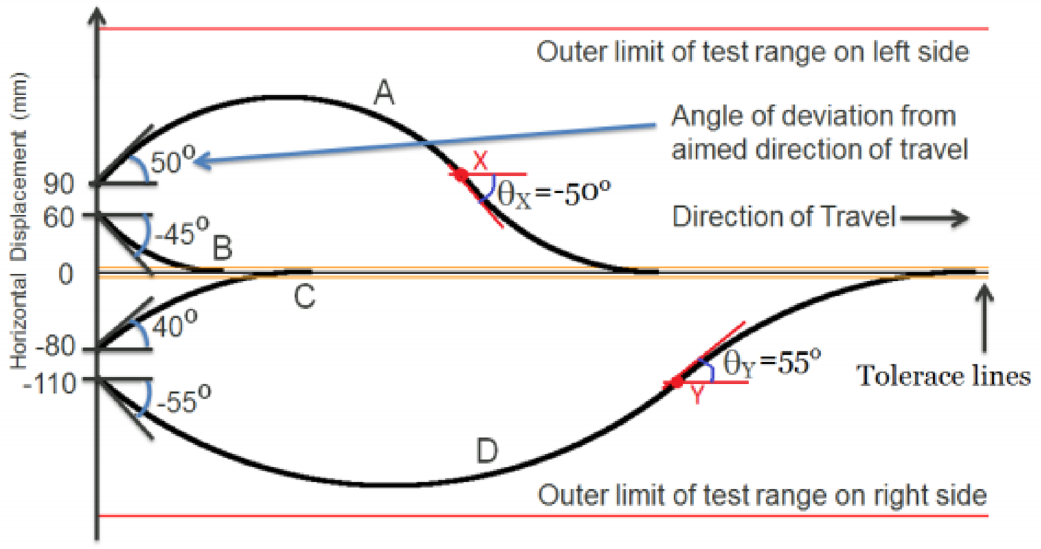

Horizontal Displacement (mm)
90
60
0
-80
-110
Outer limit of test range on left side
A
Angle of deviation from aimed direction of travel
$50^o$
X
$\theta_X=-50^o$
$-45^o$
B
Direction of Travel
C
$40^o$
$\theta_Y=55^o$
Y
Tolerace lines
$-55^o$
D
Outer limit of test range on right side

**Figure 2.9:** A previous students control method based on arcs. [14]

# Chapter 3

# Methodology

## 3.1 Software

### 3.1.1 Python

The Python Programming Language is a high level general purpose language that has become extremely popular because of its readable and concise syntax. It was chosen to write the main program in because of the ability to use many different libraries that make complicated tasks like machine learning, graphing or in this case computer vision much more manageable with the ability to call functions from these libraries.

### 3.1.2 OpenCV Library

The open source computer vision (openCV) library is a library that can be implemented with the C++ and Python languages, it has over 2500 algorithms that can be used for a variety of computer vision based tasks such as facial recognition, object tracking and object recognition. In the main program this library and some of its functions where utilized in order to achieve the functionality required.

### 3.1.3 MatLab and Simulink

MatLab is a programming language that was used to model the system. Simulink is a graphical programming tool that was developed by MathWorks for modeling and simulation. It was used in this project to model the car and develop control system. It is capable of modeling dynamic non-linear systems not usually easily modeled.

## 3.2 System Modeling and Control

### 3.2.1 Kinematic Model

The equations of motion for this system are given by,

$$\dot{X} = V\cos(\psi + \beta) \tag{3.1}$$

$$\dot{Y} = V\sin(\psi + \beta) \tag{3.2}$$

$$\dot{\psi} = \frac{V\cos(\beta)}{\ell_f + \ell_r}(\tan(\delta_f) - \tan(\delta_r)) \tag{3.3}$$

$$\beta = \arctan\left(\frac{\ell_f \tan\delta_r + \ell_r \tan\delta_f}{\ell_f + \ell_f}\right) \tag{3.4}$$

Using these equations of motion, a Simulink model was constructed that can be seen in figure 3.1 This model takes a vehicle velocity in m/s and steering angle in radians and outputs planar X and Y coordinates in cm for the vehicles center of mass using Runge-Kutta methods with a fixed time step of 0.0001s.

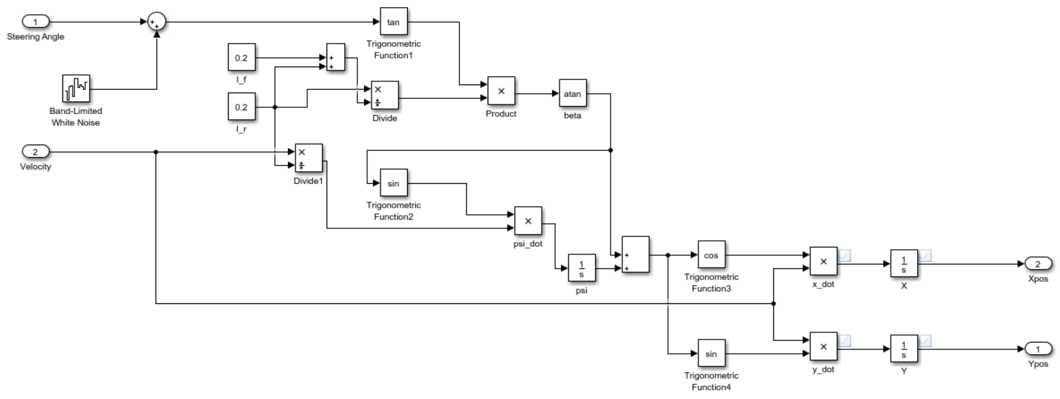


**Figure 3.1:** Simulink model.

### PID Control

To control the kinematic model, a simple PID controller was implemented, where the error signal is the distance from the x-axis. The system was linearized around a steering angle of zero. The Simulink model of simulation can be found in figure 3.1. Note that figure 4.1 is contained within the subsystem labeled ”2D car model(no slip)”.

## 3.3 Design Choices

### 3.3.1 Overview

The system that was decided is as follows:

- An RC car operates as the model.

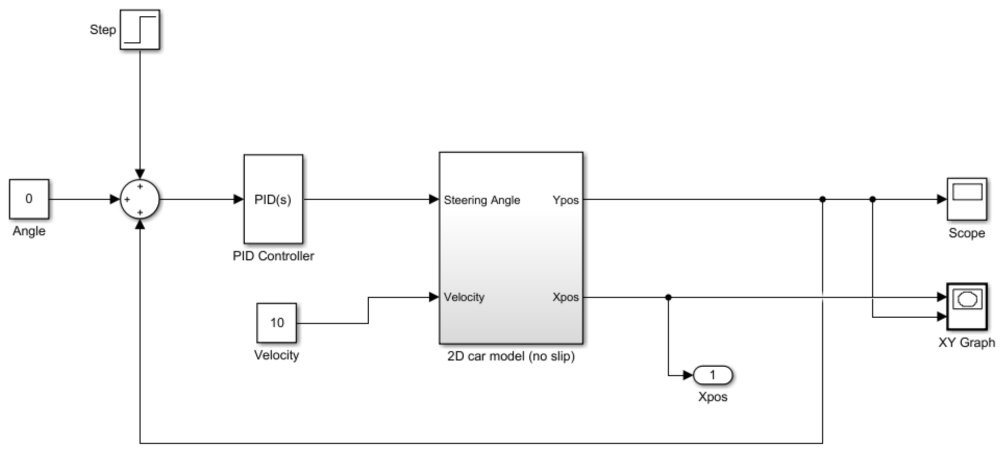


**Figure 3.2:** Simulink model of PID controlled system.

- A *birds eye view* camera and a computer vision algorithm acts as the position sensor.
- PID acts as the control method for the lateral and longitudinal errors. Although pure pursuit and pole placement based control methods were developed but not fully functional.
- Bluetooth is used as the transmission method. This signal is sent from the main computer to the car and processed with an Arduino that control the steering servo and ESC.

A photo of the apparatus is show in figure 3.3.

### 3.3.2 Vehicle Model

The requirements of the test vehicle platform are met easily by a road-going RC car. An RC car model has been purchased by the previous student who completed this project. It is powered by 3 cell lithium polymer battery, that is connected to a Leopard V2 ESC (electronic speed controller) that transforms the DC current into three AC phase signals for use by the permanent magnet DC motor that through a nylon gear drives the model. The ESC's power level is controlled by a PWM signal.

The steering for the model is controlled by a Hitech HS422 servo motor, this servo motor receives a PWM input between 0 and 180 and adjusts its position between 0 and 180 degrees, however due to the mechanical linkages that it is connected to the wheels by, (that can be found in figure 3.4) its range of motion is limited to between 30 and 150 degrees.

The chassis of the model has been cut out of carbon fiber and it features grippy foam tires and suspension both front and rear.

### 3.3.3 Moving Ground

The moving ground for the system has been supplied by the department and is a small treadmill. It has basic control over the speed that the belt rotates. A picture of the belt can be found in figure 3.3.

### 3.3.4 Vehicle Positioning System

#### Camera

The requirements for the ideal camera for the system are:

- High frame rate, because the frame rate introduces a delay in the system.
- Low resolution, for this use case more pixels means longer operations when analyzing each frame because each pixel must be operated on. However we can have a resolution that would mean low accuracy in the positioning system but this is likely lower than most cameras resolution.
- Digital, an analogue camera would require digitizing so that it could be analyzed, introducing complexity and an extra computational cost.
- Global shutter, a takes all of the cameras sensor information at once rather than having a rolling shutter that takes it one line at a time which leads to distortion in anything moving.

It was decided that for a proof of concept system to be constructed that a web camera would be used, the camera selected is a Microsoft "LifeCam HD 3000" this is because whilst it does not have a high frame rate or good optics it is cheap, easily interfaceable (USB 2.0), and because the latencies of other subsystems arent known it was deemed unwise to invest in an expensive industrial camera and lens when the concept was not proved.

The 33.3$ms$ latency this camera introduces into the system means that the stability of the model will break down at a lower speed than if a higher frame rate camera was used because control signals will not reach the car as frequently as necessary at high speed. Some industrial cameras exist with very high frame rates, global shutters, and very low communication latency that are essentially designed for computer vision tasks like this. Some suitable cameras are listed below:

- Basler ace acA640-750uc
  - Resolution: 640 $\times$ 480
  - Frame rate: 751 fps
  - Interface: USB 3.0
  - Dimensions (L$\times$W$\times$H): 29.3mm $\times$ 29mm $\times$ 29mm

- Allied Vision Mako U-029
  - Resolution: 640 × 480
  - Frame rate: 550 fps
  - Interface: USB 3.0
  - Dimensions (L×W×H): 49.5mm × 29mm × 29mm

### Computer

The requirements of the computer are that it has bluetooth, WiFi and the necessary computational power in order to run the frame analysis fast enough to not add to much additional latency. For this proof of concept system, the student's Microsoft Surface Pro 4 was used for running the software and sending the control signals via bluetooth or Wifi. As it has bluetooth and WiFi adapters inbuilt.

### PID Algorithm

The computer vision has been programmed with the Python language utilizing the openCV computer vision library. The full code can be found in appendix A.1. The algorithms functionality is quite simple and is described below:

1. The program imports the openCV, math, pyBluez and numPy libraries.
2. The bluetooth connection is setup with the pyBluez library. This is done by inputting the HC-05 bluetooth modules MAC address (98:D3:34:91:19:9A) and the type of bluetooth protocol that is desired (RFCOMM).
3. The video stream is loaded with the `cv2.VideoCapture()` function for use with the Microsoft surface the external camera is number 3 since it already has 2 inbuilt web cams.
4. Next a while loop is entered that reads each frame from the camera and begins analysis on it. Line 61 `_, frame = cap.read()` reads the individual frame from the video stream and stores the image it in the variable frame.
5. In order to perform the HSV color threshold the color format of the file must be in the HSV color format and must be converted this is done on line 62 with the `cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)` this takes the frame variable and converts the color format from RGB to HSV.
6. To threshold the image a openCV thresholding function called through `cv2.inRange()` that take HSV variables that we wish to select between, in this case Hue Saturation and Value variables of 30, 50 and 255 respectively and 180, 255, 255 that are defined in the preamble in the numPy arrays `lower_green` and `upper_green`. This function returns a binary image that has 1's for all values in the range and 0's for all values

not in the range. Figure 3.5 shows how the green rectangle becomes isolated once the threshold is preformed.

7. Next the moments are found. Again an openCV operator is called `cv2.moments()` this function returns the first seven moments of the image. Then to determine the values we are concerned about we must use some formulas that were mentioned in section 2.2.2. The Syntax used is below

    ```
    M = cv2.moments(mask)
    cx = int(M['m10'] / M['m00'])
    cy = int(M['m01'] / M['m00'])
    a = int(M['m20'] / M['m00'] - cx ** 2)
    b = int(2 * (M['m11'] / M['m00'] - cx * cy))
    c = M['m02'] / M['m00'] - cy ** 2

    theta = 1 / 2 * math.atan(b / (a - c)) + (a < c) * math.pi / 2
    thetaDeg = round(math.degrees(theta), 3)
    ```

    The variables returned by the moments function are accessed by there order for example if we wanted to access the zeroth moment stored in the variable `M` we would call `M['m00']` at the end of these lines we have the X and Y pixel coordinates of the green rectangle *centre of mass* and its angle with respect to vertical frame.

8. To determine $\dot{e_1}$ and $\dot{e_2}$ we must determine the rate at which these X and Y variables are changing. To do this the previous variables of X, Y and theta are stored and since we know the time between frames (due to the frame rate being 30fps) we can then calculate the velocity and angular velocity that green rectangle is moving with. The code that achieves this can be found below,

    ```
    velX = round((prevX - cx) * pixelDistance / 0.333, 3)
    velY = round((prevY - cy) * pixelDistance / 0.333, 3)
    velTheta = round((prevTheta - thetaDeg) / 0.333, 3)
    ```

    The function `round()` rounds whatever the number generated to a specific number of decimal points in this case 3 was specified for computational efficiency and display purposes. The variable `pixelDistance` is a distance in $mm$ that one pixel covers. This was found by measuring the number of pixels that the green rectangles long side covers, then dividing by $100mm$ (because the green rectangle is $100mm$ long).

9. To obtain the errors with respect to the center of the belt we subtract half of the frames width and height from X and Y respectively, since the frames resolution is $480 \times 640$ the next two lines of code store the center of mass's displacement error from the center.

```
    xError = round((cx - 320) * pixelDistance, 4)
    yError = round((cy - 240) * pixelDistance, 4)
```

10. Next to attain the control variables that are needed (throttle and steering) a PID loop is applied to `xError` and `yError`. This is a conventionally applied discrete PID loop. That has overshoot control, scaling and anti-windup though the use of custom functions `limitvalue` and `scaleclipped`. The full code can be found in appendix A.1.

11. For easy prototyping of this algorithm the errors and control signals have been written onto the frame so that it can be displayed real time for easy debugging. The function `cv2.putText()` overlays text onto the frame in a specific font, position and color. When run live it shows the camera stream with the variables overwritten. An example of a frame is given in Figure 3.6

12. Lastly the control variables are sent wireless using the `sock.send(toSend)` function.

This code can be exit by pressing the escape key that will terminate the bluetooth and camera streams.

### Hough Circle Algorithm

This algorithm was an early prototype that uses the hough circle transform to locate the model. It functions on a different tracking method. Its functionality is described below

1. The program imports the openCV and numPy libraries.

2. The grayscale video stream is loaded from the file
   `'output1.avi'` with the `cv2.VideoCapture()` function.

3. Next a while loop is entered that reads each frame from the camera and begins analysis on it. Line 11 `_, frame = cap.read()` reads the individual frame from the video stream and stores the image it in the variable frame.

4. A simple threshold is performed, segmenting the image. The threshold accepts values from 220-255.

5. Next the openCV function, `cv2.HoughCircles()` takes an input file which is the binary image resulting from the threshold and based on some input variables like minimum and maximum radius its stores an array of the radius and center locations of each circle detected.

   ```
   circles =cv2.HoughCircles(thresh, cv2.HOUGH_GRADIENT, 1, 20, param1=50,
   param2=14,minRadius=30,maxRadius=70)
   ```

6. To then display these circles a for loop is run for the number of circles in the array, drawing the circle its self and then the centers. The following code achieves this,

```
for i in circles[0,:]:
        # draw the outer circle
        cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
        # draw the center of the circle
        cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)
```

7. The frame is then displayed with the openCV function `cv2.imshow('frame',cimg)`

An example of a resulting frame from this video stream can be seen in figure 3.7. It depicts the circles drawn over the circular markers placed on the model. The full code can be found in appendix A.4

### 3.3.5 Transmission System

The control signal transmission system is split into to two parts, on one side there is the computer that has analyzed the frame and decided on some control output which then must send these signals to the other part, the model side. This side must receive the inputs and use them to control the motor ESC and steering servo motor.

This connection has been established using bluetooth radio frequency communication (RFCOMM) this protocol allows the wireless simulation of a serial port that can be used to send signals from the computer to the model side where there is an HC-05 bluetooth module that allows an Arduino to receive the signals and output them too the ESC and servo.

In order to enable to functionality in the python program the PyBluez library was used that allows for simple connection and use of bluetooth devices. The three main functions that are called upon from the PyBluez library are the:

- socket.connect(), function that allows connection to a device with a specific MAC address with a specific bluetooth protocol, in this case "RFCOMM".
- socket.send(), allows a string to be sent over this connection.
- socket.close(), terminates the connection once it is done.

A more detailed walk through can be had in section 3.3.4. On the model side the HC-05 bluetooth module receives the strings that are sent through the bluetooth connection as single byte format ASCII characters. The HC-05 is connected to an Arduino through the 10 and 11 pins that are used to establish a serial connection using the "SoftwareSerial.h" library, where the characters are then read, concatenated from there single characters into the double or triple digit control signals. Then using the "Servo.h" library a PWM output is written to the 3 and 6 pins to control the ESC and servo respectively.

A more detailed walk through can be had in section 3.3.4. On the model side the HC-05 bluetooth module receives the strings that are sent through the bluetooth connection as single byte format ASCII characters. And the Arduino program works in the following way,

1. The program establishes a serial connection over the two pins 10 and 11. This is done using the software serial library with the command,

   ```
   SoftwareSerial blueSerial(10, 11);
   ```

2. The program sets up the two serial communication ports with the required baud rate of 9600. As well as attaching the two PWM output pins 3 and 6 with the code,

   ```
   steer.attach(3);
   throttle.attach(6);
   ```

3. Next the program enters the loop checking every loop whether something has come over the bluetooth serial connection. Once a byte arrives it enters a perpetual while loop concatenating the incoming bytes until the delimiting character arrives that is specified as ',' at which point the program writes this value to the steering servo and breaks out of the loop.

   ```
   while(1) {            // force into a loop until ',' is received
       incomingByte = blueSerial.read();

       if (incomingByte == -1) continue;

       if (incomingByte == ',') {
           Serial.print(sC + ", ");
           steer.write(toInt(sC));
           break;
       }
       sC += String(incomingByte);
     }
   ```

4. Then the program repeats this in another loop for the next values until the next delimiting character arrives specified as '\n' the ASCII character for a new line. At which point it writes the value to the ESC and the process is repeated.

## 3.4 Summary of Delays in the System

For the system latencies can be summarized in figure 3.8. With the latency at each step estimated as:

- Camera: latency determined by frame rate.
- Signal transmission from camera to computer: $\mu s$ scale through USB 3.0.
- Processing: Dependent on algorithm and hardware employed can be greatly decreased with optimization and use of special hardware (FPGA/graphics card)

- Signal transition from computer to transmitting device: $\mu s$ scale through USB 3.0.
- Transmission: Dependent on method conventional RC can be as low 3ms and as high as 15ms.
- Actuation: Dependent on $\Delta\theta$ and servo dynamics, however $< 1ms$.
- Movement of car: Continuous.

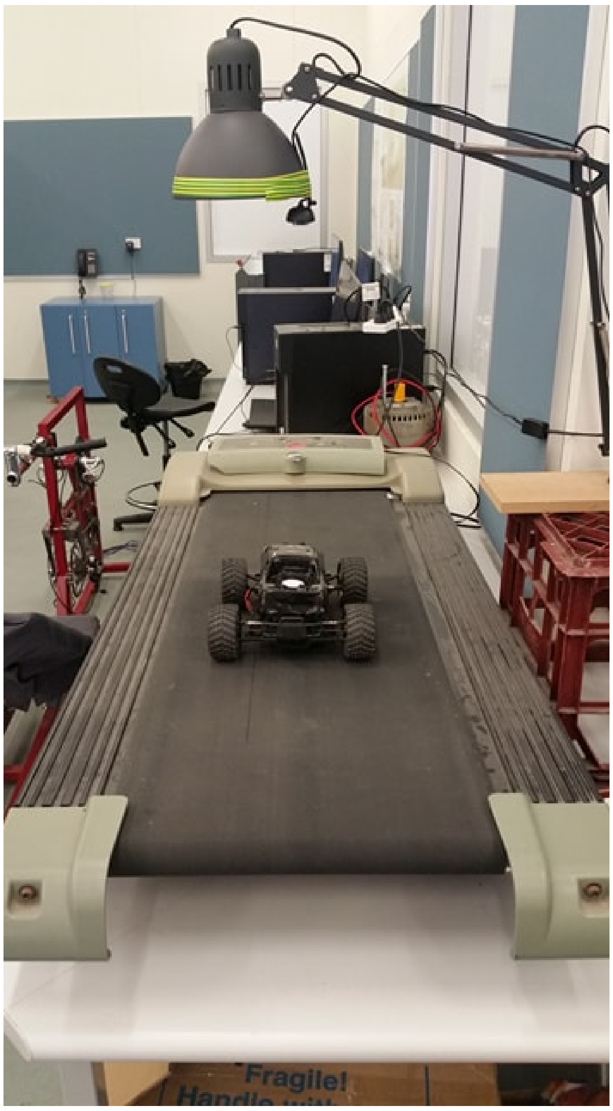


**Figure 3.3:** The moving ground, vehicle model, camera mount and camera apparatus.

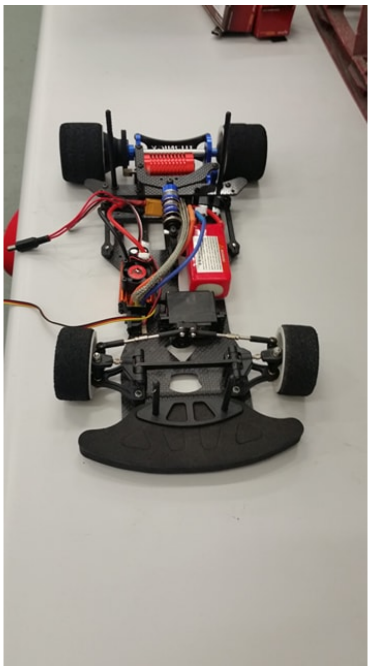


**Figure 3.4:** The RC car vehicle model.

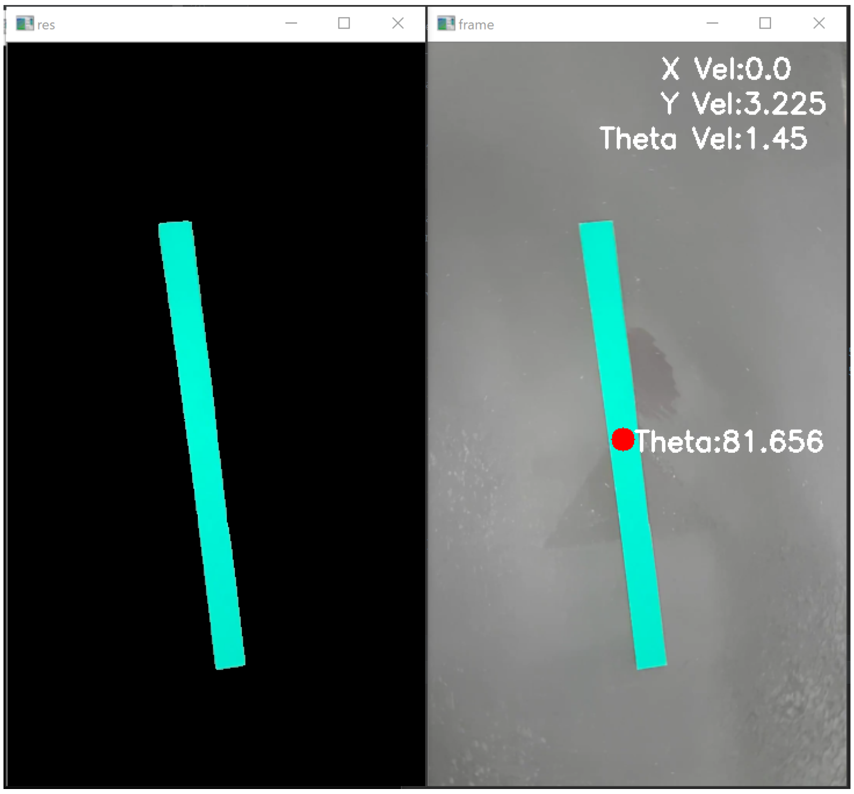
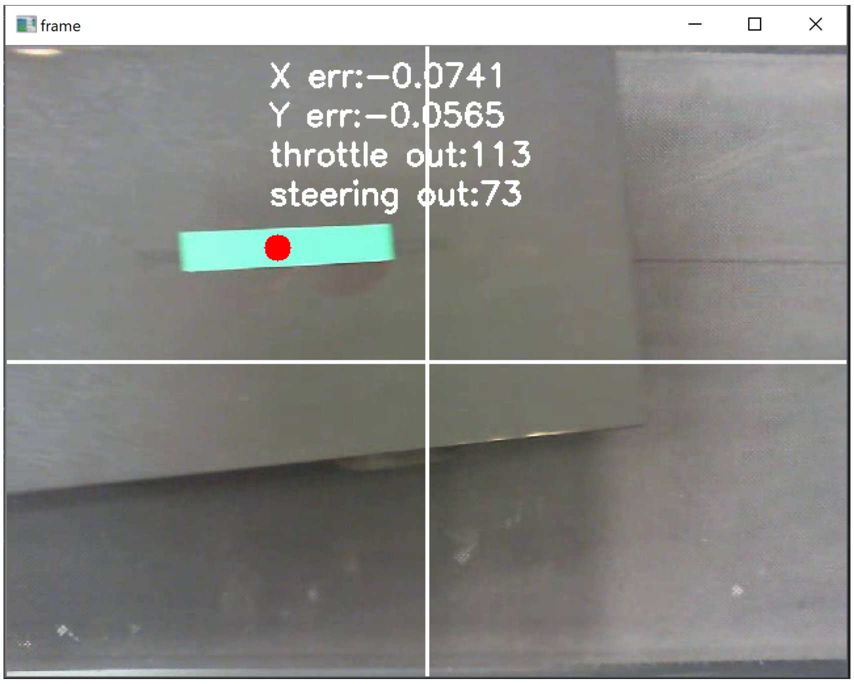


**Figure 3.5:** The threshold once complete showing the isolated green rectangle.



```
X err:-0.0741
Y err:-0.0565
throttle out:113
steering out:73
```

**Figure 3.6:** An example frame from the video stream that has been overlaid with information.

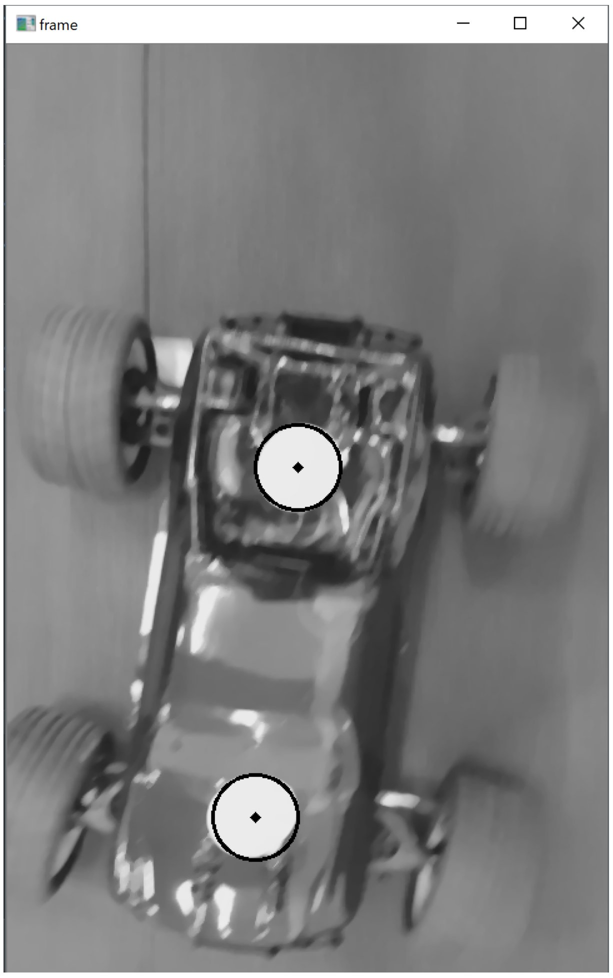


**Figure 3.7:** A resultant frame from the hough circle based algorithm

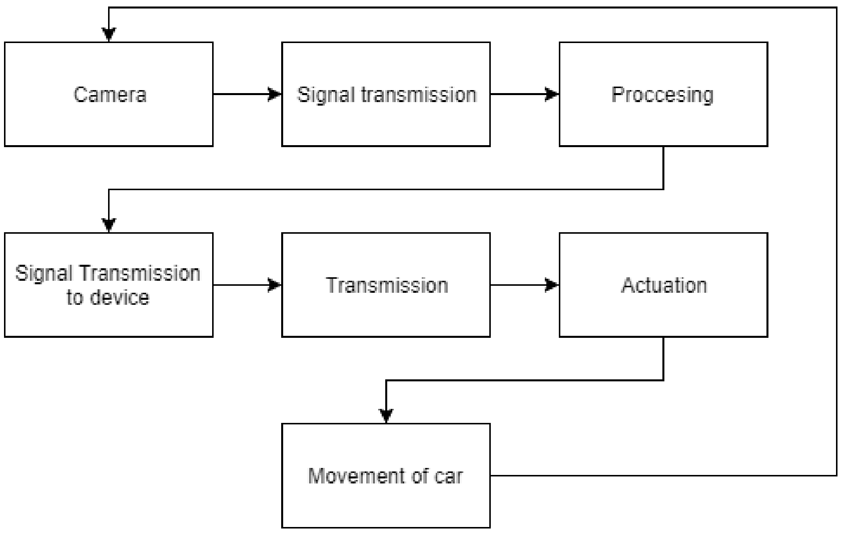


**Figure 3.8:** Summary of latencies in proposed system.

# Chapter 4

# Results

## 4.1 Kinematic System Model

### 4.1.1 Step Response

#### Open Loop

A step response for the open loop model can be seen in figure 4.1 this figure depicts the vehicle having its steering angle stepped at 1s from 0 to 0.5 radians. The model behaves as expected, a car with its steering held at a constant angle moves in a circle. Note that the reason for the vehicle not drawing over the same path is due to the white noise being added in the model to simulate real world imperfections.

#### Closed Loop

This model when stepped at 1s from 0 to 0.5 radians steering angle behaves as shown in 4.2. The velocity of the car for this simulation was 7m/s. This is very similar to the classic critically damped step response but notice the curved rise and level off due to the non-linearity of the system.

The performance of this algorithm is summarized with the following characteristics,

- Rise time: 0.3064s
- Settling time: 1.6s
- overshoot: 0cm

#### Discretization

To simulate the discrete control loop that would be operating due to the computational and frame rate delays a delay block was placed between the control input for the cars and the vehicle plant. The impact this has is clear in figure 4.3 as the delay is increased

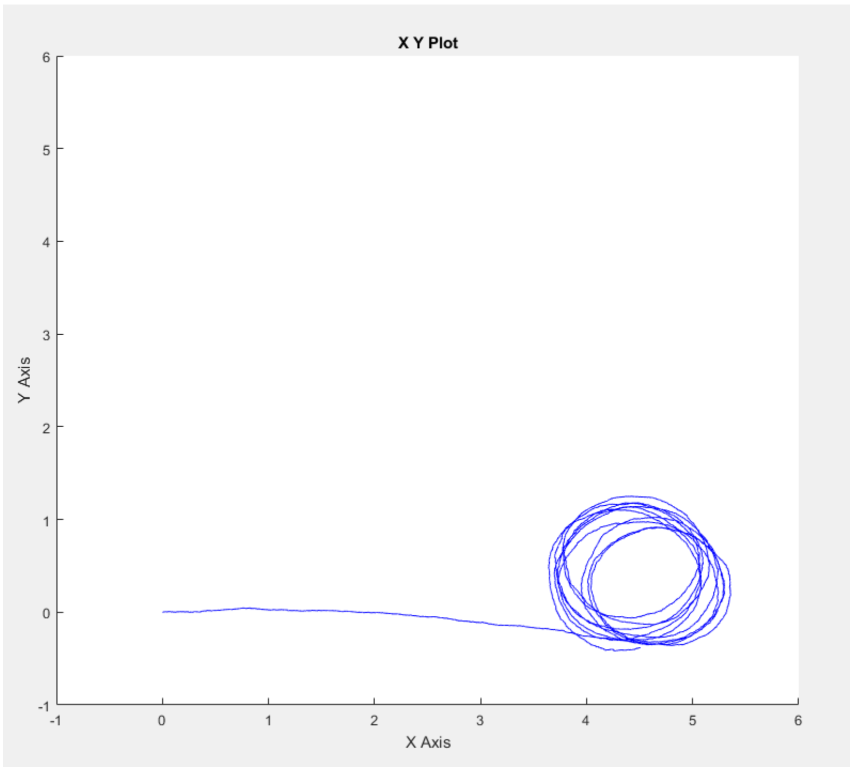


**Figure 4.1:** Open-loop step response kinematic system.

instability increases until at a delay of 22ms control breaks down and the system becomes unstable.

## 4.2 State Space System Model

### 4.2.1 Step Response

#### Open Loop

The state space model described in section 2.3.2 was entered into MatLab and it's open loop step response was measured. This response is shown in 4.4 As shown the system becomes unstable immediately without control and the errors grow. Since the time scale on the graph is so small we know that this would likely not occur in reality since there is lag in the real system, things like first order lag of the servos and inertia would prevent this.

#### Closed Loop

Once the control method has been applied a stable system can be observed in figure 4.5. The exact performance of this algorithm can be summarized with these characteristics:

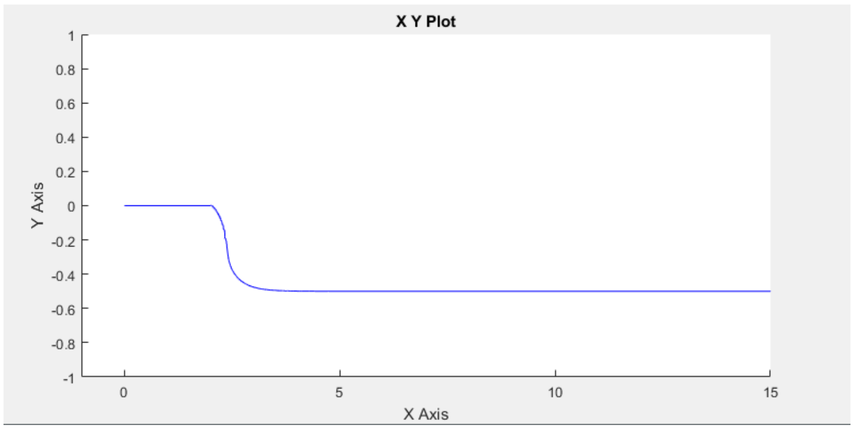


**Figure 4.2:** Step response of PID controlled system.

- Rise time: 0.626s
- Settling time: 1.09s
- overshoot: 0.125cm

## 4.3 Latencies

### 4.3.1 Computer Vision Program Latency

#### Moments Based Algorithm

By measuring the time it takes to complete each loop of the main moments Python programming loop and averaging them the latency can be measured. This value is 3ms, considering that the program must wait 33.3ms between frames from the 30fps camera, this value is insignificantly small, suggesting that camera may be the main limiting factor.

#### Hough Circle Transform Based Algorithm

By again measuring the time it takes to complete each loop of the Hough circle Python programming loop and averaging them the latency can be measured. This value is 16.5 ms, this higher value is due to the computationally more costly `cv2.HoughCircles()` function. Still, it completes within the allowable time between frames of 33.3ms

#### Comparison

The Hough circle transform method due to the way it finds features by systematically trying circles of various sizes is more computationally costly than the very computationally simple moments method. This allows for a 5.5 times reduction in frame analysis time.

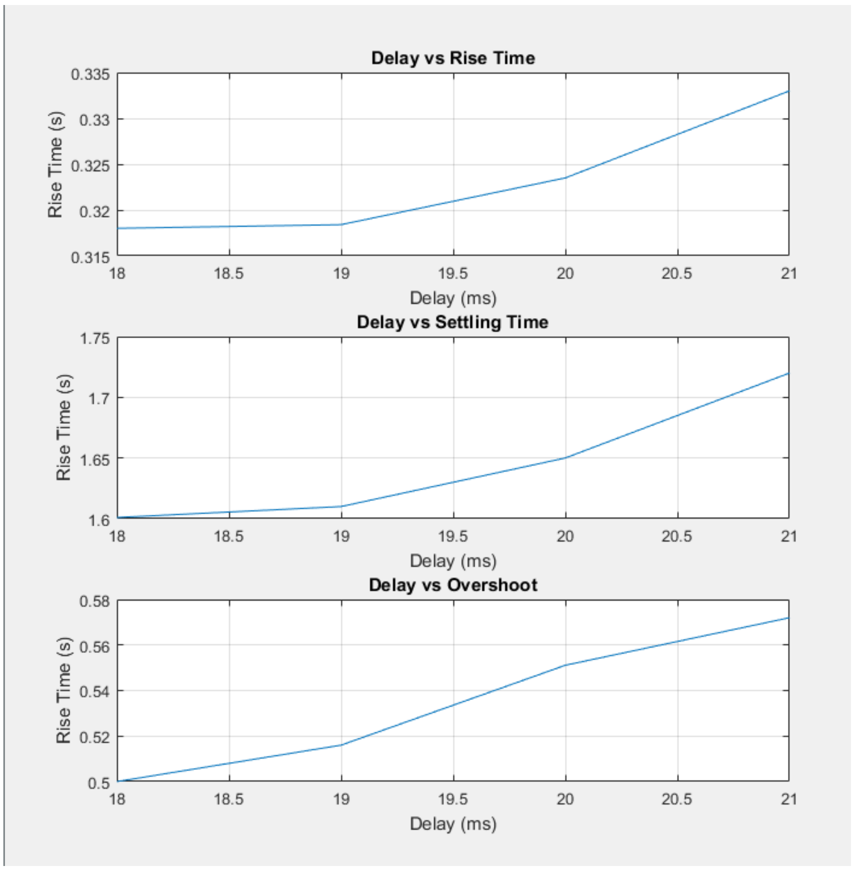


**Figure 4.3:** Step response parameters response to increased delay.

### 4.3.2 Transmission Latency

#### Bluetooth Latency

Bluetooth latency when using the RFCOMM port is limited by the baud rate in this case the HC-05 module purchased must have a baud rate of 9600 bps meaning that it can send 1200 bytes a second and since code requires 8 bytes per control signal updating it should update 150 times per second resulting in a 6.66 ms delay this can be confirmed with the same method from sections 4.3.1 and 4.3.1. That reads a 6 ms delay per control update. It can also be shown that the Arduino is able to check that a byte has or has not arrived three times between bytes arriving and can be measured to be 2ms. Meaning that the Arduino processing time is not currently significant compared to the 9600 baud rate.

### 4.3.3 Total Latency

The total latency from real time model to actuation. Can then be estimated to be the sum of the:

- Camera latency: 33.3ms

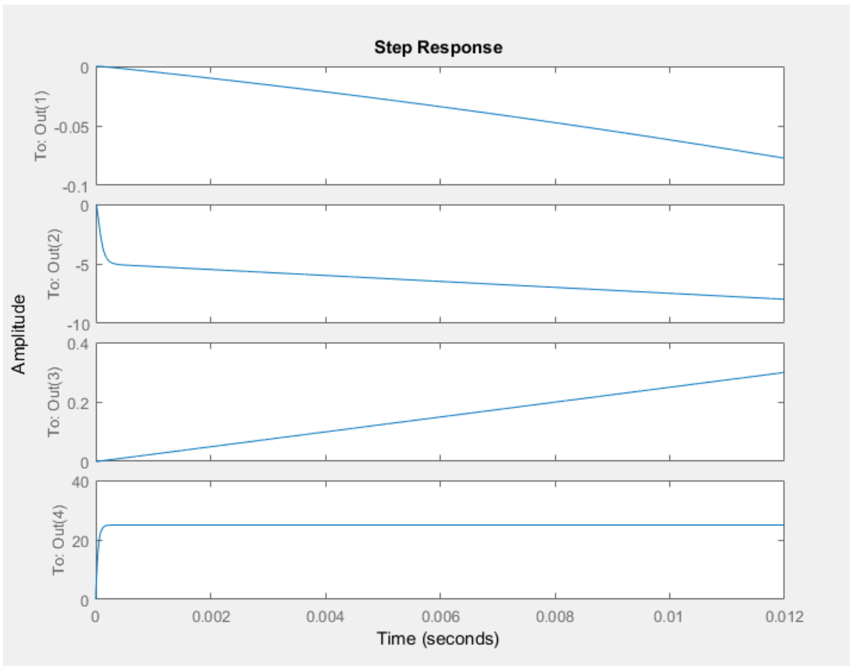


**Figure 4.4:** Open loop step response curves of $e_1, \dot{e_1}, e_2, \dot{e_2}$ from top to bottom respectively.

- Algorithm latency: 3ms
- Transmission to actuation: 6ms

This value is then 42.3 ms. with almost 79% of the delay coming from the camera. This calculation also does not take into some factors such as, USB 2.0 delay, first order lag delay of the servo and other intrinsic physical and electronic delays inherit to the system.

## 4.4 Future Work

- The models values need to be tuned to enable the model to *drive* on the belt so that its suitability as solution can be evaluated.
- The system could be tuned for maximum stability at three or more different belt velocities and then a polynomial gain scheduler could be implemented in conjunction with a belt speed measuring device for continuous stability at every belt speed.
- Work conducted to compare an experimental step response to the theoretical response. These results could be obtained with the current equipment by logging the models center of mass coordinates and having the desired set point suddenly changed. The data could then be scaled appropriately and step response performance characteristics compared to the theoretical values.

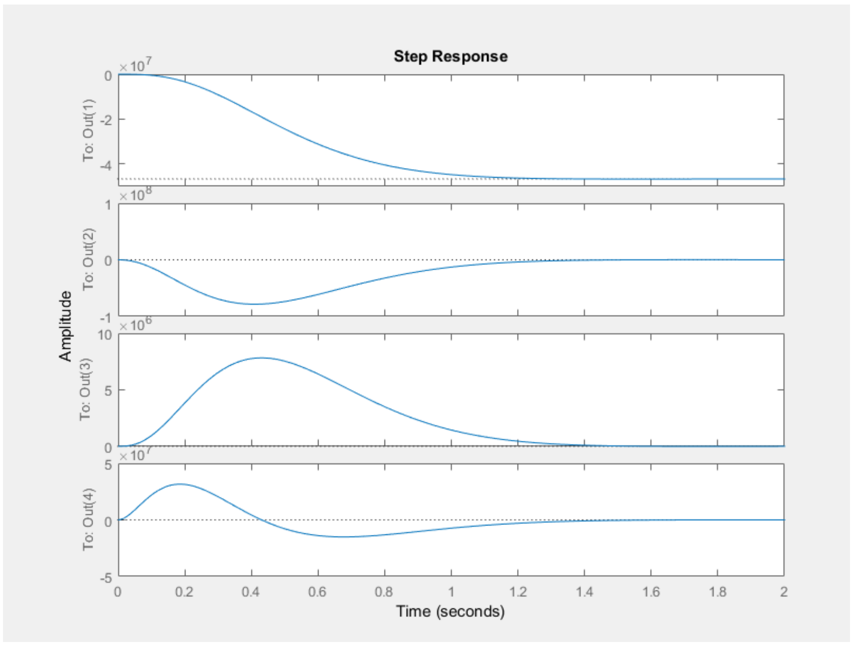


**Figure 4.5:** Closed loop step response curves of $e_1, \dot{e_1}, e_2, \dot{e_2}$ from top to bottom respectively.

- Once a prototype system was fully operational the data from the system such as a maximum velocity that the system is stable at. A new system that was balanced for maximum performance and cost efficiency could be reliably designed to decrease the delay and therefore increase the maximum velocity.

# Chapter 5

# Conclusion

In conclusion this project took a classical control theory approach with a novel sensor choice (computer vision) to the problem of autonomous automotive wind tunnel model control. An extensive literature review was conducted spanning many disciplines from control theory to wind tunnel testing.

To control the vehicle model a simulation was constructed to test ideas and a PID control algorithm, pure pursuit control algorithm and pole placement control algorithm where theoretically derived.

Next the development of a computer vision system to determine the vehicle models position for control was completed undergoing several iterations before a computationally efficient solution based on image moments was found. As well as the camera and computer hardware to make this computation possible.

A solution for the wireless transmission of control signals was found using a bluetooth RFCOMM serial port emulator made possible by an Arduino and an HC-05 bluetooth module. The algorithm that runs on the Arduino also undergoing several iterations to be able to read two separate channels of information on a single serial port computationally efficiently.

All these sub-systems work satisfactorily with them all being bottle necked mostly by the camera frame rate. Future work conducted to establish the real world stability of the system will be interesting.

# Chapter 6

# Abbreviations

| | |
|---|---|
| AWGN | Additive White Gaussian Noise |
| SISO | Single Input Single Output |
| MIMO | Multiple Input Multiple Output |
| COG | Centre Of Gravity |
| RC | Radio Controlled |
| PID | Proportional Integral Derivative |
| FPGA | Field Programmable Gate Array |
| HSV | Hue Saturation Value |
| ROI | Region of Interest |
| DC | Direct Current |
| ESC | Electronic Speed Control |
| FPS | Frames Per Second |
| USB | Universal Serial Bus |
| MAC | media access control |
| ASCII | American Standard Code for Information Interchange |
| CFD | Computational Fluid Dynamics |
| RGB | Reg Green Blue |
| PCM | Pulse Code Modulation |
| MSBS | Magnetic Suspension and Balance System |
| LDR | Light Dependent Resistors |

# Appendix A

# Software

## A.1 Main Python Program

```python
import cv2
import numpy as np
import time
import math
import bluetooth

bd_addr = "98:D3:34:91:19:9A" #the address from the Arduino sensor
port = 1
sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
sock.connect((bd_addr, port))

cap = cv2.VideoCapture('output.avi')

pixelDistance = 0.00065
xIZone = 100
yIZone = 100

baseMin = 0
baseMax = 1
limitMin = 0
limitMax = 255

throttleKp = 10
throttleKi = 0
throttleKd = 0

steeringKp = 10
steeringKi = 0
```

```
steeringKd = 0


font = cv2.FONT_HERSHEY_SIMPLEX
prevX = 0
prevY = 0
prevTheta = 0

prevXError = 0
prevYError = 0

xIntegral = 0
xDerivative = 0
yIntegral = 0
yDerivative = 0


def limitvalue(val, maximum, minimum):
    if val > maximum:
        return maximum
    elif val < minimum:
        return minimum
    else:
        return val


def scaleclipped(valuein, basemin, basemax, limitmin, limitmax):
    xControlUnclipped = ((limitmax - limitmin) * (valuein - basemin) / (basemax -
    return limitvalue(xControlUnclipped, limitmax, limitmin)

lower_green = np.array([30, 50, 255])
upper_green = np.array([180, 255, 255])

while cap.isOpened():
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)



    mask = cv2.inRange(hsv, lower_green, upper_green)
    res = cv2.bitwise_and(frame, frame, mask=mask)

    M = cv2.moments(mask)
```

```
cx = int(M['m10'] / M['m00'])
cy = int(M['m01'] / M['m00'])
a = int(M['m20'] / M['m00'] - cx ** 2)
b = int(2 * (M['m11'] / M['m00'] - cx * cy))
c = M['m02'] / M['m00'] - cy ** 2

theta = 1 / 2 * math.atan(b / (a - c)) + (a < c) * math.pi / 2
thetaDeg = round(math.degrees(theta), 3)

velX = round((prevX - cx) * pixelDistance / 0.333, 3)
velY = round((prevY - cy) * pixelDistance / 0.333, 3)
velTheta = round((prevTheta - thetaDeg) / 0.333, 3)

xError = round((cx - 320) * pixelDistance, 4)
yError = round((cy - 240) * pixelDistance, 4)

if abs(xError) < xIZone:
    xIntegral = (xError * throttleKi) + xIntegral
else:
    xIntegral = 0
xDerivative = velX * throttleKd
prevXError = xError
throttleSignal = xError * throttleKp + xIntegral + xDerivative
throttleControl = int(scaleclipped(-throttleSignal, 0, 1.0, 95, 120))

if abs(yError) < yIZone:
    xIntegral = (xError * throttleKi) + xIntegral
else:
    xIntegral = 0
yDerivative = velY * steeringKd
prevYError = yError
steeringSignal = yError * steeringKp + yIntegral + yDerivative
steeringControl = int(scaleclipped(steeringSignal, -1.0, 1.0, 60, 120))

cv2.putText(frame, "X err:" + str(xError), (200, 30), font, 0.8, (255, 255, 2
cv2.putText(frame, "Y err:" + str(yError), (200, 60), font, 0.8, (255, 255, 2
cv2.putText(frame, "throttle out:" + str(throttleControl), (200, 90), font, 0
cv2.putText(frame, "steering out:" + str(steeringControl), (200, 120), font,
cv2.circle(frame, (cx, cy), 10, (0, 0, 255), -2)
cv2.line(frame, (320, 0), (320, 480), (255, 255, 255), 2)
cv2.line(frame, (0, 240), (640, 240), (255, 255, 255), 2)

cv2.imshow('frame', frame)
```

```
    #cv2.imshow('mask', mask)
    #cv2.imshow('res', res)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

    toSend = str(steeringControl) + "," + str(throttleControl) + '\n'
    print(toSend)
    sock.send(toSend)

    prevX = cx
    prevY = cy
    prevTheta = thetaDeg

    #time.sleep(0.1)
sock.close()
cv2.destroyAllWindows()
cap.release()
```

## A.2 Main Arduino Program

```
#include <SoftwareSerial.h>
#include "Servo.h";
Servo steer;
Servo throttle;

//Change these to whatever pins you're using
SoftwareSerial blueSerial(10, 11); // RX, TX
String steerCon;

unsigned int integerValue=0;  // Max value is 65535
char incomingByte;

void setup()
{
  Serial.begin(9600);
  blueSerial.begin(9600);
  pinMode(9, OUTPUT);
  digitalWrite(9, HIGH);
  steer.attach(3);
  throttle.attach(6);
}



void loop() {
  if (blueSerial.available() > 0) {   // something came across serial

    String sC = "";
    String tC = "";

    while(1) {              // force into a loop until 'n' is received
      incomingByte = blueSerial.read();

      if (incomingByte == -1) continue;  // if no characters are in the buffer r

      if (incomingByte == ',') {
          Serial.print(sC + ", ");
          break;
      }
      sC += String(incomingByte);
    }

    steer.write(toInt(sC));

    while(1) {              // force into a loop until 'n' is received
      incomingByte = blueSerial.read();

      if (incomingByte == -1) continue;  // if no characters are in the buffer r

      if (incomingByte == '\n') {
          Serial.println(tC);
          break;
      }
      tC += String(incomingByte);
    }
    throttle.write(toInt(tC))
  }
}
```

## A.3 MatLab State-Space Program

```
vx=10;
psi=0;
lf=0.2;
lr=0.2;
caf=50000;
car=50000;
d=0;
m=1.2;
iz=0.03;
t=0:0.001:5;
u = zeros(size(t));

A=[0                1                      0                            0
   0    -(2*caf+2*car)/(m*vx)   -(2*caf+2*car)/(m) -(2*caf*lf+2*car*lr)/(m*vx)
   0                0                       0                           1
   0    -(2*caf*lf-2*car*lr)/(iz*vx) -(2*caf*lf-2*car*lr)/(iz) -(2*caf*lf^2+2*car

B=[0
   ((2*caf*d)/m)-(((2*caf*lf-2*car*lr)/(m*vx))-vx)*psi
   0
   ((2*caf*lf)/iz)-((2*caf*lf^2+2*car*lr^2)/(iz*vx))*psi];
C=[1 0 0 0
   0 1 0 0
   0 0 1 0
   0 0 0 1];
D=[0
    0
    0
    0];
p = [-5-3i
    -5+3i
    -7
    -10];
K = place(A,B,p);
sys=ss(A-B*K,B,C,D);
sys1=ss(A,B,C,D);
%step(sys)
step(sys1)
%lsim(sys,u,t);
```

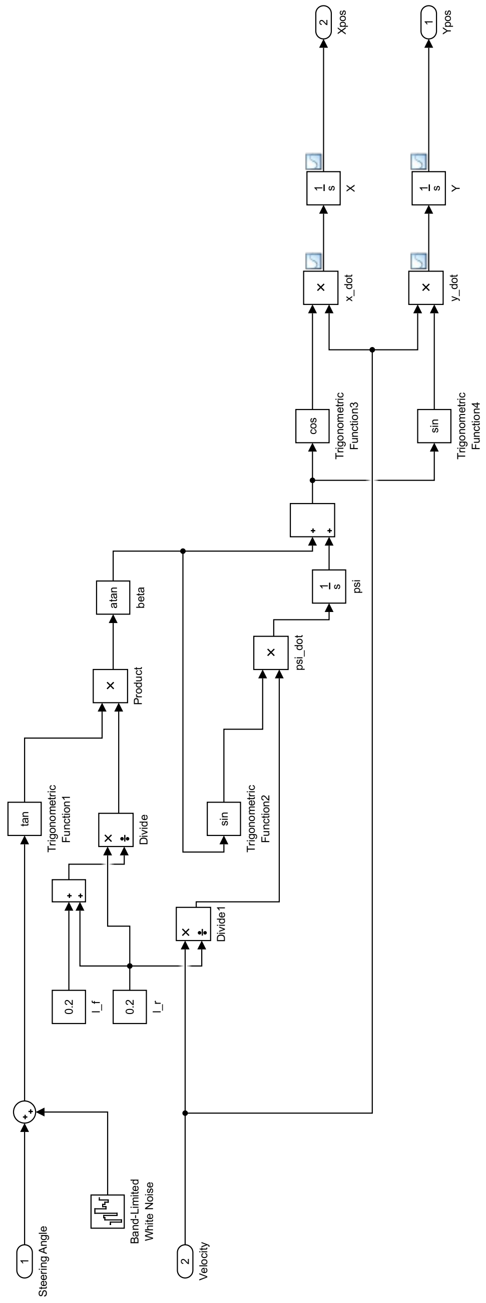
## A.4 Hough Circle Algorithm

```
import cv2
import numpy as np
import time

cap = cv2.VideoCapture('output1.avi')


while cap.isOpened():

    # Take each frame
    _, frame = cap.read()
    frame = cv2.medianBlur(frame,5)

    cimg = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(cimg, 220, 255, cv2.THRESH_BINARY)
    circles = cv2.HoughCircles(thresh, cv2.HOUGH_GRADIENT, 1, 20, param1=50, para

    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in circles[0,:]:
            # draw the outer circle
            cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
            # draw the center of the circle
            cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

    cv2.imshow('frame',cimg)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    #time.sleep(0.01)

cv2.destroyAllWindows()
```

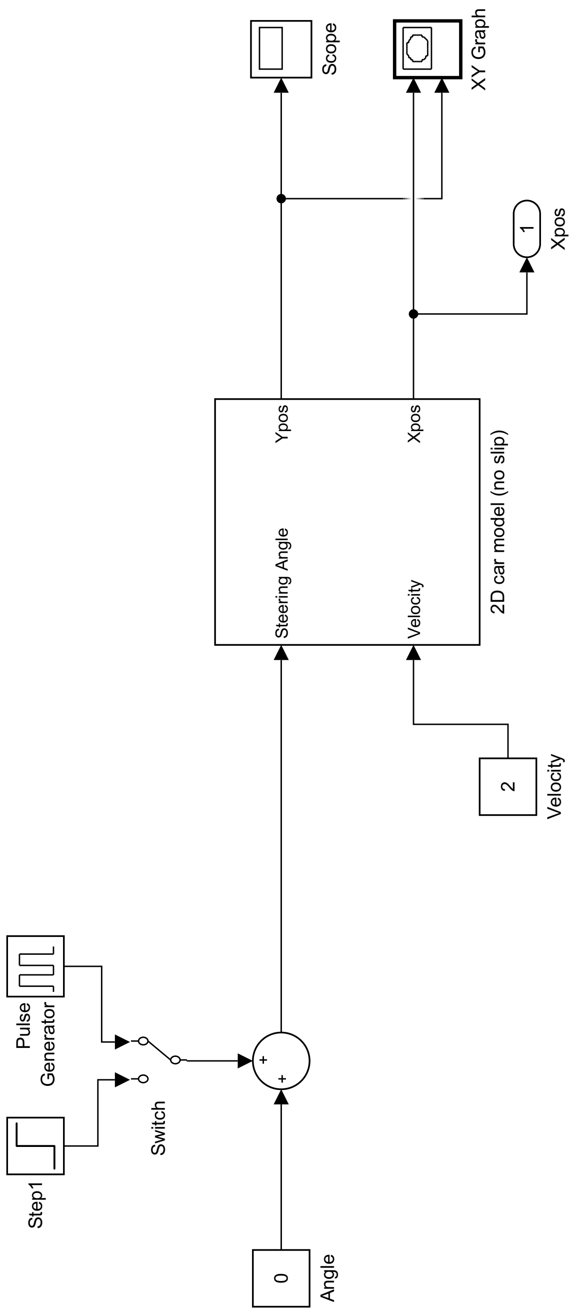
# Appendix B

# Simulink Models

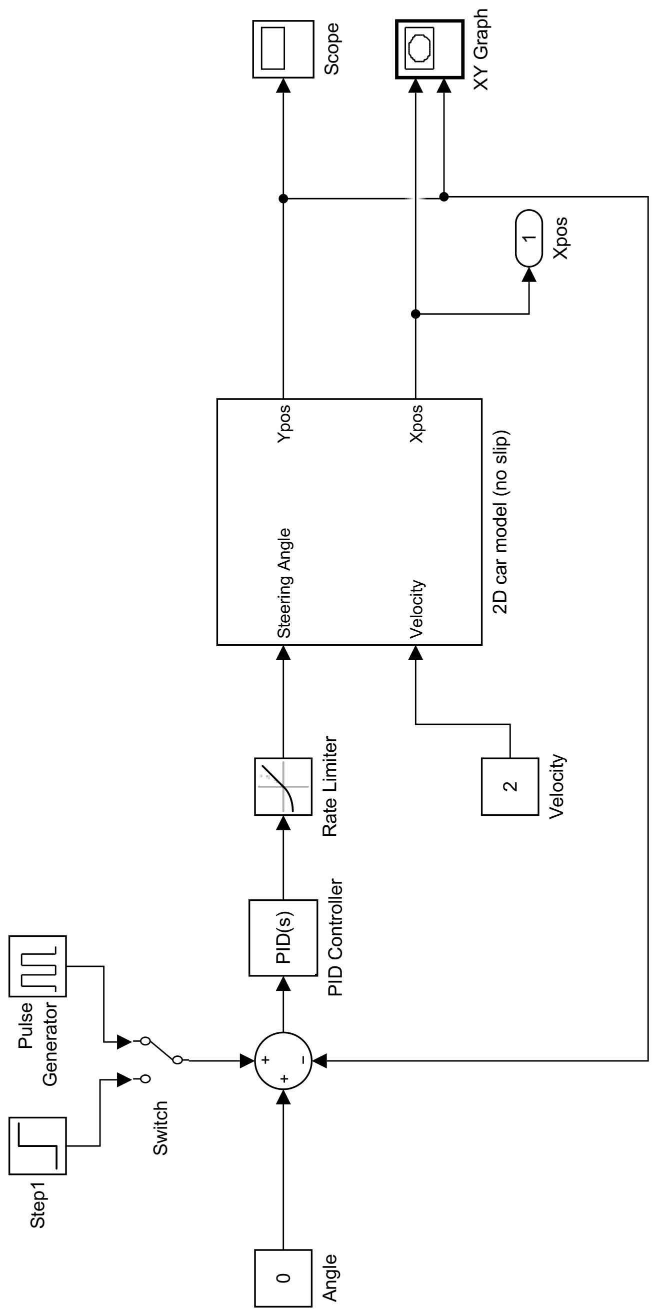
## B.1 Open Loop Simulink Model Sub-system



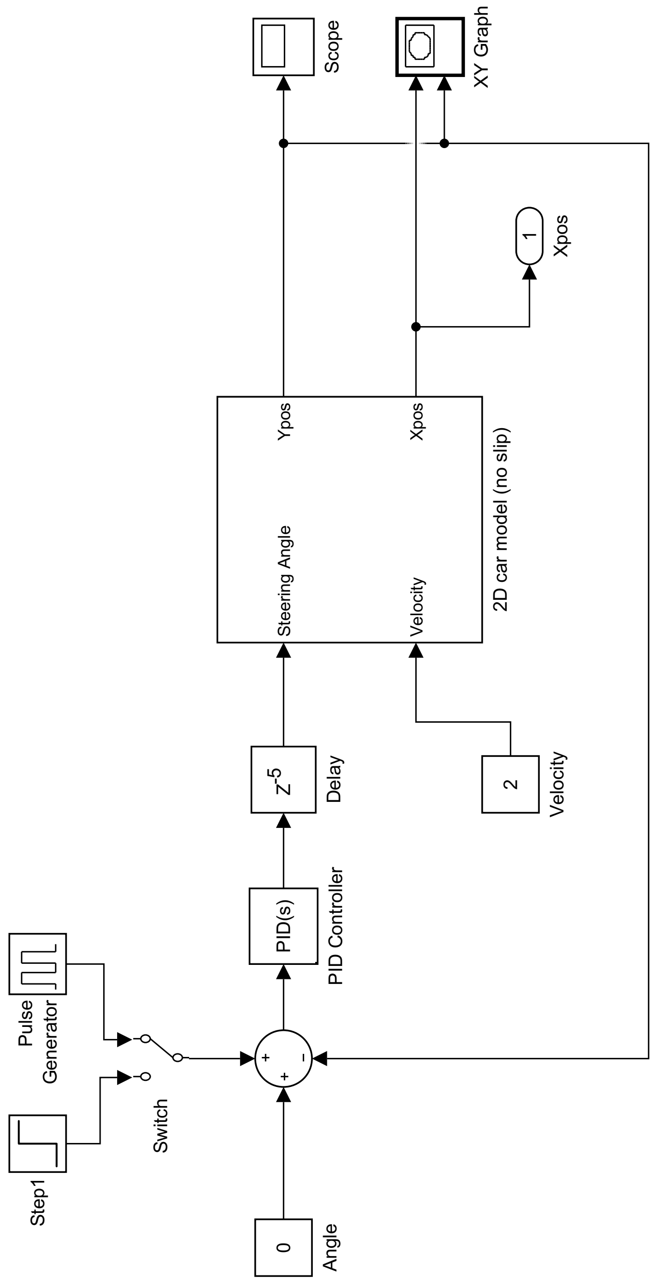
## B.2 Open Loop Simulink Model



## B.3 Closed Loop PID controlled Simulink Model



## B.4 Discretized Loop PID controlled Simulink Model



# Bibliography

[1] U.S. Patent 4 918 672, 1990.

[2] U.S. Patent 5 262 837, 1993.

[3] B.Hetherington and D. Sims-Williams, "Support strut interference effects on passenger and racing car wind tunnel models," *SAE Technical papers*, 2006.

[4] A. Cogotti, "Evolution of performance of an automotive wind tunnel," *Journal of Wind Engineering*, 2008.

[5] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," *Defence Techincal Information Center*, 1992.

[6] S. Diasinos, "The aerodynamic interaction of a rotating wheel and a downforce producing wing in ground effect," Ph.D. dissertation, University of New South Wales, 2009.

[7] W. G.Johnson and D. A.Dress, "The 13-inch magnetic suspension and balance system wind tunnel," *NASA technical Memorandum*, 1989.

[8] A. S. Huang and L. Rudolph, *Bluetooth for Programmers*, C. U. Press, Ed. Cambridge University press, 2007.

[9] K. Huibert and S. Raphael, *Linear Optimal Control systems*, 1st ed., Wiley-Interscience, Ed. Wiley, 1972.

[10] Itseez, *Open Source Computer Vision Library*, https://github.com/itseez/opencv.

[11] L. Kotoulas and I. Andreadis, "Image analysis using moments," in *5th International Conference on Technology and Automation*, 2005.

[12] C. l. VanGordon and J. Walter, "Overview of wind tunnel testing for automotive development," *SAE international*, 2008.

[13] Z. LIU, W. CHEN, and Y. ZOU, "Regions of interest extraction based on hsv color space," *IEEE*, 2012.

[14] S. Mahmud, "Autonomous control system with drag measurement capability for a wind tunnel model vehicle," Master's thesis, Macquarie University, 2016.

[15] N. S. Nise, *Control Systems Engineering*, 7th ed., Wiley, Ed. Wiley, 2014.

[16] R. Rajamani, *Vehicle Dynamics and Control*, Springer, Ed. Springer, 2006.

[17] L. G. Reagan, P. D. Elmer, and A. B. Argel, "Path planning for quadrotor uav using genetic algorithm," in *7th IEEE International Conference Humanoid, Nanotechnology, Information Technology Communication and Control, Environment and Management*, 2014.

[18] H. Rhody, "Hough circle transform," *Center for Imaging Science*, 2005.

[19] D. Schramm, M. Hiller, and R. Bardini, *Vehicle Dynamics*, Springer, Ed. Springer, 2014.

[20] SharkD, "Hsv colorspace," https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png, Dec. 2015.

[21] J. M. Snider, "Automatic steering methods for autonomous automobile path tracking," *IEEE*, 2009.

[22] R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, Ed. Springer, 2010.

[23] S. F. Team, "Wind tunnel," Web page, 2017, https://www.sauberf1team.com/corporate/factory.

[24] J. Vujic, A. Marincic, M. Ercegovac, and B. Milovanovic, "Nikola tesla: 145 years of visionary ideas," in *5th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service. TELSIKS 2001*, 2001.