UNIVERSITY OF CALIFORNIA SAN DIEGO

A Guide for Understanding and Implementing Optimal Control for Autonomous Vehicles

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Mechanical and Aerospace Engineering

by

Dominic James Nightingale

Committee in charge:

Professor Mauricio de Oliveira, Chair
Professor Robert Bitmead, Co-Chair
Professor Thomas Bewley

2023

The Thesis of Dominic James Nightingale is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

I would like to express my sincere gratitude to the individuals who have contributed to the successful completion of my research and thesis. First and foremost, I would like to thank my advisor, Dr. Mauricio de Oliveira for his unwavering support, guidance, and mentorship throughout my research journey. He taught me how to approach challenging problems by segmenting the fundamental components and how to think more deeply about the theory and application of my experiments. He pushed me to become a better engineer, provided me with the resources and opportunities to develop my skills, and gave me the confidence to take risks and tackle difficult challenges. Dr. Oliveira's ability to balance humor and hard work made the research process both productive and enjoyable. I am deeply grateful for his mentorship and for the many laughs we shared along the way.

Dr. Jack Silberman introduced me to the world of robotics and what it means to be a hacker. His enthusiasm and expertise in the field inspired me to pursue research in robotics, and he provided me with the foundational knowledge and technical skills necessary to begin exploring this exciting field.

Dr. Robert Bitmead introduced me to the world of control and estimation theory and showed me how to turn any dull moment into a joke. He made learning about these complex topics fun and accessible, and his humor and enthusiasm kept me motivated and engaged throughout his courses.

Dr. Thomas Bewley introduced me to the world of numerical estimation and dynamic modeling and instilled in me the importance of working efficiently. He taught me how to approach problems with a systematic, analytical mindset, and how to use numerical methods to develop accurate models and simulations.

Dr. Sven Brueggemann, a fellow student, introduced me to the world of optimal control and vehicle modeling, and taught me about the ideas of safe control methods. His insights and technical expertise were invaluable to my research, and his friendship and encouragement helped to keep me motivated and engaged throughout the research process.

Siddharth Saha, another fellow student, is the best programmer I know, and he helped me interface with many hardware components and provided support during many long nights of programming. His technical skills and knowledge were instrumental to the success of my research, and his humor and camaraderie made even the most challenging tasks feel more manageable.

Zhuolin Niu, another fellow student, helped me set up and perform many of the experiments that went into my thesis, as well as testing the algorithms discussed in the thesis. Her technical expertise and attention to detail were essential to the accuracy and reliability of my research.

Hoojon Shiin, another fellow student, helped me develop navigation algorithms that were lidar-based. His technical knowledge and creativity helped to ensure that the algorithms were accurate, efficient, and effective.

I want to thank my closest friends, Sepher Bostan, Chris Light, Deena Jaber, Alexander Luke, Kevin Lam, and Karen Hernandez, for their unwavering support and encouragement. They have been there for me at one point or another in my life and have given me nothing but support, laughter, and offerings of true friendship. I will always be grateful for having them in my life as their support has helped to sustain my sanity throughout my research journey and continue to be important people in my life.

Last but not least, I want to thank my beloved mother, who has always been my source of strength and inspiration and has supported me in every step of my life. Without her, I would not be where I am today. Thank you all for your contributions, encouragement, and love.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT OF THE THESIS

A Guide for Understanding and Implementing Optimal Control for Autonomous Vehicles

by

Dominic James Nightingale

Master of Science in Mechanical and Aerospace Engineering

University of California San Diego, 2023

Professor Mauricio de Oliveira, Chair
Professor Robert Bitmead, Co-Chair
Professor Thomas Bewley

There is a notable gap in existing autonomous vehicle control literature that provides comprehensive guides bridging control design theory to its real-world implementation. The primary objective of this thesis is to address this gap by facilitating a clear understanding of the control design process, allowing readers to seamlessly transition from theory to application in implementing controllers for autonomous vehicles. This thesis is designed to operate as a user's

manual, divided into two parts, providing a guide for the understanding of the theoretical background of autonomous vehicles, discussed in Chapter 1, followed by a detailed guide on the procedures for implementing control theory on physical autonomous vehicles, discussed in Chapter 2. Topics such as vehicle modeling, state estimation, system identification, and control are covered in Chapter 1, while Chapter 2 guides the reader through the core algorithms used, the utilization of the autonomous vehicle framework and detailed experimental procedures for data collection and controller testing, both in simulation and on the physical vehicle. Tailored for a broad educational audience, this thesis assumes only a foundational knowledge of Linux systems, linear algebra, differential equations, and basic physics related to moving objects.

Chapter 1 Theory

1.1 Introduction

For implementing control, knowing how to characterize the system is a very important step. This characterization can be a single or set of equations that explain the dynamics of the system in terms of state variables which allow the use of many linear and optimal control theories to be implemented. This thesis employs several variations of the kinematic and dynamic bicycle car models for their simplicity and robustness across various vehicle platforms. It also discusses the differences between the linear and non-linear models and when to use them. Understanding the dynamics however is not enough because in most scenarios the model contains unknown parameters that must be determined experimentally. This is explored using standard system identification techniques which are under the assumption that the data collected during the experiments are noise free, meaning that the data needs to be filtered, which brings up the idea of Kalman filtering. Several types of Kalman filters are used for the filtering process which is done online for state measurements and offline for parameter estimations. Once the system parameters are known and the sensor data is filtered, optimal control techniques from solving the dynamic programming equation to Schur factorizations are used to get the vehicle following a reference trajectory. Everything is derived from the most fundamental levels except for general mathematical identities and mathematical abstractions which are given in the appendix. It is to be assumed that the background knowledge of the reader should be that of differential equations, linear algebra, and basic physics of moving objects. It is assumed that the reader has little to no understanding of vehicle modeling and explains in detail how the different

models are derived. A high-level preview of what is to be accomplished can be illustrated in the classic block diagram shown in Figure 1. The following chapters are broken down by understanding each of these blocks and signals.

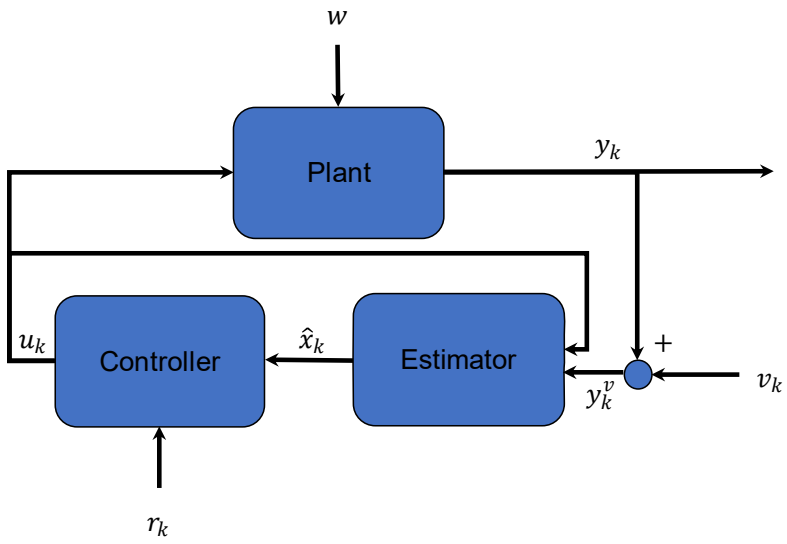

Figure 1: Block Diagram of Linear Quadratic Gaussian Control

With the following:

- Blocks
    - Plant : ***System dynamics (system identification techniques)***
    - Estimator : ***Optimal state of the system (Kalman filter techniques)***
    - Controller : ***Optimal control to apply to plant (Riccati-based techniques)***
- Signals
    - u : Controller input
    - w : disturbance noise
    - y : Plant output
    - v : measurement noise
    - y_v : Measured plant output

- r : reference signal
- $\hat{x}$: optimal state estimate

To understand the plant block, the dynamics and system identification techniques will have to be to be studied to determine certain parameters that are particular to the dynamics of the system. Then for the estimator block, various forms of the Kalman filter are studied and use cases will vary depending on online or offline estimation. Finally for the controller block, Riccati-based feedback control is used to determine the optimal control to apply to the system given some reference to track. The idea of duality between the estimator and controller blocks is also discussed to further show the simplicity and convenience of the Linear Quadratic Gaussian (LQG) controller.

1.2 **Vehicle Modeling**

1.2.1 **Introduction**

The plant block is the first study of interest. The plant block represents the *real* system. The goal is to come up with a mathematical *model* that describes the plant block to be used in the estimator and control blocks. Other than disturbances and errors from discretization, this can be one of the predominant sources of error between what was measured from the real plant and what was predicted by the model. This brings up the notion of design trade-offs. One example would be using a simple or complex motion model to describe the system. Perhaps the simple model is easy to design with but is not as accurate or not robust enough for all types of environments or scenarios. Contrastly for the complex model, it could be difficult to design with but is robust enough for many environments or scenarios. It is important to think about these trade-offs when designing and to fully understand the limitations of the controller to confidently define a safe space for the system to operate in. Figure 2 highlights the state and output equations that attempt to define the real system. The goal is to come up with a mathematical model that describes the plant block to be used in the estimator and control blocks and identify model limitations.

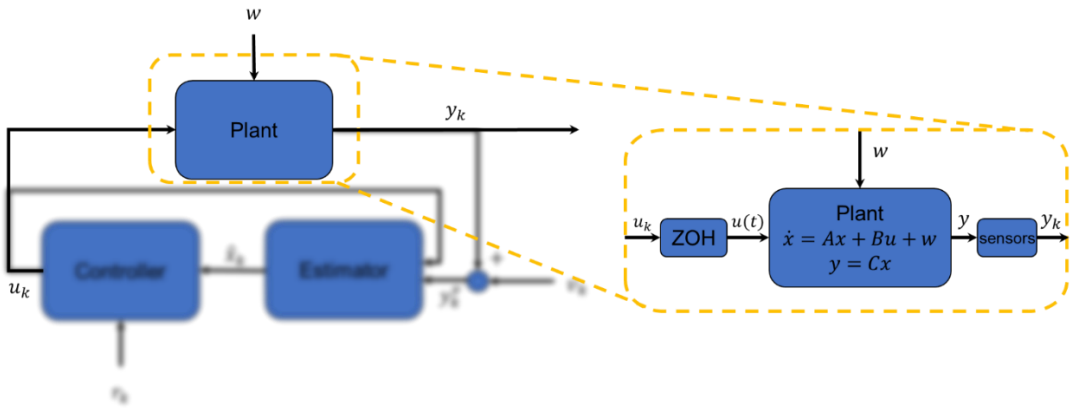

Figure 2: Defining plant block with input and output relationships

1.2.2 **Kinematic Models**

Kinematic models are typically the easiest type of models to work with as they neglect the forces acting on a system and only attempt to describe the motion itself [1]. These models can be very useful when perhaps there are unknown properties of the system that are correlated with the forces acting on it.

1.2.2.1 **Bicycle Car Model**

To first understand the motion prescribed by an Ackermann steering system vehicle, the geometric description of the motion is considered which neglects the forces involved that dictate the motion but are discussed later. Figure 3 depicts this geometric representation of the four-wheel system. The motion is also assumed to be planar which ignores any motion with respect to pitch and roll and only considers yaw.

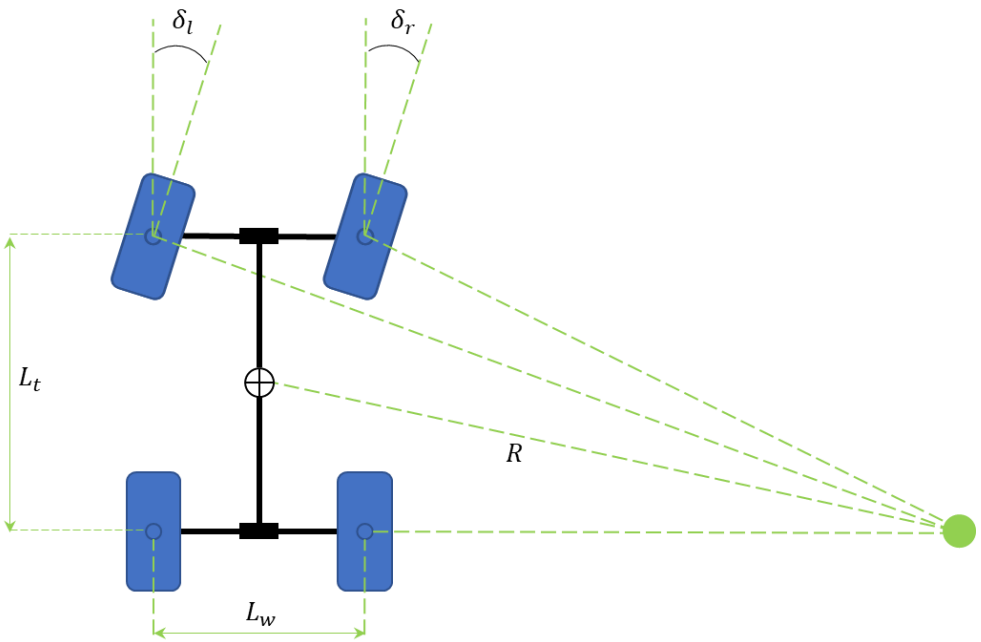

Figure 3: Geometric representation of an Ackermann steering system vehicle

Let:

- L_w be the wheelbase length
- L_t be the track width
- δ_l and δ_r be the front left and right wheel angles respectively
- R be the radius of the arc trajectory traveled around *Point O* to the Center of Gravity (*CG*) of the vehicle in a steady state turning condition

The front wheels each travel at different radii when turning along some arbitrary path. Since each wheel travels a different radius, this means that each of the front wheel angles δ_l and δ_r are different but do not usually vary greatly. Now consider the bicycle model representation of the same system shown in Figure 4.

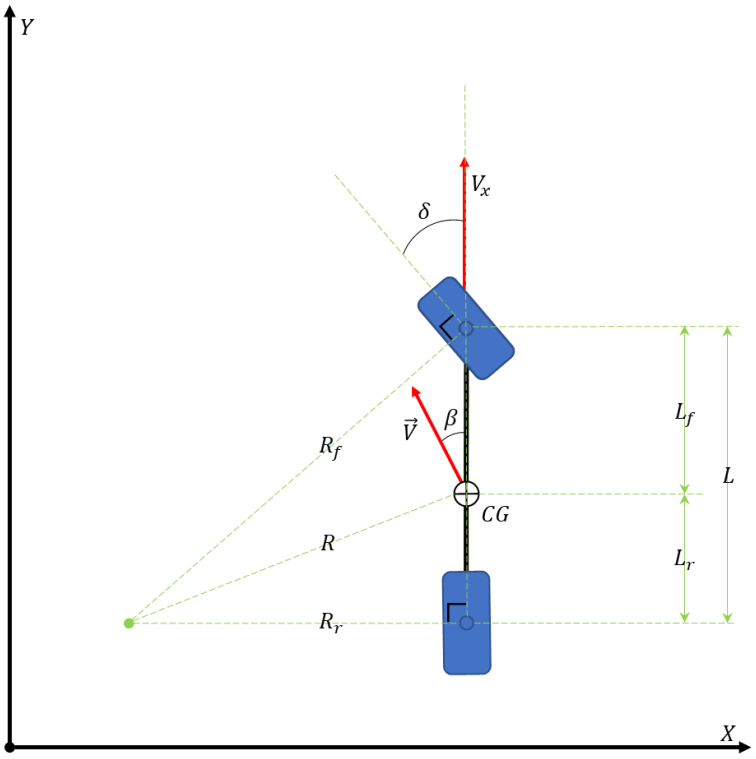

Figure 4: Kinematic Bicycle car model

Let:

- $\boldsymbol{L_r}$ be the length from the real wheel (*Point B*) to the *CG* (*Point C*)
- $\boldsymbol{L_f}$ be the length from the front wheel (*Point A*) to the *CG* (*Point C*)
- δ_f be the angle of the front wheel (*Point A*)
- ψ be the heading angle from the global *X* axis to the longitudinal/local x axis of the vehicle
- $\vec{\boldsymbol{V}}$ be the vehicle velocity vector at the *CG* (*Point C*)
- β be the side slip angle from the longitudinal axis of the vehicle to the *CG* velocity vector $\vec{\boldsymbol{V}}$

This model combines the left and right wheels in the front and the rear of the car to be represented as a single wheel in the front and rear and introduces the possibility of the *CG* not

being directly in the center. Using this geometric model, the equations of motion are governed below.

Linear and angular velocities

$$\dot{X} = V cos(\psi + \beta) \tag{1.1}$$

$$\dot{Y} = V sin(\psi + \beta) \tag{1.2}$$

$$\dot{\psi} = \frac{V cos(\beta)}{L_f + L_r} tan\delta_f \tag{1.3}$$

Where the side-slip angle is

$$\beta = tan^{-1}\left(\frac{L_f tan\delta_r + L_r tan\delta_f}{L_f + L_r}\right)$$

Assuming rear wheel in a fixed straight position then yields

$$\beta = tan^{-1}\left(\frac{L_r tan\delta_f}{L_f + L_r}\right) \tag{1.4}$$

Noticing the distinction between the upper and lowercase letters for coordinate frames. Uppercase letters represent state variables with respect to the global coordinate system and lower case with respect to the body frame of the vehicle.

The radius of curvature is calculated as follows:

$$R = \frac{L_r}{sin(\beta)}$$

Now inserting 2.4

$$R = \frac{L_r}{sin\left(tan^{-1}\left(\frac{L_r tan\delta_f}{L_f + L_r}\right)\right)}$$

Now in vector form

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} cos(\psi + \beta) \\ sin(\psi + \beta) \\ \frac{cos(\beta)}{L_f + L_r} tan\delta_f \end{bmatrix} V \quad (1.5)$$

1.2.3 **Dynamic Models**

The kinematic model is only valid at low slip angles and when the forces acting on the tires are minimal [1] and [3]. The dynamic model is introduced to account for the higher speed spectrum. To do this, the lateral forces on the front and rear tires are considered by applying Newton's law along the body frame y-axis of the vehicle and a moment balance about the z-axis as seen in Figure 5.

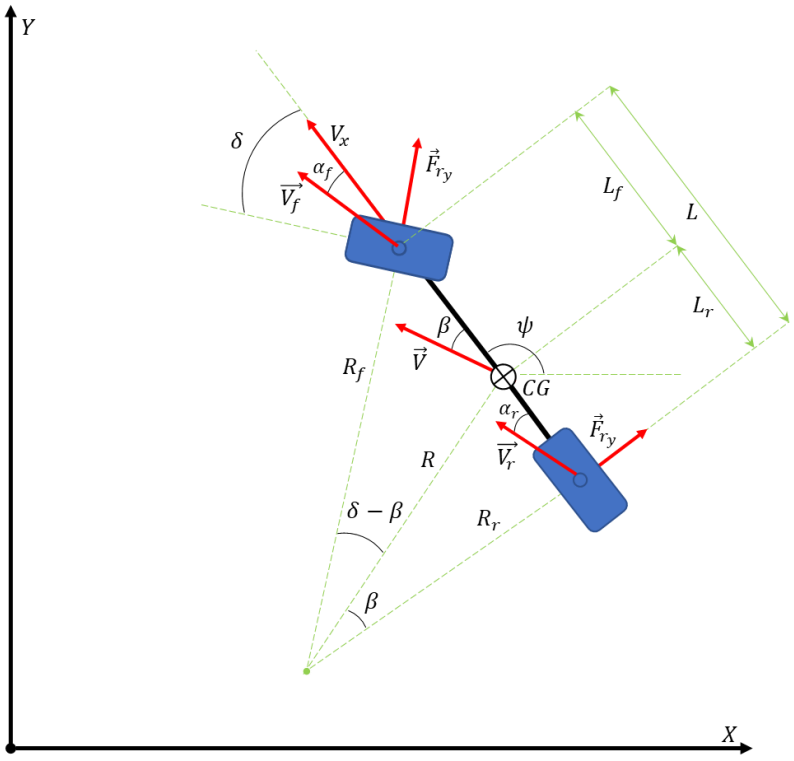

Figure 5: Dynamic Bicycle car model

Letting

$$V_x = \boldsymbol{V}\, cos(\beta)$$

$$V_y = \boldsymbol{V} sin(\beta)$$

Then taking the force balance

$$\sum F_y = m\left(\dot{V}_y + \dot{\psi} V_x\right) = F_{yf} cos(\delta) - F_{yx} sin(\delta) + F_{yr}$$

Assuming constant longitudinal velocity $\left(F_{yx} = 0\right)$ yields

$$\sum F_y = m\left(\dot{V}_y + \dot{\psi} V_x\right) = F_{yf} cos(\delta) + F_{yr} \tag{1.6}$$

Now applying the sum of moments about the CG z-axis

$$\sum M_z = I_z \ddot{\psi} = L_f F_{yf} cos(\delta) - L_r \left(F_{yr} - F_{xr} sin(\delta)\right)$$

Again, assuming constant longitudinal velocity $\left(F_{yx} = 0\right)$ yields

$$\sum M_z = I_z \ddot{\psi} = L_f F_{yf} cos(\delta) - L_r F_{yr} \tag{1.7}$$

Now using a linear model of the lateral tire forces F_{yr} *and* F_{yf}

$$F_{yf} = C_f\left(\delta - \alpha_f\right) \tag{1.8}$$

$$F_{yr} = C_r(-\alpha_r) \tag{1.9}$$

Where C_f and C_r are termed as the tire cornering stiffness coefficients and α_f and α_r are the slip angles of the front and rear tires respectively that are defined as follows

$$\alpha_f = tan^{-1}\left(\frac{V_y + L_f \dot{\psi}}{V_x}\right) \tag{1.10}$$

$$\alpha_r = tan^{-1}\left(\frac{V_y - L_r \dot{\psi}}{V_x}\right) \tag{1.11}$$

1.2.3.1 **Non-linear Bicycle Car Model**

The non-linear model is found by inserting 1.8-1.11 into equations 1.6-1.7 then solving for the lateral and angular accelerations. Starting first with the lateral dynamics yields the following

$$\sum F_y = m(\dot{V}_y + \dot{\psi}V_x) = F_{yf}cos(\delta) + F_{yr}$$

$$\sum F_y = m(\dot{V}_y + \dot{\psi}V_x) = C_f(\delta - \alpha_f)cos(\delta) + C_r(-\alpha_r)$$

$$\sum F_y = m(\dot{V}_y + \dot{\psi}V_x) = C_f\left(\delta - tan^{-1}\left(\frac{V_y + L_f\dot{\psi}}{V_x}\right)\right)cos(\delta) + C_r\left(-tan^{-1}\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right)\right)$$

$$\dot{V}_y = \frac{C_f\left(\delta - tan^{-1}\left(\frac{V_y + L_f\dot{\psi}}{V_x}\right)\right)cos(\delta) - C_r\left(tan^{-1}\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right)\right) - m\dot{\psi}V_x}{m} \quad (1.12)$$

Now for the angular dynamics

$$L_fF_{yf}cos(\delta) - L_rF_{yr} = I_z\ddot{\psi}$$

$$L_fC_f(\delta - \alpha_f)cos(\delta) - L_rC_r(-\alpha_r) = I_z\ddot{\psi}$$

$$L_fC_f\left(\delta - tan^{-1}\left(\frac{V_y + L_f\dot{\psi}}{V_x}\right)\right)cos(\delta) - L_rC_r\left(-tan^{-1}\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right)\right) = I_z\ddot{\psi}$$

$$\ddot{\psi} = \frac{L_fC_f\left(\delta - tan^{-1}\left(\frac{V_y + L_f\dot{\psi}}{V_x}\right)\right)cos(\delta) + L_rC_r\left(tan^{-1}\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right)\right)}{I_z} \quad (1.13)$$

Now putting 1.12 and 1.13 In vector form

$$f(x,u) = \begin{bmatrix}\dot{V}_y \\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}\frac{C_f\left(\delta - tan^{-1}\left(\frac{V_y + L_f\dot{\psi}}{V_x}\right)\right)cos(\delta) - C_r\left(tan^{-1}\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right)\right) - m\dot{\psi}V_x}{m} \\ \frac{L_fC_f\left(\delta - tan^{-1}\left(\frac{V_y + L_f\dot{\psi}}{V_x}\right)\right)cos(\delta) + L_rC_r\left(tan^{-1}\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right)\right)}{I_z}\end{bmatrix} \quad (1.14)$$

Next thing to consider is the linearization of this model so that conventional linear control techniques and tools may be applied but this non-linear model will be revisited again in the optimal state estimation section for non-linear systems.

1.2.3.2 **Linearized Bicycle Car Model**

Now the design trade-off comes in with the desire to work with a simpler model at the sacrifice controller robustness for large slip angles (>15 degrees). In normal driving and racing conditions, the slip angle is usually below this threshold. It is when the vehicle starts drifting that the assumption breaks down and is no longer valid. The control design for the rest of this work will consider normal driving and racing conditions and not extreme conditions such as drifting. The linearization procedure for both the lateral and angular dynamics are achieved by using the small angle approximation for the steering angle δ and both tire slip angles α_f and α_r

$$cos(\delta) \approx 1 \text{ and } tan(\alpha_i) \approx \alpha_i$$

Using these approximations in eq.1.12 yields

$$\dot{V}_y = \frac{C_f\left(\delta - \frac{V_y + L_f\dot{\psi}}{V_x}\right) - C_r\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right) - m\dot{\psi}V_x}{m}$$

Then collecting terms

$$\dot{V}_y = \frac{-(C_f + C_r)V_y}{mV_x} + \frac{(C_f L_f - C_r L_r - mV_x^2)\dot{\psi}}{mV_x} + \frac{C_f\delta}{m} \tag{1.15}$$

Now the same for eq. 1.13 yields

$$\ddot{\psi} = \frac{L_f C_f\left(\delta - \frac{V_y + L_f\dot{\psi}}{V_x}\right) + L_r C_r\left(\frac{V_y - L_r\dot{\psi}}{V_x}\right)}{I_z}$$

Then collecting terms

$$\ddot{\psi} = \frac{(-L_f C_f + L_r C_r)V_y}{I_z V_x} - \frac{\left(L_f^2 C_f + L_r^2 C_r\right)\dot{\psi}}{I_z V_x} + \frac{L_f C_f\delta}{I_z} \tag{1.16}$$

Now in state space representation

$$\begin{bmatrix} V_y \\ \dot{V}_y \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(C_f + C_r)}{mV_x} & 0 & \frac{-mV_x{}^2 - (L_f C_f - L_r C_r)}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-(L_f C_f - L_r C_r)}{I_z V_x} & 0 & \frac{-(L_f{}^2 C_f + L_r{}^2 C_r)}{I_z V_x} \end{bmatrix} \begin{bmatrix} y \\ V_y \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{C_f}{m} \\ 0 \\ \frac{L_f C_f}{I_z} \end{bmatrix} \delta \qquad (1.17)$$

With the following states

- y : Lateral position w.r.t center of rotation of body frame
- $\dot{y}$: Lateral velocity (V_y) w.r.t center of rotation of body frame
- ψ : Yaw angle w.r.t world frame
- $\dot{\psi}$: Angular velocity w.r.t world frame

This linear model is the classically known planar bicycle car model used very widely in academia and industry [1]-[7]. This model is used for making predictions on vehicle behavior but is still missing tracking capabilities in terms of path following. This feature is what is sought out next.

1.2.3.2.1 **With Respect to Side-Slip Angle**

This model is useful for verifying parameter estimation which is discussed in more detail in the system identification section.

$$\begin{bmatrix} \dot{\beta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{-(C_f + C_r)}{mV_x} & \frac{-mV_x{}^2 - (L_f C_f - L_r C_r)}{mV_x{}^2} \\ \frac{-(L_f C_f - L_r C_r)}{I_z} & \frac{-(L_f{}^2 C_f + L_r{}^2 C_r)}{I_z V_x} \end{bmatrix} \begin{bmatrix} \beta \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{C_f}{mV_x} \\ \frac{L_f C_f}{I_z} \end{bmatrix} \delta \qquad (1.18)$$

1.2.3.2.2 **Path Consideration**

For path tracking purposes, eq.1.17 can be transformed such that the state variables are in terms of position and orientation error of the desired path to be followed [1],[3],[5] as shown in

Figure 6. The states will be given the following new nomenclature for clarity between the two models.

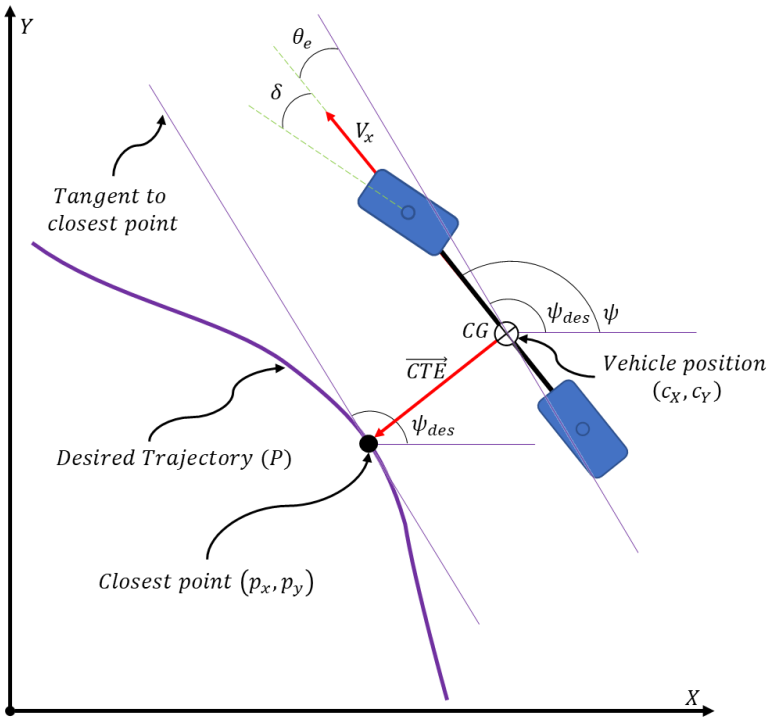

Figure 6: Dynamic model with respect to desired trajectory

With the following states

- e_y Lateral position error (cross track error) of body frame CG with the desired position in the world frame
- $\dot{e}_y$ Lateral velocity error of body frame CG with the desired velocity in the world frame
- e_ψ Yaw angle error of body frame x-axis with the desired path angle in the world frame
- $\dot{e}_\psi$ Yaw rate error of body frame x-axis with the desired path yaw rate in the world frame

The desired lateral position is user defined but measuring it is discussed in the following section. The desired yaw rate is defined by the speed of the car and the radius R (or curvature κ) of the trajectory to track as shown

$$\dot{\psi}_{des} = \frac{V_x}{R} = V_x \kappa \tag{1.19}$$

The desired acceleration is formulated as the normal acceleration component

$$\ddot{y}_{des} = \frac{V_x^2}{R} = V_x \dot{\psi}_{des} \tag{1.20}$$

Now since the desired acceleration and yaw rate have been defined, they must be correlated with the vehicles current states to obtain their error forms. To do this, relative dynamics of a translating and rotating body must be used because the vehicle has been defined in its own reference body frame and the desired trajectory to follow is in the global coordinate frame. This will breakdown the state vectors into their normal and tangential components. The velocity and acceleration of the vehicle with respect to the global coordinate frame is then

$$V_y = \dot{y} + V_x \psi$$

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

Now obtaining the new error state equations

$$e_\psi = \psi - \psi_{des} \tag{1.21}$$

$$\dot{e}_\psi = \dot{\psi} - \dot{\psi}_{des} \tag{1.22}$$

$$\dot{e}_y = \dot{y} + V_x e_\psi \tag{1.23}$$

$$\ddot{e}_y = \ddot{y} + V_x \dot{e}_\psi \tag{1.24}$$

Eliminating V_y and $\dot{\psi}$ from the original state equations by plugging in 1.21-1.24 into 1.15 and 1.16 yields the following

$$m\ddot{e}_y = -\left(\frac{C_f + C_r}{V_x}\right)\dot{e}_y + (C_f + C_r)e_\psi - \left(\frac{L_f C_f - L_r C_r}{V_x}\right)\dot{e}_\psi + C_f\delta - \left(\frac{L_f C_f - L_r C_r}{V_x}\right)\dot{\psi}_{des}$$

$$I_z\ddot{e}_\psi = -\left(\frac{L_f C_f - L_r C_r}{V_x}\right)\dot{e}_y + (L_f C_f - L_r C_r)e_\psi - \left(\frac{{L_f}^2 C_f + {L_r}^2 C_r}{V_x}\right)\dot{e}_\psi + L_f C_f\delta$$

$$-\left(\frac{{L_f}^2 C_f + {L_r}^2 C_r}{V_x}\right)\dot{\psi}_{des} - I_z\ddot{\psi}_{des}$$

Assuming constant speed along the body x-axis $(\ddot{\psi}_{des} = 0)$, then the new state space equations become

$$\begin{bmatrix}\dot{e}_y\\ \ddot{e}_y\\ \dot{e}_\psi\\ \ddot{e}_\psi\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{-(C_f + C_r)}{mV_x} & \frac{(C_f + C_r)}{m} & \frac{-(L_f C_f - L_r C_r)}{mV_x}\\ 0 & 0 & 0 & 1\\ 0 & \frac{-(L_f C_f - L_r C_r)}{I_z V_x} & \frac{(L_f C_f - L_r C_r)}{I_z} & \frac{-({L_f}^2 C_f + {L_r}^2 C_r)}{I_z V_x}\end{bmatrix}\begin{bmatrix}e_y\\ \dot{e}_y\\ e_\psi\\ \dot{e}_\psi\end{bmatrix} + \begin{bmatrix}0\\ \frac{C_f}{m}\\ 0\\ \frac{L_f C_f}{I_z}\end{bmatrix}\delta$$

$$+\begin{bmatrix}0\\ \frac{-(L_f C_f - L_r C_r + mV_x^2)}{mV_x}\\ 0\\ \frac{-({L_f}^2 C_f + {L_r}^2 C_r)}{I_z V_x}\end{bmatrix}\dot{\psi}_{des} \qquad (1.25)$$

$$\dot{x} = Ax + B_\delta\delta + B_{\dot{\psi}}\dot{\psi}_{des}$$

Notice that the model has V_x within the A and B matrices, this means that if the speed of the car is to change, the model needs to be updated which implies that this dynamic model is parameter-varying. Such a form is known as a Linear-Parameter-Varying (LPV) form.

1.2.3.2.3 **Road Grade Consideration**

If on a non-flat track, the force of gravity now effects 1.15 and can be adjusted by the new form

$$\dot{V}_y = \frac{-(C_f + C_r)V_y}{mV_x} + \frac{(C_f L_f - C_r L_r - mV_x{}^2)\dot{\psi}}{mV_x} + \frac{C_f \delta}{mV_x} + mgsin(\phi) \qquad (1.26)$$

Where g is the acceleration from gravity and ϕ is the road bank angle as shown in Figure 7. The yaw dynamics of the vehicle are not affected by road bank angle and remain the same.

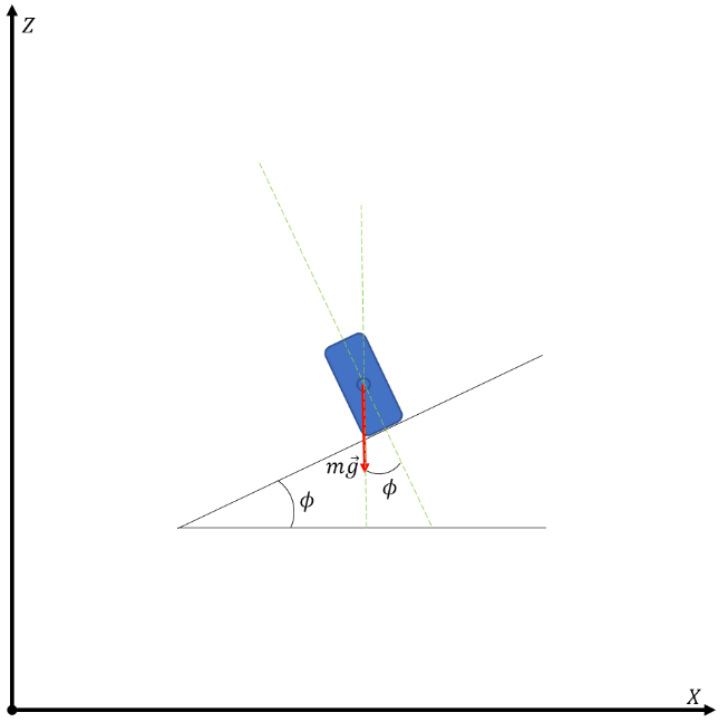

Figure 7: Road grade consideration

1.2.3.3 **Methods for Measuring Cross Track Error**

There are several different methods for calculating the cross-track error ($\overrightarrow{CTE}$) [2]. A function-based approach Linear Function Interpolation (LFI) (see **Error! Reference source not found.**) and a vector-based approach Linear Path Interpolation (LPI) are described below but was found that the LPI was more stable due to possible infinite or zero slopes that could arise in the LFI method.

The magnitude of the $\overrightarrow{CTE}$ was calculated using linear functions and its sign was determined by the sign of the function's derivative. The high-level procedure called Linear Function Interpolation (LFI) is first defined followed by a formal derivation.

LFI Procedure:

1. Calculate the slope and y-intercepts from the nearest points $(p_1 \; and \; p_2)$ on the desired trajectory where p_1 is the closest point and p_2 is the next point that respects the orientation of the path

$$m_P = \frac{Y_{p2} - Y_{p1}}{X_{p2} - X_{p1}} \text{ and } b_P = Y_{p1} - X_{p1} m_p$$

2. Create a linear function (P) using results from step 1

$$P = m_P X + b_P$$

3. Create another linear function (Q) this is perpendicular to *P* and intersects the *CG* of the vehicle (p_{CG})

$$\frac{dQ}{dx} = m_Q = \frac{-1}{m_P} \text{ and } b_Q = Y_{CG} - X_{CG} m_Q$$

$$Q = m_Q X + b_Q$$

4. Determine the intersection point (p_{CTE}) of the P and Q functions

$$P = Q$$

$$m_P X_{CTE} + b_P = m_Q X_{CTE} + b_Q$$

$$X_{CTE} = \frac{b_P - b_Q}{m_Q - m_P}$$

$$Y_{CTE} = m_Q X_{CTE} + b_Q$$

5. Calculate the magnitude of the $\overrightarrow{CTE}$ from the absolute distance from p_{CG} to p_{CTE}

$$\left\|\overrightarrow{CTE}\right\| = \sqrt{(X_{CTE} - X_{CG})^2 + (Y_{CTE} - Y_{CG})^2}$$

6. Calculate the sign of the $\overrightarrow{CTE}$ by evaluating the sign the derivative of Q

$$sign\left(\overrightarrow{CTE}\right) = sign\left(\frac{dQ}{dx}\right)$$

7. The $\overrightarrow{CTE}$ is then calculated as the product of steps 5 and 6

$$\overrightarrow{CTE} = sign(CTE)\left\|\overrightarrow{CTE}\right\|$$

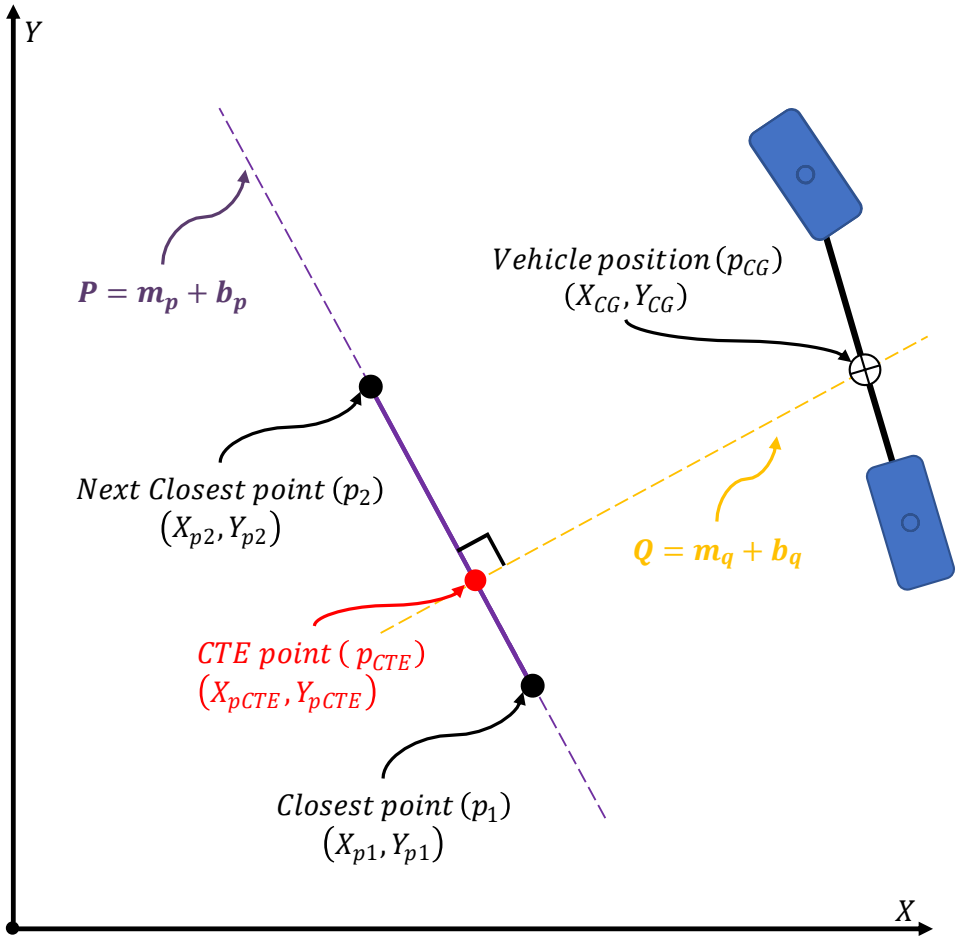

Figure 8: LFI method for calculating cross track error

LPI Procedure:

1. Create a unit vector $\left(\vec{\hat{P}}_{p1p2}\right)$ from the two nearest points (p_1 and p_2) on the desired trajectory where p_1 is the closest point and p_2 is the next point in the path that respects the orientation of the path

$$\vec{\hat{P}}_{p1p2} = \frac{(X_{p2} - X_{p1}, Y_{p2} - Y_{p1})}{\|\vec{P}_{12}\|}$$

2. Create a normalized orthogonal vector $\left(\vec{\hat{P}}^{\perp}_{p1p2}\right)$ from $\vec{\hat{P}}_{p1p2}$ (apply rotation with $\theta = \pi/2$)

$$\vec{\hat{P}}^{\perp}_{p1p2} = \begin{bmatrix} cos\,\theta & -sin\,\theta \\ sin\,\theta & cos\,\theta \end{bmatrix} \vec{\hat{P}}_{p1p2}$$

3. Create a vector $\left(\vec{V}_{cgp1}\right)$ from p_{CG} to p_1

$$\vec{V}_{cgp1} = \left(X_{CG} - X_{p1}, Y_{CG} - Y_{p1}\right)$$

4. The $\left\|\overrightarrow{CTE}\right\|$ and its sign (path being to the left or right of vehicle) is the result of the following dot product

$$\left\|\overrightarrow{CTE}\right\| = \overrightarrow{\hat{P}}^{\perp}_{p1p2} \cdot \overrightarrow{V}_{cgp1}$$

5. The coordinate of the cross-track error point (p_{CTE}) is the result of scaling $\vec{\hat{P}}_{p1p2}$ by the following dot product

$$p_{CTE} = (X_{CTE}, Y_{CTE}) = \overrightarrow{\hat{P}}_{p1p2}\left(\vec{\hat{P}}_{p1p2} \cdot \overrightarrow{V}_{cgp1}\right)$$

6. The $\overrightarrow{CTE}$ is then calculated as the difference between p_{CTE} and p_{CG}

$$\overrightarrow{CTE} = (p_{CTE} - p_{CG}) = (X_{CTE} - X_{CG}, Y_{CTE} - Y_{CG})$$

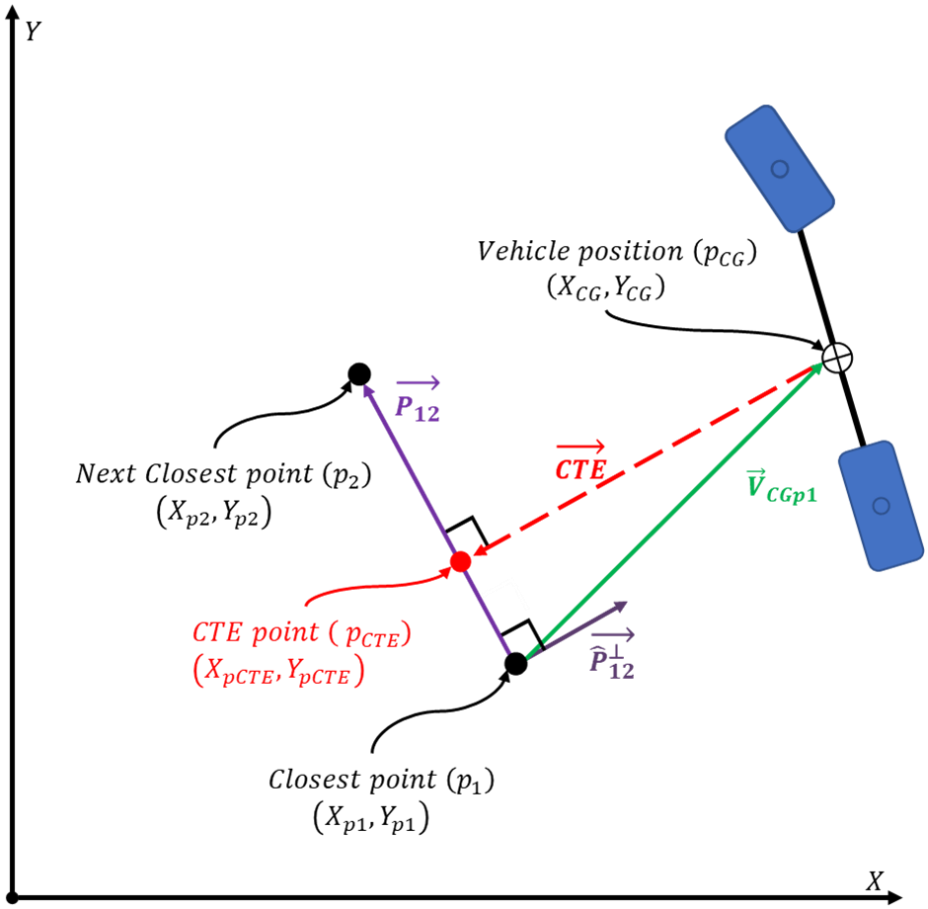

Figure 9: LPI method for calculating cross track error

1.3 **System Identification**

1.3.1 **Introduction**

The focus of this section is on the plant block and its dynamics, owing to the presence of certain unknown parameters that need to be defined. Upon determining these parameters, a complete understanding of the plant block and the models employed to describe it can be attained, paving the way for the consideration of state estimation and control. The only goal in this section is to establish values for the tire stiffness coefficients through system identification.

Most of the parameters can be closely approximated by measuring the vehicle's mass. Depending on the scales available, the total mass can be determined by weighing the front and rear axles or each tire individually, then summing them.

$$m_f = m_{fr} + m_{fl}$$

$$m_r = m_{rr} + m_{rl}$$

$$m = m_f + m_r \tag{1.27}$$

The distances from the CG are estimated by the weight distribution and the actual length of the vehicle which was measured simply with a measuring tape

$$L_f = L\left(1 - \frac{m_f}{m}\right) \tag{1.28}$$

$$L_r = L\left(1 - \frac{m_r}{m}\right) \tag{1.29}$$

The moment of Inertia is estimated by lumping the mass on the front and rear axles as individual particles that are connected by a massless rod which is calculated as shown below

$$I_z = m_f L_f^2 + m_r L_r^2 \tag{1.30}$$

1.3.2 **Least Squares**

Now the only unknown parameters in the dynamic model given in 1.2.3.2 (shown below for convenience) are the tire stiffness coefficients. This model is special as it can be reformed in terms of the unknown parameters which leads to the idea of performing linear regression or "least squares" to fit a line through the collected data (experiment setup and results are shown in 2.4.2.1) with the tire stiffness coefficients being the parameters to adjust the shape of this fit [5]-[7], [11].

$$\begin{bmatrix} \dot{V}_y \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{-(C_f + C_r)}{mV_x} & \frac{-mV_x{}^2 - (L_f C_f - L_r C_r)}{mV_x} \\ \frac{-(L_f C_f - L_r C_r)}{I_z V_x} & \frac{-\left(L_f^2 C_f + L_r^2 C_r\right)}{I_z V_x} \end{bmatrix} \begin{bmatrix} V_y \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{C_f}{m} \\ \frac{L_f C_f}{I_z} \end{bmatrix} \delta$$

$$\dot{V}_y = \frac{-(C_f + C_r)V_y}{mV_x} + \frac{\left(-mV_x{}^2 - (L_f C_f - C_r L_r)\right)\dot{\psi}}{mV_x} + \frac{C_f \delta}{m}$$

$$\ddot{\psi} = \frac{-(L_f C_f - L_r C_r)V_y}{I_z V_x} - \frac{\left(L_f^2 C_f + L_r^2 C_r\right)\dot{\psi}}{I_z V_x} + \frac{L_f C_f \delta}{I_z}$$

$$\dot{V}_y = -\frac{V_y}{mV_x} C_f - \frac{V_y}{mV_x} C_r - \frac{L_f \dot{\psi}}{mV_x} C_f + \frac{L_r \dot{\psi}}{mV_x} C_r - V_x \dot{\psi} + \frac{\delta}{m} C_f$$

$$\ddot{\psi} = -\frac{L_f V_y}{I_z V_x} C_f + \frac{L_r V_y}{I_z V_x} C_r - \frac{L_f^2 \dot{\psi}}{I_z V_x} C_f - \frac{L_r^2 \dot{\psi}}{I_z V_x} C_r + \frac{L_f \delta}{I_z} C_f$$

Combining coefficients

$$\dot{V}_y = \left(\frac{\delta V_x - V_y - L_f \dot{\psi}}{mV_x}\right) C_f + \left(\frac{L_r \dot{\psi} - V_y}{mV_x}\right) C_r - V_x \dot{\psi}$$

$$\ddot{\psi} = \left(\frac{L_f \delta V_x - L_f V_y - L_f^2 \dot{\psi}}{I_z V_x}\right) C_f + \left(\frac{L_r V_y - L_r^2 \dot{\psi}}{I_z V_x}\right) C_r$$

Vectorizing

$$\begin{bmatrix} \dot{V}_y + V_x \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \left(\frac{\delta V_x - V_y - L_f \dot{\psi}}{mV_x}\right) & \left(\frac{L_r \dot{\psi} - V_y}{mV_x}\right) \\ \left(\frac{L_f \delta V_x - L_f V_y - L_f^2 \dot{\psi}}{I_z V_x}\right) & \left(\frac{L_r V_y - L_r^2 \dot{\psi}}{I_z V_x}\right) \end{bmatrix} \begin{bmatrix} C_f \\ C_r \end{bmatrix}$$

Now discretizing using Euler

$$\frac{dx}{dt} = G(x)$$

$$G(x) = \frac{dx}{dt} \approx \frac{\Delta x}{\Delta t} = \frac{x(t + \Delta t) - x(t)}{\Delta t}$$

$$x(t + \Delta t) - x(t) = G(x)\Delta t$$

Then

$$\begin{bmatrix} V_y(k+1) - V_y(k) + V_x(k)\dot{\psi}(k)\Delta t \\ \dot{\psi}(k+1) - \dot{\psi}(k) \end{bmatrix}$$

$$= \begin{bmatrix} \left(\frac{\delta(k)V_x(k) - V_y(k) - L_f\dot{\psi}(k)}{mV_x(k)}\right)\Delta t & \left(\frac{L_r\dot{\psi}(k) - V_y(k)}{mV_x(k)}\right)\Delta t \\ \left(\frac{L_f\delta(k)V_x(k) - L_fV_y(k) - L_f^2\dot{\psi}(k)}{I_zV_x(k)}\right)\Delta t & \left(\frac{L_rV_y(k) - L_r^2\dot{\psi}(k)}{I_zV_x(k)}\right)\Delta t \end{bmatrix} \begin{bmatrix} C_f \\ C_r \end{bmatrix}$$

Where *k* is a single sample data point. Now for an entire sequence of *N* data points from an experiment, let

$$Y = \begin{bmatrix} V_y(k+1) - V_y(k) + V_x(k)\dot{\psi}(k)\Delta t \\ \dot{\psi}(k+1) - \dot{\psi}(k) \\ \vdots \\ \vdots \\ V_y(N) - V_y(N-1) + V_x(N)\dot{\psi}(N)\Delta t \\ \dot{\psi}(N+1) - \dot{\psi}(N) \end{bmatrix}$$

and

$$\Phi = \begin{bmatrix} \left(\frac{\delta(k) - V_y(k) - L_f\dot{\psi}(k)}{mV_x(k)}\right)\Delta t & \left(\frac{L_r\dot{\psi}(k) - V_y(k)}{mV_x(k)}\right)\Delta t \\ \left(\frac{L_f\delta(k) - L_fV_y(k) - L_f^2\dot{\psi}(k)}{I_zV_x(k)}\right)\Delta t & \left(\frac{L_rV(k)_y - L_r^2\dot{\psi}(k)}{I_zV_x(k)}\right)\Delta t \\ \vdots & \vdots \\ \vdots & \vdots \\ \left(\frac{\delta(N) - V_y(N) - L_f\dot{\psi}(N)}{mV_x(N)}\right)\Delta t & \left(\frac{L_r\dot{\psi}(N) - V_y(N)}{mV_x(N)}\right)\Delta t \\ \left(\frac{L_f\delta(N) - L_fV_y(N) - L_f^2\dot{\psi}(N)}{I_zV_x(N)}\right)\Delta t & \left(\frac{L_rV(k)_y - L_r^2\dot{\psi}(N)}{I_zV_x(N)}\right)\Delta t \end{bmatrix}$$

Or in compact form

$$Y = \begin{bmatrix} y^T(k) \\ \vdots \\ y^T(N) \end{bmatrix} \text{ and } \Phi = \begin{bmatrix} \varphi(k) \\ \vdots \\ \varphi(N) \end{bmatrix}$$

Where,

$$y^T(k) = \begin{bmatrix} V_y(k+1) - V_y(k) + V_x(k)\dot{\psi}(k)\Delta t & \dot{\psi}(k+1) - \dot{\psi}(k) \end{bmatrix}$$

$$\varphi(k) = \begin{bmatrix} \left(\frac{\delta(k) - V_y(k) - L_f\dot{\psi}(k)}{mV_x(k)}\right)\Delta t & \left(\frac{L_r\dot{\psi}(k) - V_y(k)}{mV_x(k)}\right)\Delta t \\ \left(\frac{L_f\delta(k) - L_fV_y(k) - L_f^2\dot{\psi}(k)}{I_zV_x(k)}\right)\Delta t & \left(\frac{L_rV(k)_y - L_r^2\dot{\psi}(k)}{I_zV_x(k)}\right)\Delta t \end{bmatrix}$$

Note: Φ is a 2Nx2 and Y is a 2Nx1

Then θ

$$Y = \Phi\theta$$

$$\theta = (\Phi^T\Phi)^{-1}\Phi^T Y$$

$$\theta = \left(\begin{bmatrix} \varphi(k) \\ \vdots \\ \varphi(N) \end{bmatrix}^T \begin{bmatrix} \varphi(k) \\ \vdots \\ \varphi(N) \end{bmatrix}\right)^{-1} \begin{bmatrix} \varphi(k) \\ \vdots \\ \varphi(N) \end{bmatrix}^T \begin{bmatrix} y^T(k) \\ \vdots \\ y^T(N) \end{bmatrix} \tag{1.31}$$

This is the least-squares solution where:

- Φ is the regressor matrix
- Y is the output matrix
- θ is the parameter vector

1.4 **Optimal State Estimation**

1.4.1 **Introduction**

The next focus of study is the estimator block, which aims to minimize the error between the plant block (real system) and its models. The approach is to use the model to make predictions, compare them with sensor measurements, and then multiply the difference by a gain that minimizes the prediction and measurement. Calculation of the gain is based on sensor and model noise, which results from assumptions made in the model. The first goal to achieve is to be able to understand how the Kalman filter can fuse various sensor measurements together with a vehicle model to provide a more accurate awareness of the vehicle's orientation and location with respect to some reference trajectory and the next goal is to understand the procedure for using the Kalman filter in its various forms.

For effective vehicle control, many system properties that vary with time and speed must be known but can be challenging to measure or calibrate. Kalman filters provide optimal estimates of the vehicle's properties and states. The estimator block is depicted in Figure 10. The Kalman filter is not derived here but can be solved through duality theory [9] which is described later in 1.5 This insight facilitates the understanding of both control and estimation concepts, indicating that the two processes are the inverses of one another. Tuning the estimator will be discussed in the final section and the concept of observability is explored first.

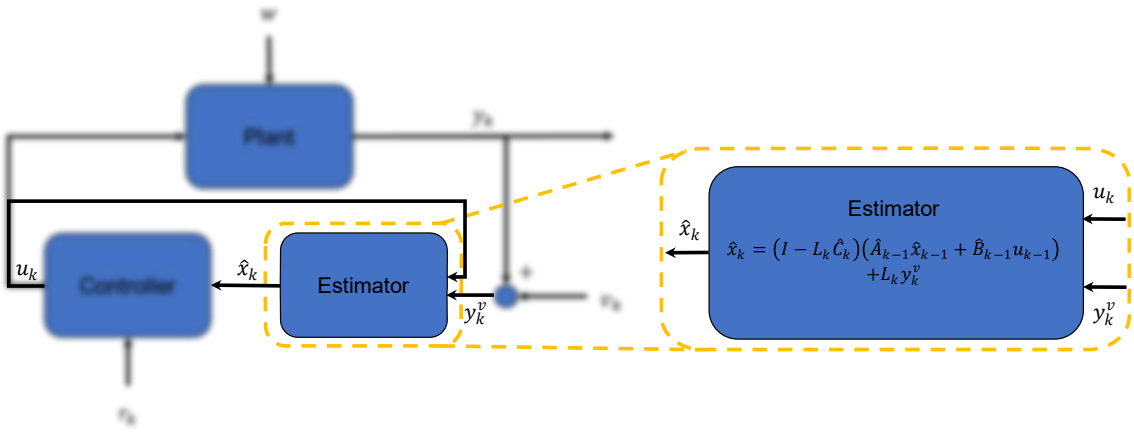

Figure 10: Estimator block detailed view

1.4.2 **Observability**

Observability is a property of a system that can determine the initial state from future inputs and outputs. This means that if some of the state variables in the system cannot be measured directly with available sensors, but the system is observable, then it is possible to estimate what the unmeasurable states would be provided the available input and output data of the system. The observability matrix and Gramian are tools for determining if a system is observable and evaluating the cost for estimating the states of the system [13]-[14]. Next is defining the observability Gramian, which will be used in the filtering process discussed in the next section.

For Continuous-Time System

$$W = \int_0^t \left(\Phi(t,\tau)^T C^T C \Phi(t,\tau)\right) d\tau \tag{1.32}$$

Discrete-Time System

$$W = \sum_{\tau=0}^{\tau=t} \Phi(t,\tau)^T C^T C \Phi(t,\tau)$$

Observability criteria can be evaluated with either the observability matrix $\left(O(A,C)\right)$ or the PBH (Popov–Belevitch–Hautus) rank tests and is handled the same for continuous and discrete time systems.

$$O(A,B) = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} = \begin{bmatrix} C^T & A^TC^T & A^{T^2}C^T & \cdots & A^{T^{n-1}}C^T \end{bmatrix}^T$$

$$rank[O(A,C)] = rank\begin{bmatrix} A-\lambda I \\ C \end{bmatrix} = \begin{bmatrix} (A-\lambda I)^T & C^T \end{bmatrix}^T = n$$

With the following definitions

- A is the state matrix of the system
- C is the output matrix of the system
- λ are the eigen values of A
- n is the size of the square matrix (number of rows=numbers of columns) A

The matrix C classifies which of the states x that can be measured. This can vary depending on what sensors are available. For example, if only e_y can be measured, then C would have the following form

$$C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

Another example is if both e_y and e_ψ can be measured, then C would have the following form

$$C_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

If the rank of $O(A,C) = n$ then the system is said to be observable. Using eq.(1.17) for A (giving n=4) and C_2 from the example above, along with eq.(1.27)-(1.30) and the following values for the remaining analysis in this thesis

$$m_f = 320kg \; and \; m_r = 380kg$$

$$L = 2.9718m$$

$$V_x = [1{:}\,100]\,\frac{N}{rad}$$

$$C_f = 59800\,\frac{N}{rad}\; and\; C_r = 63200\,\frac{N}{rad}$$

Yielded a rank 4 system which is equal to n for the entire range of V_x, meaning that the system is observable. This means that even using C_2 as the measurement matrix, the entire state vector can be reconstructed, providing estimates for the entire state vector x. This now introduces where the Kalman filter comes in to calculate the optimal state estimate.

1.4.3 **State Estimators**

The Kalman filter is a mathematical tool used to estimate the state of a system based on noisy observations. It is designed to work with linear systems that have Gaussian noise, and it provides the optimal estimate of the system state under these assumptions. The linearized Kalman filter is an extension of the Kalman filter that can be used for nonlinear systems by linearizing the system around the current state estimate. The Extended Kalman filter is a further extension that can be used for nonlinear systems by using the systems non-linear models during the update processes. These filters are widely used in engineering and science for a variety of applications, such as control and navigation. The proceeding sections highlight how these filters are defined mathematically and how they differ from one another.

1.4.3.1 **Kalman Filter**

This is a linear time varying filter designed for linear systems. The procedure for the Kalman filter is sequenced by a measurement update from available sensors and then a time update to the model of the system [9-12]. The subscripts k represent the quantity for the discretized system at that sample time. The procedure is shown below.

Signal Model

$$x_{k+1} = A_k x_k + B_k u_k + w_k$$

$$y_k = C_k x_k + v_k$$

Time Update

$$\hat{x}_{k+1|k} = A_k \hat{x}_{k|k} + B_k u_k$$

$$P_{k+1|k} = A_k P_{k|k} A_k^T + Q_k$$

Measurement Update

$$K_k = P_{k|k-1} C_k^T \left(C_k P_{k|k-1} C_k^T + R_k \right)^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \left(y_k^m - C_k \hat{x}_{k|k-1} \right)$$

$$P_{k|k} = P_{k|k-1} - P_{k|k-1} C_k^T \left(C_k P_{k|k-1} C_k^T + R_k \right)^{-1} C_k P_{k|k-1}$$

Or in a more compact form

$$K_k = P_{k|k-1} C_k^T \left(C_k P_{k|k-1} C_k^T + R_k \right)^{-1}$$

$$\hat{x}_{k|k} = (I_n - K_k C_k)\left(A_{k-1} \hat{x}_{k-1|k-1} + B_{k-1} u_{k-1} \right) + K_k y_k^m$$

$$P_{k|k} = (I - K_k C_k) P_{k|k-1}$$

With the following definitions

- k is the step index
- A_k system dynamics matrix
- B_k input matrix
- C_k observation matrix of which states are actively being measured
- u_k is the input (*ie.* δ_k)
- y_k^m is the unfiltered measurement
- $\hat{x}_{k|k}$ filtered state estimate
- $\hat{x}_{k+1|k}$ predicted state estimate
- w_k process noise
- v_k measurement noise
- Q_k process noise covariance matrix
- R_k measurement noise covariance matrix
- $P_{k|k}$ filtered covariance matrix
- $P_{k+1|k}$ predicted covariance matrix
- K_k Kalman gain

An example iteration is shown below starting with the initial guess/measurement

$$\hat{x}_{0|-1} = A_{-1}\hat{x}_{-1|-1} + B_{-1}u_{-1}$$

at step $k = 0$

$$K_0 = P_{0|-1}C_0^T\left(C_0P_{0|-1}C_0^T + R_0\right)^{-1}$$

$$\hat{x}_{0|0} = \hat{x}_{0|-1} + K_0\left(y_0 - C_0\hat{x}_{0|-1}\right)$$

$$P_{0|0} = (I - K_0C_0)P_{0|-1}$$

at step $k = 1$

$$K_1 = P_{1|0}C_1^T\left(C_1P_{1|0}C_1^T + R_1\right)^{-1}$$

$$\hat{x}_{1|1} = (I_n - K_1C_1)\left(A_0\hat{x}_{0|0} + B_0u_0\right) + K_1y_1$$

$$P_{1|1} = (I - K_1C_1)P_{1|0}$$

at step $k = 2$

$$K_2 = P_{2|1}C_2^T\left(C_2P_{2|1}C_2^T + R_2\right)^{-1}$$

$$\hat{x}_{2|2} = (I_n - K_2C_2)\left(A_1\hat{x}_{1|1} + B_1u_1\right) + K_2y_2$$

$$P_{2|2} = (I - K_2C_2)P_{2|1}$$

at step $k = 3$

$$K_3 = P_{3|2}C_3^T\left(C_3P_{3|2}C_3^T + R_3\right)^{-1}$$

$$\hat{x}_{3|3} = A_2\hat{x}_{2|2} + B_2u_2 + K_3\left(y_3 - C_2\left(A_2\hat{x}_{2|2} + B_2u_2\right)\right)$$

$$P_{3|3} = (I - K_3C_3)P_{3|2}$$

$$\vdots$$

The Kalman filter is a recursion, meaning that the state estimate and covariance at step k need to be stored and used on step $k + 1$ to properly converge to the optimal solution.

1.4.3.2 **Linearized Kalman Filter**

This is a linearly time varying filter designed for nonlinear systems. The procedure for the Linearized Kalman Filter (LKF) is very similar to the standard Kalman filter but allows for state estimation of nonlinear system dynamics and measurements which requires some additional care in the setup procedure. The difference is replacing the nonlinear equations simply with their linearization around a nominal point but is solved identically to the linear Kalman filter. The procedure is shown below.

Signal Model

$$x_{k+1} = f_k(x_k, u_k, w_k)$$

$$y_k = h_k(x_k, v_k)$$

Linearization

$$F_k = \frac{\partial f_k}{\partial x}|_{x=\bar{x}} \quad G_k = \frac{\partial f_k}{\partial u}|_{u=\bar{u}} \quad L_k = \frac{\partial f_k}{\partial w}|_{w=\bar{w}}$$

$$H_k = \frac{\partial h_k}{\partial x}|_{x=\bar{x}} \qquad\qquad M_k = \frac{\partial h_k}{\partial v}|_{v=\bar{v}}$$

Time Update

$$\hat{x}_{k+1|k} = F_k(x_k - \bar{x}) + G_k(u_k - \bar{u})$$

$$P_{k+1|k} = F_k P_{k|k} F_k^T + L_k Q_k L_k^T$$

Measurement Update

$$K_k = P_{k|k-1} H_k^T \left(H_k P_{k|k-1} H_k^T + M_k R_k M_k^T\right)^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\left(y_k^m - h_k\left(\hat{x}_{k|k-1}, v_k = 0\right)\right)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

With the following new definitions

- F_k is the linearization of f_k with respect to x_k

- G_k is the linearization of f_k with respect to u_k
- L_k is the linearization of f_k with respect to w_k
- H_k is the linearization of h_k with respect to x_k
- M_k is the linearization of h_k with respect to v_k

1.4.3.3 **Extended Kalman Filter**

The procedure for the Extended Kalman Filter (EKF) is very similar to the KF but differs by using the nonlinear system dynamics and measurements in both time and measurement updates for the state calculation and the calculations for the Kalman gain and covariance matrix use the Jacobian of the non-linear state and output equations. The procedure is shown below

Signal Model

$$x_{k+1} = f_k(x_k, u_k, w_k)$$

$$y_k = h_k(x_k, v_k)$$

Jacobian of dynamics, measurement, and noise models

$$F_k = \frac{\partial f_k}{\partial x}|_{x=\hat{x}_{k|k}} \qquad L_k = \frac{\partial f_k}{\partial w}|_{x=\hat{x}_{k|k}}$$

$$H_k = \frac{\partial h_k}{\partial x}|_{x=\hat{x}_{k|k-1}} \qquad M_k = \frac{\partial h_k}{\partial v}|_{x=\hat{x}_{k|k-1}}$$

Time Update

$$\hat{x}_{k+1|k} = f_k(x_k, u_k, w_k = 0)$$

$$P_{k+1|k} = F_k P_{k|k} F_k^T + L_k Q_k L_k^T$$

Measurement Update

$$K_k = P_{k|k-1} H_k^T \left(H_k P_{k|k-1} H_k^T + M_k R_k M_k^T\right)^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \left(y_k^m - h_k\left(\hat{x}_{k|k-1}, v_k = 0\right)\right)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

With the following new definitions

- F_k is the Jacobian of f_k with respect to x_k
- L_k is the Jacobian of f_k with respect to w_k
- H_k is the Jacobian of h_k with respect to x_k
- M_k is the Jacobian of h_k with respect to v_k

1.4.4 **Tuning The Estimator**

Tuning a Kalman filter involves adjusting the values of the parameters in the filter to achieve the desired performance in estimating the system states. The two main parameters that need to be tuned are the process noise Q_k and measurement noise R_k covariance matrices. The process noise covariance represents the level of uncertainty in the dynamic model of the system, while the measurement noise covariance represents the level of uncertainty in the measurements obtained from the sensors. One common approach to tuning the Kalman filter is to use trial and error. This involves running simulations of the system with different values of the noise covariances and observing the resulting estimation performance. The noise covariances can be adjusted until the desired level of estimation accuracy is achieved.

It is important to note that tuning a Kalman filter is not usually a one-time task, as the optimal noise covariances may change as the system operates in different conditions. Therefore, periodic re-tuning of the Kalman filter may be necessary to maintain optimal estimation performance.

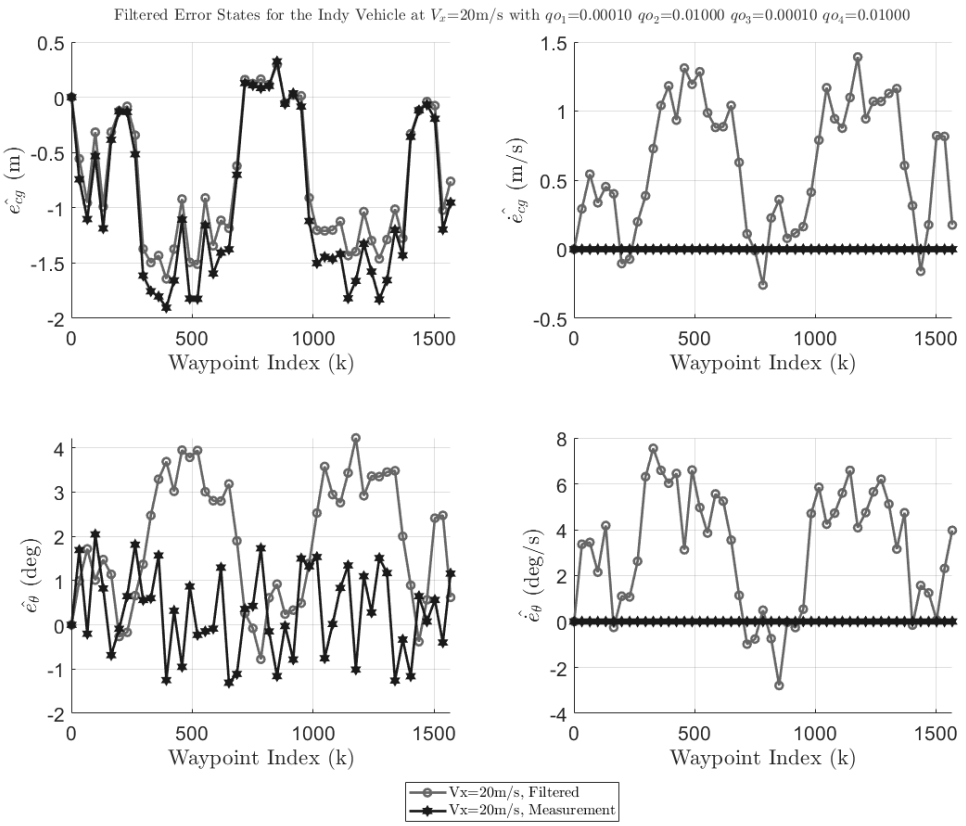

Figure 11: Trial 1 of tuning

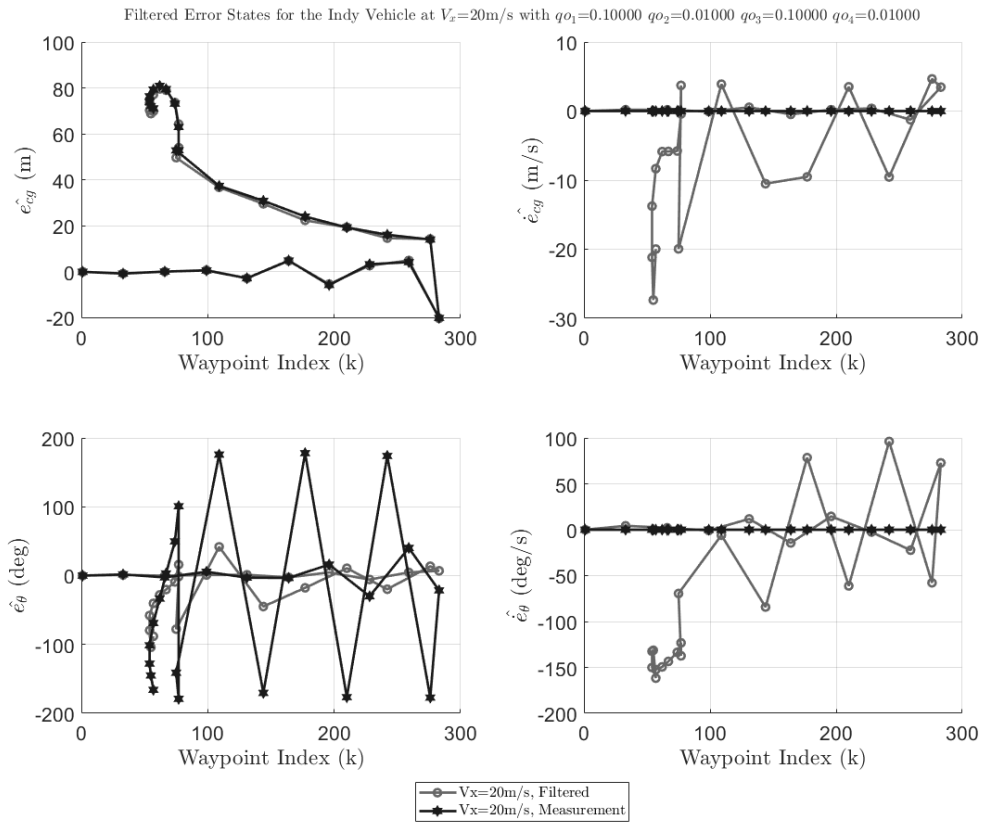

Figure 12: Trial 2 of tuning

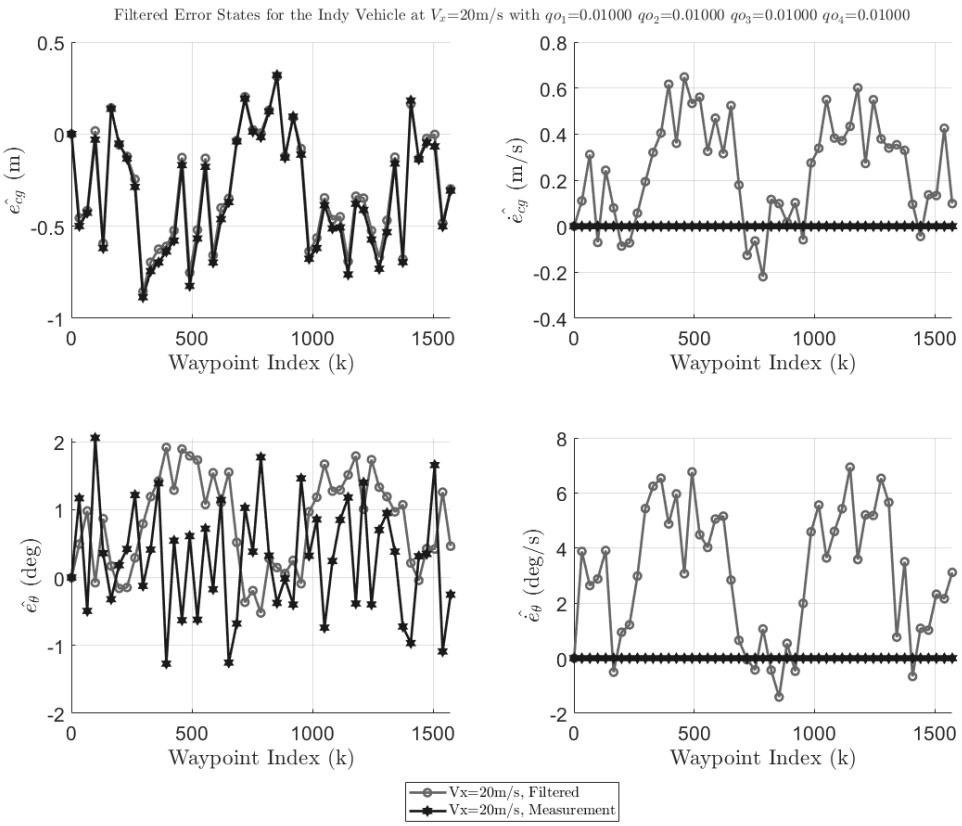

Figure 13: Trial 3 of tuning

1.5 **Optimal Control**

1.5.1 **Introduction**

The final element of the block diagram is the control block seen in Figure 14, which aims to apply an optimal input that minimizes the cost function, comprising of state performance and control effort criteria. The concept is to use the optimal state estimate that is being calculated from the estimator block as the feedback component and then include the important feed-forward term that depends on a reference signal (r_k) to reduce the steady-state errors when executing a turn at a specific reference curvature [1].

The first goal to achieve is to be able to follow a straight path as close as possible, which will be resolved by understanding the feedback component of the control design (how to calculate the gain matrix K). The final goal is to be able to follow arbitrary paths that contain curvature (with the assumption that the reference path has considered the vehicles dynamics during its construction i.e. the vehicles minimum turning radius at various speeds) by adding additional compensation with feed-forward (how to calculate the added steering term δ_{ff}).

The final goal is to be able to follow arbitrary paths that contain curvature (with the assumption that the reference path has considered the vehicles dynamics during its construction i.e. the vehicles minimum turning radius at various speeds.

As mentioned in the optimal estimation section, duality theory is used for deriving the optimal feedback controller and is solved identically as in the estimation problem but in its control form. Tuning the controller will be discussed in the final section and the concept of controllability is explored first as it begins the process of achieving straight path tracking.

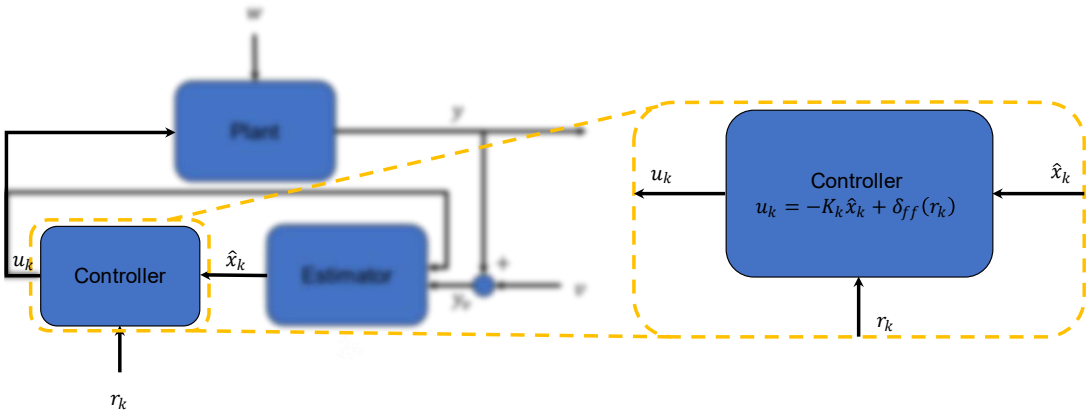

Figure 14: Control block detailed view

1.5.2 **Controllability**

Controllability is a property of a system that can start at any initial state and be able to return to the origin in a finite number of steps. If the states of the system are in terms of the error dynamics from some specified reference, this means that given some initial error condition on the system, it can drive the states back to zero error. The controllability matrix and Gramian are tools for determining if a system is controllable and evaluating the cost of the state performance of the system [13]-[14]. Now defining the controllability Gramian, which will be used in the control process discussed in the next section.

Continuous-Time System

$$X = \int_0^t (\Phi(t,\tau)BB^T\Phi(t,\tau)^T)d\tau \tag{1.33}$$

Discrete-Time System

$$X = \sum_{\tau=0}^{\tau=t} \Phi(t,\tau)BB^T\Phi(t,\tau)^T$$

Controllability criteria can be evaluated with either the controllability matrix $\big(C(A,B)\big)$ or the PBH (Popov–Belevitch–Hautus) rank tests and is handled the same for continuous and discrete time systems.

$$C(A,B) = [B\ AB\ A^2B\ \cdots\ A^{n-1}B]$$

$$rank[C(A,B)] = rank[A - \lambda I \quad B] = n$$

With the following definitions

- A is the state matrix of the system
- B is the input matrix of the system
- λ are the eigen values of A
- n is the size of the square matrix (number of rows=numbers of columns) A

If the rank of $C(A,B) = n$ then the system is said to be controllable. Using eq.(1.17) for A (giving n=4) and B, along with eq.(1.27)-(1.30) and the following values

$$m_f = 320kg\ m_r = 380kg\ L = 2.9718m\ V_x = [1:100]m/s$$

Yielded a rank 4 system which is equal to n for the entire range of V_x, meaning that the system is controllable. This means given the proper input to the system, the state will converge to the origin. This now introduces the next discussion on how to calculate the optimal input.

1.5.3 **Controllers**

In control theory, there are several types of controllers used for controlling a system. The Linear Quadratic Regulator (LQR) controller is a type of optimal controller used for controlling linear systems. The Linear Quadratic Gaussian (LQG) controller is an optimal controller that considers measurement noise and is used for controlling systems with noisy sensors. Both controllers are derived in the following sections with the intent of to achieve the first goal. The system in 1.2.3.2.2 has 2 poles sitting on the edge of the unit disk on the Z-plane (discrete time) or 2 poles sitting at the origin of the S-plane (continuous time) which implies the system is unstable and will be addressed using the controllers in the following sections to push the poles inwards to the origin on the Z-plane or further into the left hand plane in the S-plane.

1.5.3.1 **Linear Quadratic Regulator (Full State Feedback)**

This type of controller would only be implemented if the consideration of both the sensor noise and the errors in the model used to define the plant was negligible or in a noise-free simulation. If either of these components are not negligible then the LQG controller discussed in 0 should be implemented. The infinite-horizon, continuous-time, LQR variant is derived below. The state feedback controller u

$$u(t) = Kx(t)$$

Which minimizes the cost function J

$$J = \int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t)\right) dt$$

With the following definitions

- Q is the semi-definite ($Q \succeq 0$) state performance weight matrix
- R is the positive-definite ($R \succ 0$) control effort weight matrix
- K is the optimal feedback gain matrix
- $x(t)$ is the state of the system at time t
- $u(t)$ is the input for the system at time t $\left(ie. \delta(t)\right)$

The closed loop system and solution

$$\dot{x}(t) = (A + BK)x(t), x(0) = x_0$$

$$x(t) = \Phi(t, \tau)x_0 = e^{(A+BK)t}x_0$$

The closed loop cost is then

$$J = \int_0^\infty \left(x(t)^T Q x(t) + \left(Kx(t)\right)^T R\left(Kx(t)\right)\right) dt$$

$$J = \int_0^\infty \left(x(t)^T (Q + K^T R K) x(t)\right)$$

$$J = x_0^T \left(\int_0^\infty \left(e^{(A+BK)^T t}(Q + K^T R K) e^{(A+BK)t}\right)\right) x_0$$

Now that the cost function is in terms of the controllability Gramian eq. (1.33), J can then be computed as

$$J = x_0^T X\, x_0$$

Where X is the solution to the Lyapunov equation

$$(A + BK)^T X + X(A + BK) + Q + K^T R K = 0$$

The goal now is evaluating the state feedback gain matrix K which is found by completing the squares

$$A^T X + XA - XBR^{-1}B^T X + Q + (XBR^{-1} + K^T)R(R^{-1}B^T X + K) = 0$$

Which identifies the solution for K

$$(XBR^{-1} + K^T)R(R^{-1}B^T X + K) \geq 0$$

$$K = -R^{-1}B^T X \tag{1.34}$$

With this solution for K, it changes the Lyapunov equation to the Algebraic Riccati Equation (ARE) in X

$$A^T X + XA - XBR^{-1}B^T X + Q = 0 \tag{1.35}$$

To proceed, X can be solved via the Hamiltonian matrix (H) which requires an Eigen or Schur decomposition. An Eigen decomposition is not always possible and is generally not used in practice while the Schur decomposition is guaranteed to always exist [13] and is the method explained below.

Making a substitution $Z = BR^{-1}B^T$ in eq.(1.35) and then factoring into the following form

$$A^T X + XA - XZX + Q = 0$$

$$XA + Q - XZX + A^T X = 0$$

$$(-XA - Q)(-I) + (-XZ - A^T)(-X) = 0$$

$$\begin{bmatrix}(-XA - Q) & (-XZ - A^T)\end{bmatrix}\begin{bmatrix}-I\\X\end{bmatrix} = 0$$

$$\left[\begin{bmatrix}-X & I\end{bmatrix}\begin{bmatrix}A\\-Q\end{bmatrix} \quad \begin{bmatrix}-X & I\end{bmatrix}\begin{bmatrix}-Z\\-A^T\end{bmatrix}\right]\begin{bmatrix}-I\\X\end{bmatrix} = 0$$

$$\begin{bmatrix}-X & I\end{bmatrix}\begin{bmatrix}A & -Z\\-Q & -A^T\end{bmatrix}\begin{bmatrix}-I\\X\end{bmatrix} = 0$$

then identifying

$$H = \begin{bmatrix} A & -Z \\ -Q & -A^T \end{bmatrix}$$

now factoring H into the Schur decomposition form

$$H = UTU^T$$

where U is given by

$$U = \begin{bmatrix} U_{11} & U_{12}^* \\ U_{21} & U_{22}^* \end{bmatrix}$$

With the following definitions

- U : Is a full unitary matrix of Schur vectors
    - U_{ij} corresponds to the stable left-hand plane (LHP) eigenvalues in T
    - U_{ij}^* corresponds to the unstable right-hand plane (RHP) eigenvalues in T
- T : Is an upper triangular matrix with the Eigenvalues of H along the main diagonal
- H : Is Hamiltonian which abides the symmetric root property (every pole in the LHP has a corresponding pole in the RHP)

This decomposition must be done carefully such that *T* is ordered from (left-most) LHP Eigenvalues to (right-most) RHP Eigenvalues. This identifies the stable solutions that span the first *n* columns of *U*, which *U* is related to the solution of eq.(1.35) by

$$X = U_{21}U_{11}^{-1} \tag{1.36}$$

Where the matrix block indices are given by

$$U_{11} = U_{1:n,1:n}$$

$$U_{21} = U_{n+1:2n,1:n}$$

1.5.3.2 **Linear Quadratic Gaussian Control (Optimal State Estimate Feedback)**

The linear quadratic gaussian (LQG) controller at its core is simply the LQR controller using the Kalman filtered state estimate for the state feedback controller. Through the dynamic programming principle, the state covariance matrix can be numerically calculated through an iterative process. This is because the value function is a contraction which implies that the solution will converge to a steady state value in a finite number of steps for the finite horizon problem. It is to be noted that the solution is not guaranteed to converge in the infinite horizon problem unless a discounted cost is implemented. This section shows the principle of dynamic programming and the dynamic programming equation [13], [15]. This method results in the same solution for the covariance. The finite-horizon, discrete-time, LQG is used as the stabilizing feedback controller and is defined as follows.

Consider the system of the form

$$f(x,u,w) = \hat{x}_{k+1} = A\hat{x}_k + Bu_k + w_k$$

With the assumption $E[w_k] = 0$ and $E[w_k^T w_k] = W$

The state feedback controller

$u_k = K\hat{x}_k$ Which minimizes the cost function

$$J = \frac{1}{2}E\{\sum_{k=0}^{T-1}(x_k^T Q x_k + u_k^T R u_k) + x_T^T F x_T\}$$

With the following definitions

- $Q \succeq 0$ is the state performance weight matrix
- $F \succeq 0$ is the terminal cost weight matrix
- $R \succ 0$ is the control effort weight matrix

- w_k is the mean noise at step k
- $E[\cdot]$ is the expectation operator
- $\hat{x}_k$ is the optimal state estimate at step k from the Kalman filter
- u_k is the input for the system at step k $(ie. \delta_k)$

Then defining the value function to be

$$V(k,x) = \min_u J(k,x,u)$$

$$V(k,x) = \min_u \{l(x,u) + E[V(k+1, f(x,u,w)]\}$$

Where the running cost is now defined as

$$l(x,u) = \frac{1}{2}(x_k^T Q x_k + u_k^T R u_k)$$

Now guessing a solution for the value function to be of the form

$$V(k,x_k) = \frac{1}{2}x_k^T X_k x_k + \gamma_k$$

$$V(k+1,x_{k+1}) = \frac{1}{2}x_{k+1}^T X_{k+1} x_{k+1} + \gamma_{k+1}$$

$$V(k+1,x_{k+1}) = \frac{1}{2}(Ax_k + Bu_k + w_k)^T X_{k+1}(Ax_k + Bu_k + w_k) + \gamma_{k+1}$$

Then

$$V(k,x_k) = \min_u \left\{\frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + E\left[\frac{1}{2}(Ax_k + Bu_k + w_k)^T X_{k+1}(Ax_k + Bu_k + w_k) + \gamma_{k+1}\right]\right\}$$

$$\frac{1}{2}x_k^T X_k x_k + \gamma_k = \min_u \left\{\frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + E\left[\frac{1}{2}(x_k^T A^T + u_k^T B^T + w_k^T) X_{k+1}(Ax_k + Bu_k + w_k) + \gamma_{k+1}\right]\right\}$$

Knowing that $E[w_k] = 0$

$$\frac{1}{2}x_k^T X_k\, x_k + \gamma_k = \min_u \left\{\frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + \frac{1}{2}(x_k^T A^T + u_k^T B^T) X_{k+1}(Ax_k + Bu_k) + E\left[\frac{1}{2}w_k^T X_{k+1} w_k + \gamma_{k+1}\right]\right\}$$

Which then first identifies γ_k

$$\gamma_k = E\left[\frac{1}{2}w_k^T X_{k+1} w_k + \gamma_{k+1}\right]$$

$$\frac{1}{2}x_k^T X_k\, x_k + \gamma_k = \min_u \left\{\frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + \frac{1}{2}(x_k^T A^T X_{k+1} A x_k + x_k^T A^T X_{k+1} B u_k + u_k^T B^T X_{k+1} A x_k + u_k^T B^T X_{k+1} B u_k) + \gamma_k\right\}$$

$$\frac{1}{2}x_k^T X_k\, x_k = \frac{1}{2}(x_k^T (Q + A^T X_{k+1} A) x_k) + \min_u \left\{\frac{1}{2}(u_k^T (R + B^T X_{k+1} B) u_k) + \frac{1}{2}(x_k^T A^T X_{k+1} B u_k + u_k^T B^T X_{k+1} A x_k)\right\}$$

$$\frac{1}{2}x_k^T X_k\, x_k = \frac{1}{2}(x_k^T (Q + A^T X_{k+1} A) x_k) + \min_u \left\{\frac{1}{2}(u_k^T (R + B^T X_{k+1} B) u_k) + u_k^T B^T X_{k+1} A x_k\right\}$$

Then letting

$$\tilde{R} = R + B^T X_{k+1} B$$

$$G = \frac{1}{2}\left(u_k^T \tilde{R} u_k\right) + u_k^T B^T X_{k+1} A x_k$$

Seeking the minimum u_k, so

$$\frac{\partial G}{\partial u} = 0 = \frac{\partial}{\partial u}\left(\frac{1}{2}\left(u_k^T \tilde{R} u_k\right) + u_k^T B^T X_{k+1} A x_k\right)$$

Using vector derivative identities

$$\frac{\partial}{\partial y}(y^T M y) = 2My \text{ and } \frac{\partial}{\partial y}(y^T M x) = Mx$$

then

$$\frac{\partial G}{\partial u} = 0 = \tilde{R} u_k + B^T X_{k+1} A x_k$$

Gives optimal control and gain

$$\hat{u}_k = -\tilde{R}^{-1} B^T X_{k+1} A x_k$$

$$K = -\tilde{R}^{-1}B^T X_{k+1}A \; and \; K^T = -A^T X_{k+1}B\tilde{R}^{-1} \tag{1.37}$$

$$\hat{u}_k = Kx_k \tag{1.38}$$

Inserting $\hat{u}_k$

$$\frac{1}{2}x_k^T X_k \, x_k = \frac{1}{2}(x_k^T(Q + A^T X_{k+1}A)x_k) + \frac{1}{2}\left(x_k^T K^T \tilde{R} K x_k\right) + x_k^T K^T B^T X_{k+1} A x_k$$

$$\frac{1}{2}x_k^T X_k \, x_k = \frac{1}{2}(x_k^T(Q + A^T X_{k+1}A + K^T(R + B^T X_{k+1}B)K + 2K^T B^T X_{k+1}A)x_k)$$

Then Identifies the state covariance

$$X_k = Q + A^T X_{k+1}A + K^T(R + B^T X_{k+1}B)K + 2K^T B^T X_{k+1}A$$

Now plugging in (1.37) results in the Riccati Difference Equation (RDE)

$$X_k = Q + A^T X_{k+1}A + A^T X_{k+1}B\tilde{R}^{-1}\tilde{R}\tilde{R}^{-1}B^T X_{k+1}A - 2A^T X_{k+1}B\tilde{R}^{-1}B^T X_{k+1}A$$

Upon further simplification leads to

$$X_k = Q + A^T X_{k+1}A - A^T X_{k+1}B\tilde{R}^{-1}B^T X_{k+1}A$$

$$X_k = Q + A^T X_{k+1}\left(I - B\tilde{R}^{-1}B^T X_{k+1}\right)A$$

$$X_k = Q + A^T X_{k+1}(I - B(R + B^T X_{k+1}B)^{-1}B^T X_{k+1})A \tag{1.39}$$

It should be noted that as $k \to \infty$ eq.(1.39) will approach a steady state and result in the Discrete Algebraic Riccati Equation (DARE). To proceed, X_k can be solved via the value iteration method. The iteration begins by initializing eq.(1.39) with the Final cost F and then iteratively solves the RDE backwards in time to the current sample time k. Depending on the system, this method can converge very quickly, and its implementation is very straightforward which is given in an algorithm found in 2.2.5. Below is an example of the value iteration method.

$$X_T = F$$

$$X_{T-1} = Q + A^T F\left(I - B\tilde{R}^{-1}B^T F\right)A$$

$$X_{T-2} = Q + A^T X_{T-1}\left(I - B\tilde{R}^{-1}B^T X_{T-1}\right)A$$

$$\vdots$$

$$X_{k+1} = Q + A^T X_{k+2}\left(I - B\tilde{R}^{-1}B^T X_{k+2}\right)A$$

$$X_k = Q + A^T X_{k+1}\left(I - B\tilde{R}^{-1}B^T X_{k+1}\right)A$$

Lastly, it should be noted the similarities in the expressions for the gain K and covariance X derived in this section with their counterparts shown in 1.4.3.

1.5.4 **Tuning The Controller**

Tuning a controller for a parameter and time varying dynamic system is not an easy feat but can be done with the help of a simulation of the system which can look at the system's zero-state response, impulse response, and reference tracking performance. With LQG, the controller is tuned by modifying the Q and R matrices. The Q matrix is tuned for the output state performance (i.e. driving the errors to zero) and the R matrix is for modifying the input control effort (i.e. amount of effort applied to steering angle). The following sections show the differences between using a constant Q^c and its speed varying form $Q^c(V_x)$. The track used for the following analysis can be seen in Appendix A in Figure 50.

1.5.4.1 **Constant Weights**

In **Error! Reference source not found.**, **Error! Reference source not found.**, **Error! Reference source not found.** the measured, filtered and steering inputs can be seen at speed ranging from 10-80m/s with the following configuration of Q^c and R^c

$$Q^c = \begin{bmatrix} 0.025 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.001 \end{bmatrix}$$

$$R^c = [0.1]$$

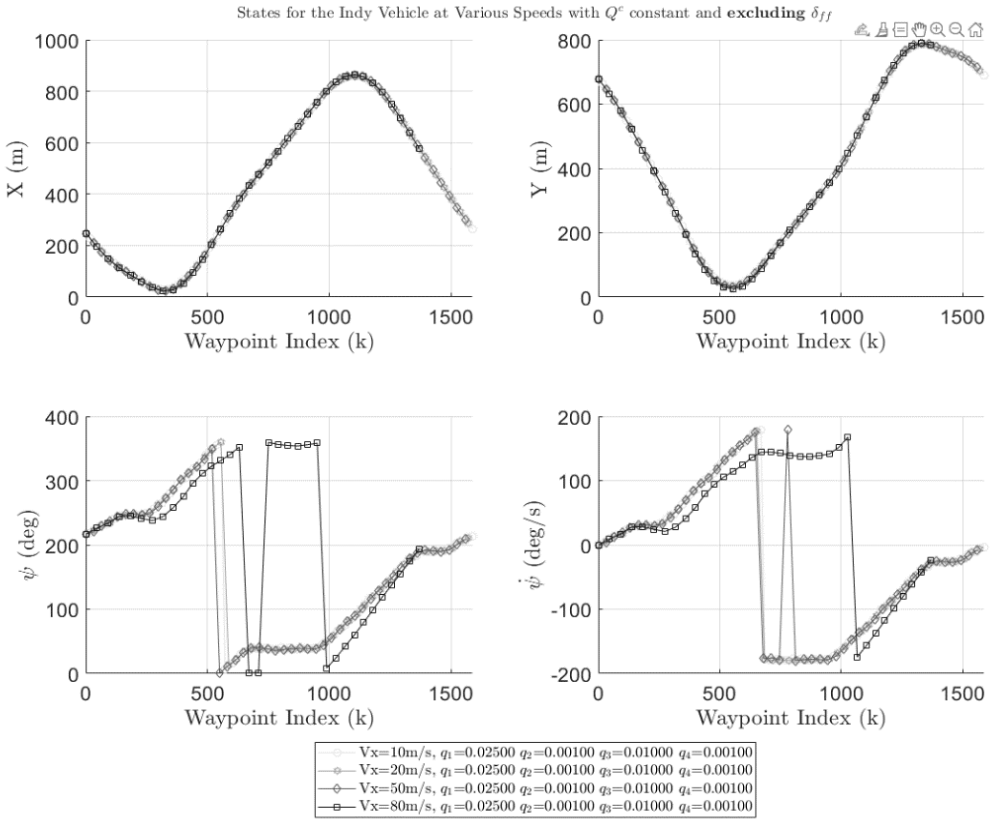

Figure 15: Trial 1 results of vehicle states for Q^c~constant

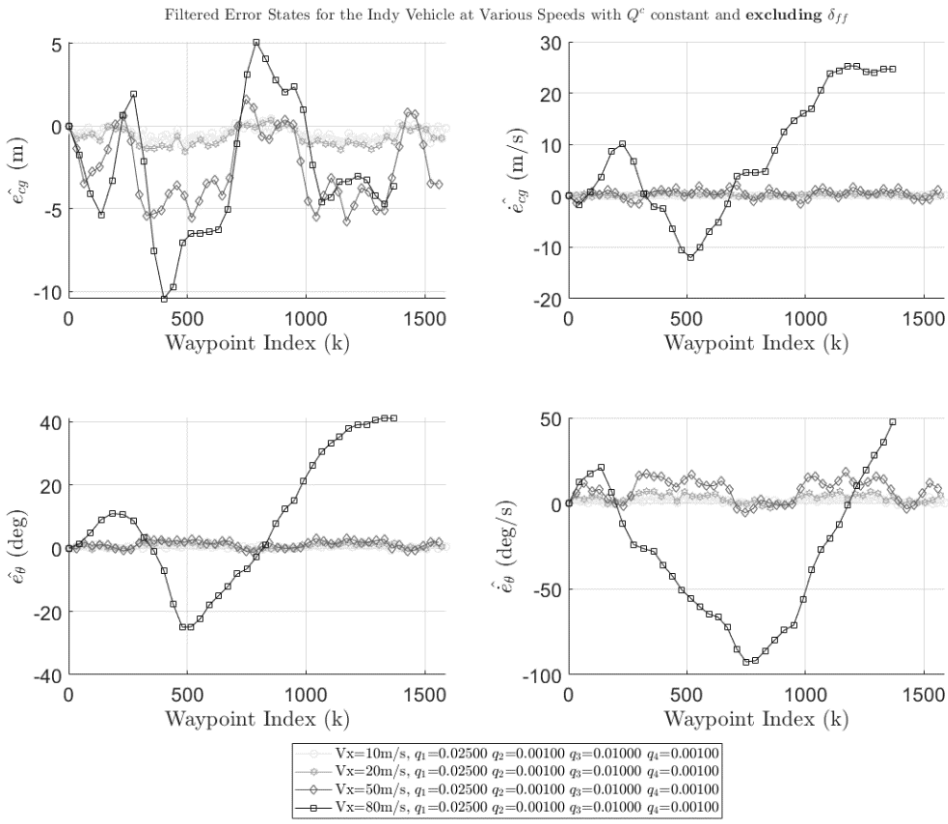

Figure 16: Trial 1 results of vehicle error states for Q^c~constant

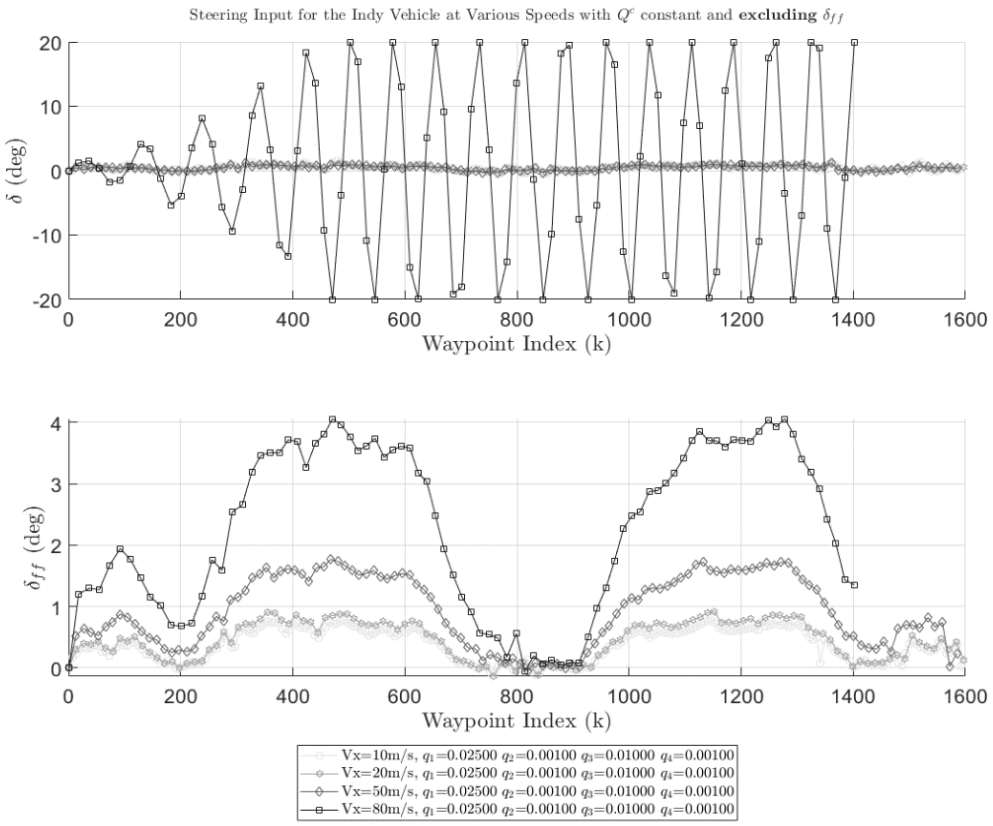

Figure 17: Trial 1 results of the optimal steering inputs for Q^c~constant

It is seen that the vehicle at speeds greater than 50m/s has significant oscillations, unacceptable cross track error and is coincidently asymptotically stable due to the steering constraints preventing higher magnitude steering values to be applied to the vehicle.

The gains from trial 1 were too strong at higher speeds so the values in Q^c are reduced further in trial 2 and again in **Error! Reference source not found.**, **Error! Reference source not found.**, **Error! Reference source not found.** the measured, filtered and steering inputs can be seen at speed ranging from 10-80m/s with the following configuration of Q^c and R^c

$$Q^c = \begin{bmatrix} 0.0175 & 0 & 0 & 0 \\ 0 & 0.0007 & 0 & 0 \\ 0 & 0 & 0.007 & 0 \\ 0 & 0 & 0 & 0.0007 \end{bmatrix}$$

$$R^c = [0.1]$$

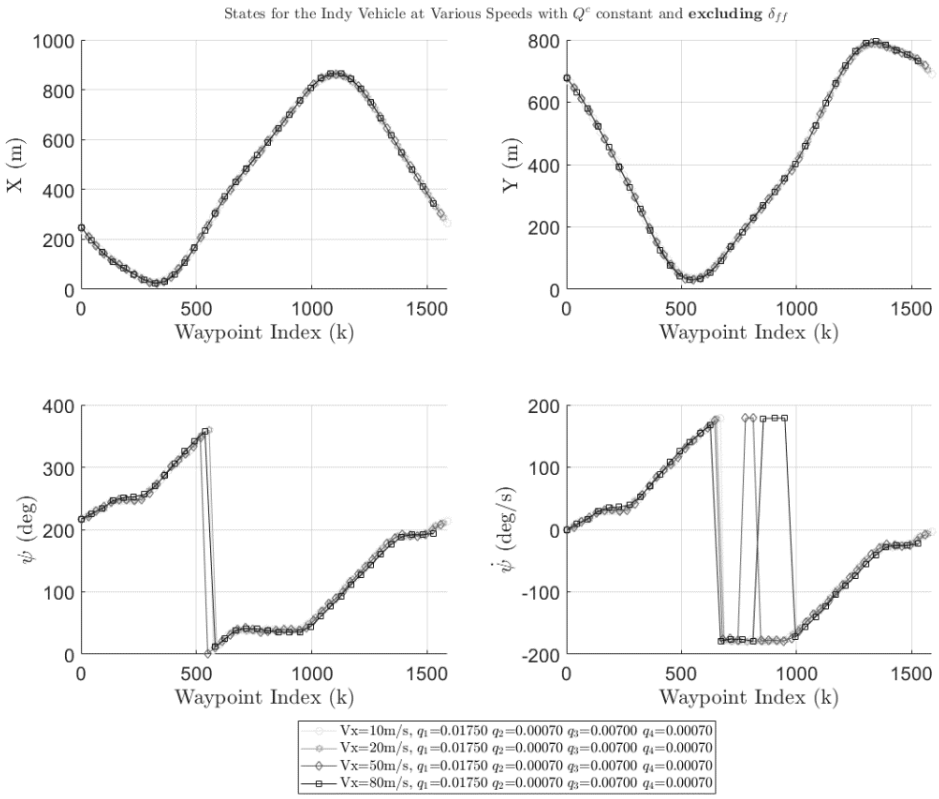

Figure 18: Trial 2 results of vehicle states for Q^c~constant

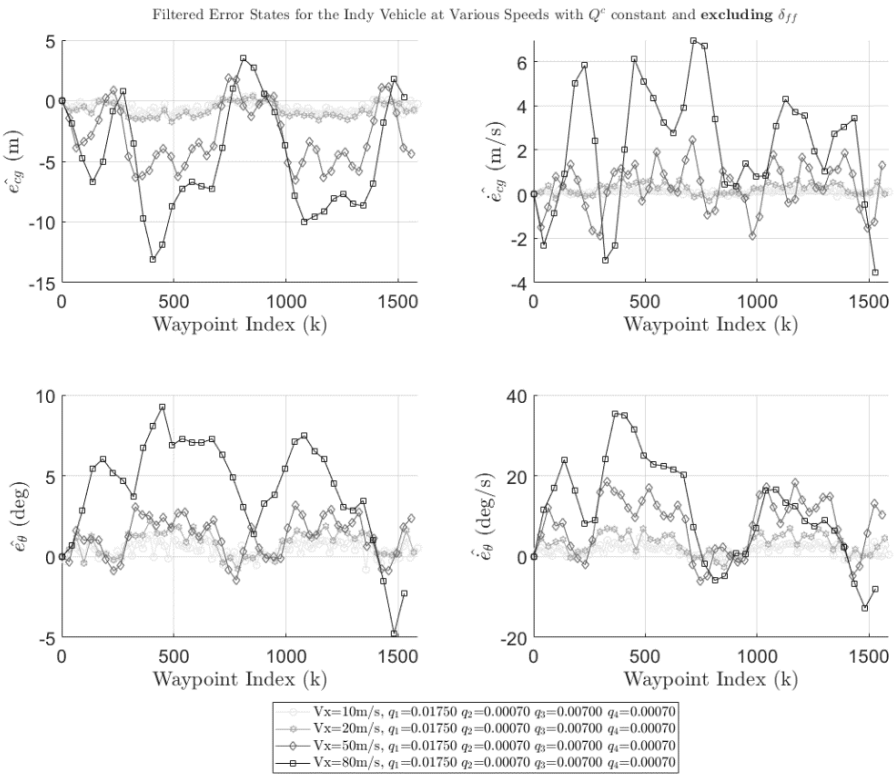

Figure 19: Trial 2 results of vehicle error states for Q^c~constant

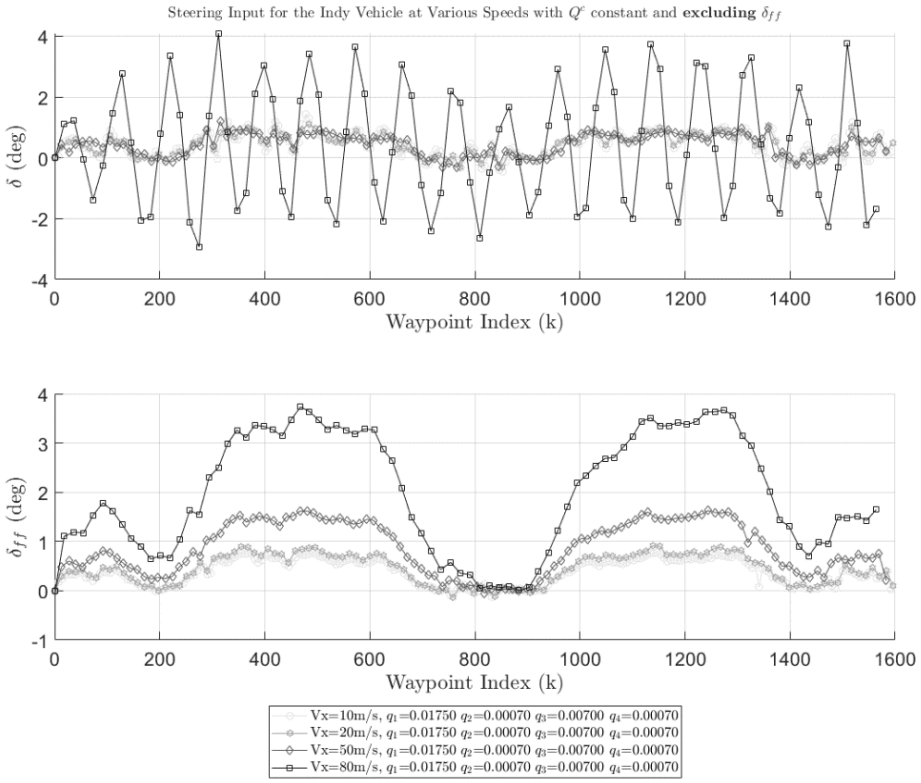

Figure 20: Trial 2 results of the optimal steering inputs for Q^c~constant

It is seen that the same issue persists in trial 2 without having much effect on the response of the vehicle's stability and path tracking capabilities.

The values in Q^c are reduced even further in trial 3 to attempt to reduce oscillations in the system. Again in **Error! Reference source not found.**, **Error! Reference source not found.**, **Error! Reference source not found.** show the measured, filtered and steering inputs can be seen at speed ranging from 10-80m/s with the following configuration of Q^c and R^c

$$Q^c = \begin{bmatrix} 0.0025 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.0001 \end{bmatrix}$$

$$R^c = [0.1]$$

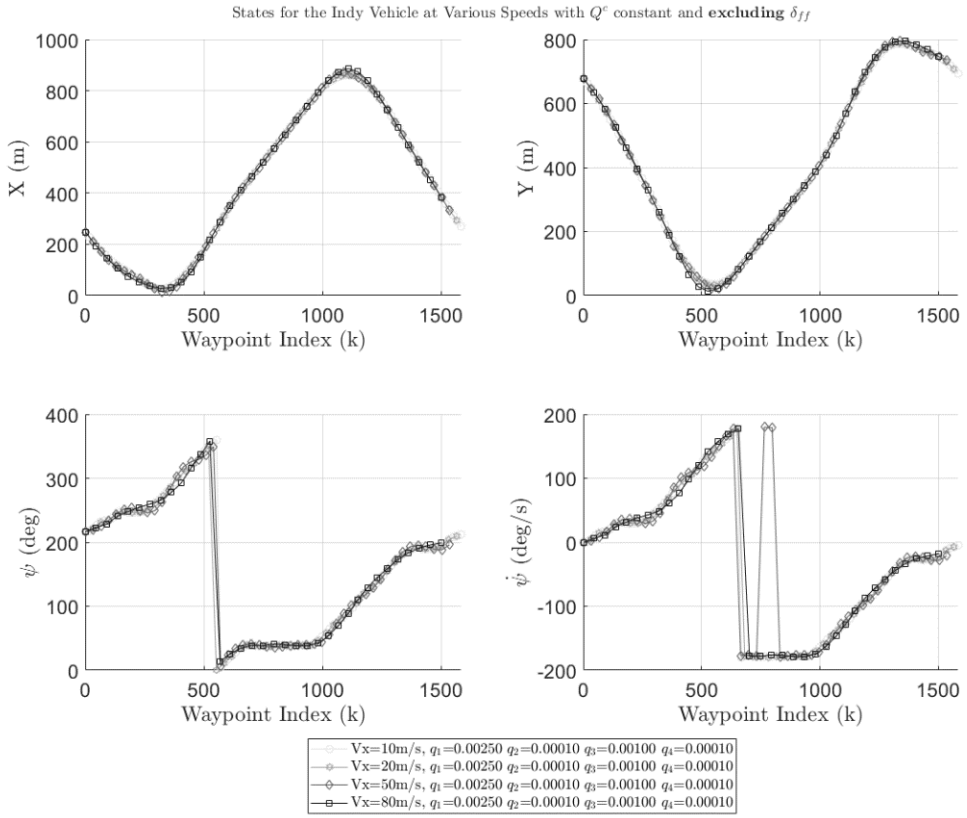

Figure 21: Trial 3 results of vehicle states for Q^c~constant

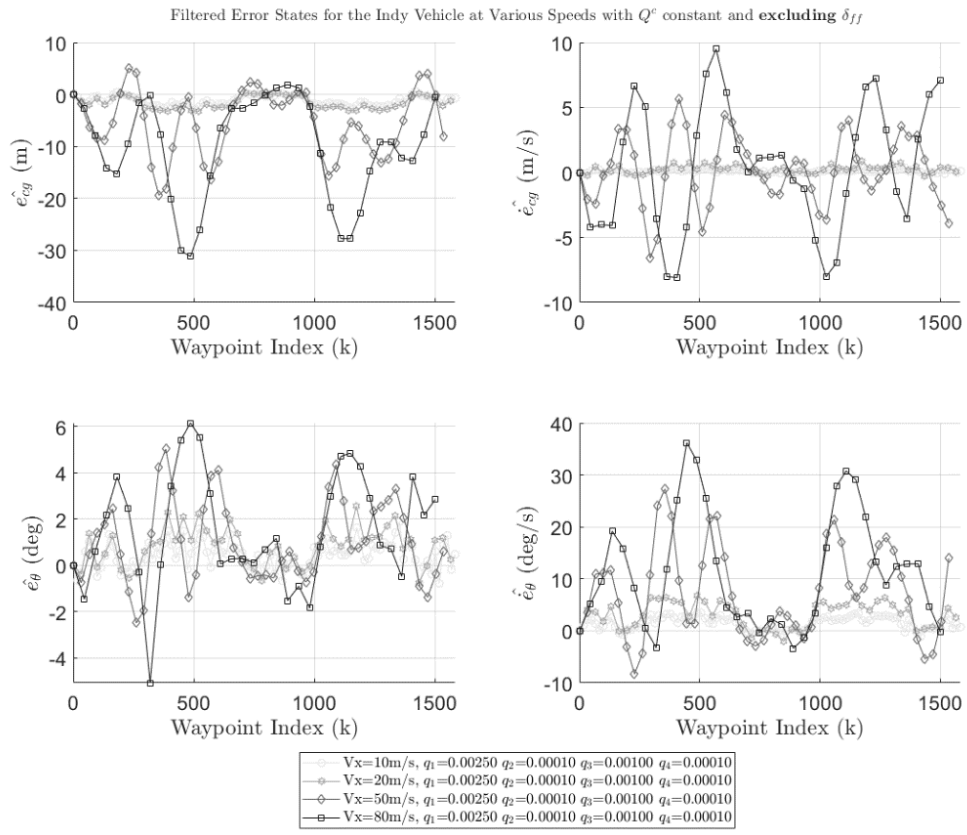

Figure 22: Trial 3 results of vehicle error states for Q^c~constant

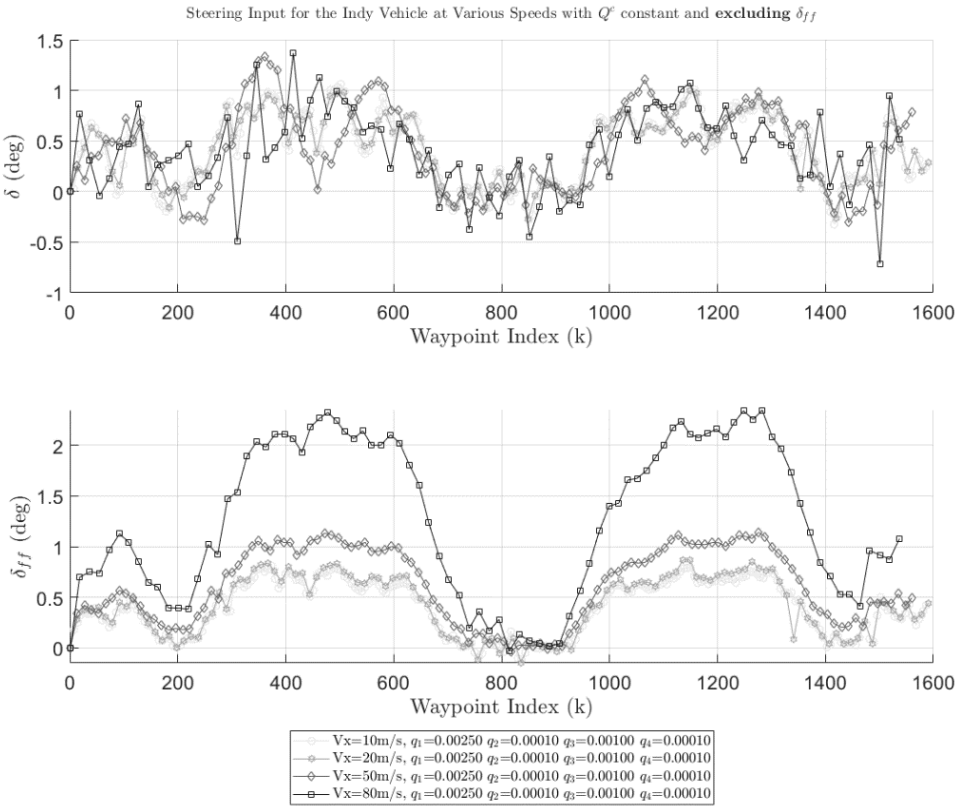

Figure 23: Trial 3 results of the optimal steering inputs for Q^c~constant

It's now observed that the system has reached a much more stable point by reducing the oscillations but at the cost of having a higher cross track error than in the previous trial. The following section seeks to reduce the errors for a range of speeds by allowing Q^c to varying with V_x.

1.5.4.2 **Parameter Varying Weights**

Since the model given in eq.(1.25) is LPV, the weights that go into the cost function were explored further by letting them vary with velocity as the system did. It was seen that as the velocity increased, the Q matrix performed better when its components progressively got smaller while the R matrix remained constant. Linear and non-linear trends were studied and found that

the non-linear fits performed the best in terms of consistent vehicle behavior at a range of speeds. This study could be explored further to determine even better performance.

Defining that each state performance weighting function abide by the power law which can be seen in Figure 24 and Figure 25.

$$q_i^c(V_x) \;=\; a_i V_x^{b_i} \tag{1.40}$$

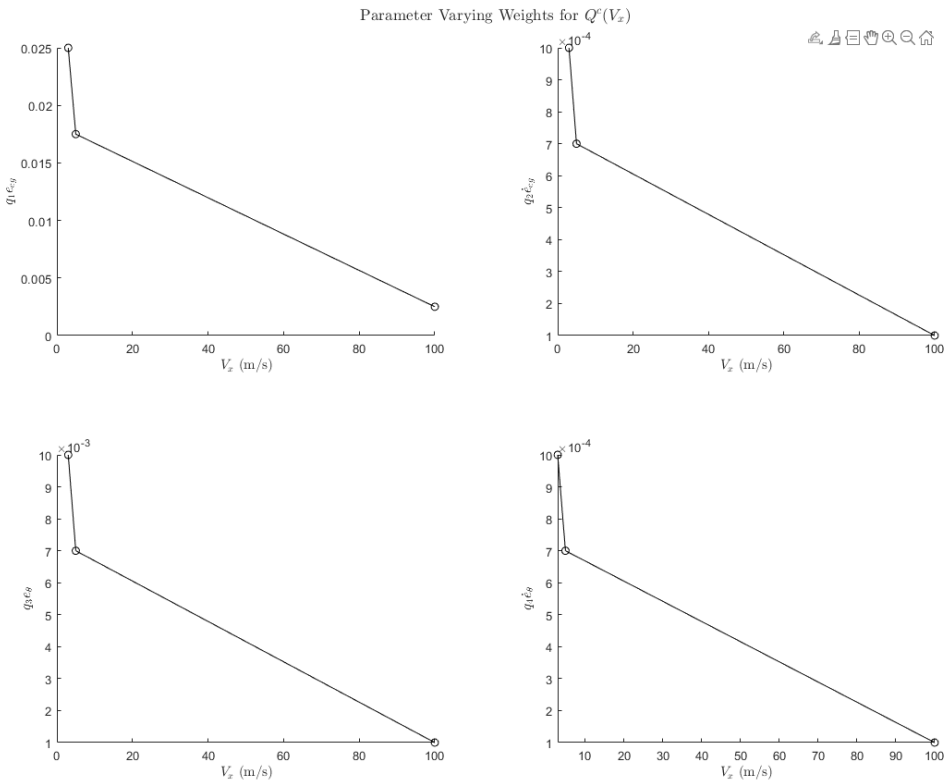

Figure 24: Choice of weights for each state

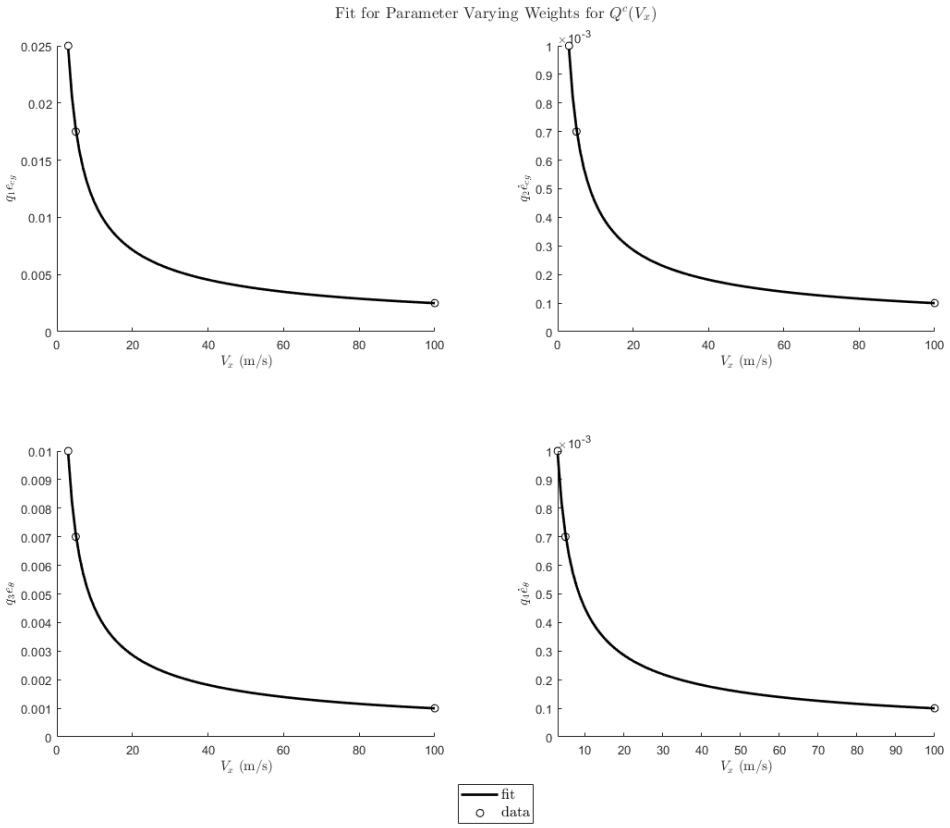

Figure 25: Parameter varying $Q^c(V_x)$ using power law fit

Now the state performance weight matrix will be updated as the longitudinal velocity of the vehicle changes providing a more robust solution at wide range of speeds which can be seen in more detail in Figure 26, Figure 27, and Figure 28.

$$Q^c(V_x) = \begin{bmatrix} q_1^c(V_x) & 0 & 0 & 0 \\ 0 & q_2^c(V_x) & 0 & 0 \\ 0 & 0 & q_3^c(V_x) & 0 \\ 0 & 0 & 0 & q_4^c(V_x) \end{bmatrix}$$

$$R^c = [0.1]$$

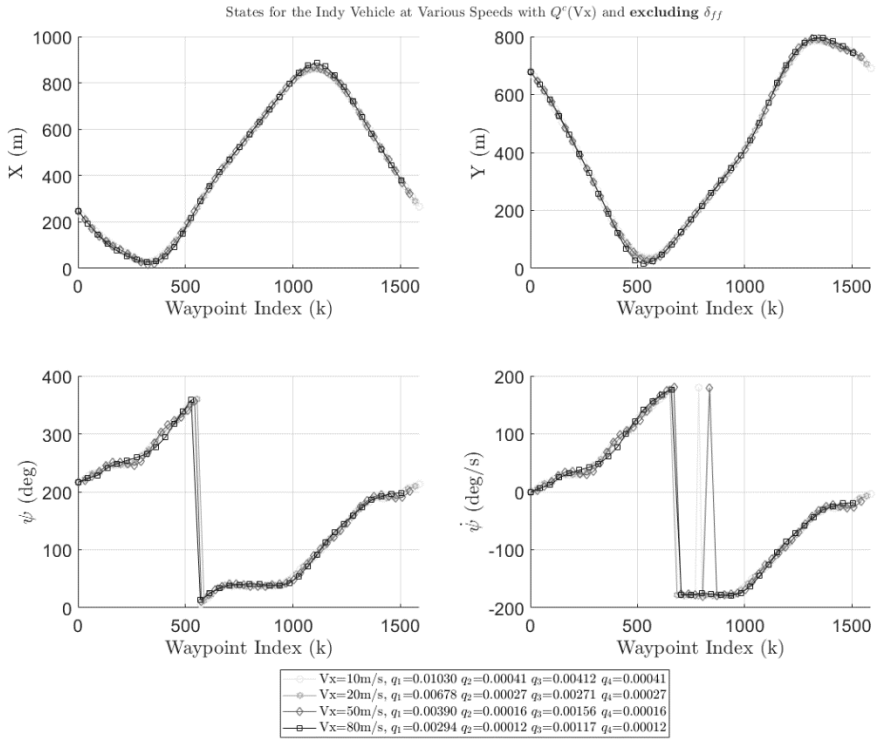

Figure 26: Results of vehicle states for $Q^c(V_x)$

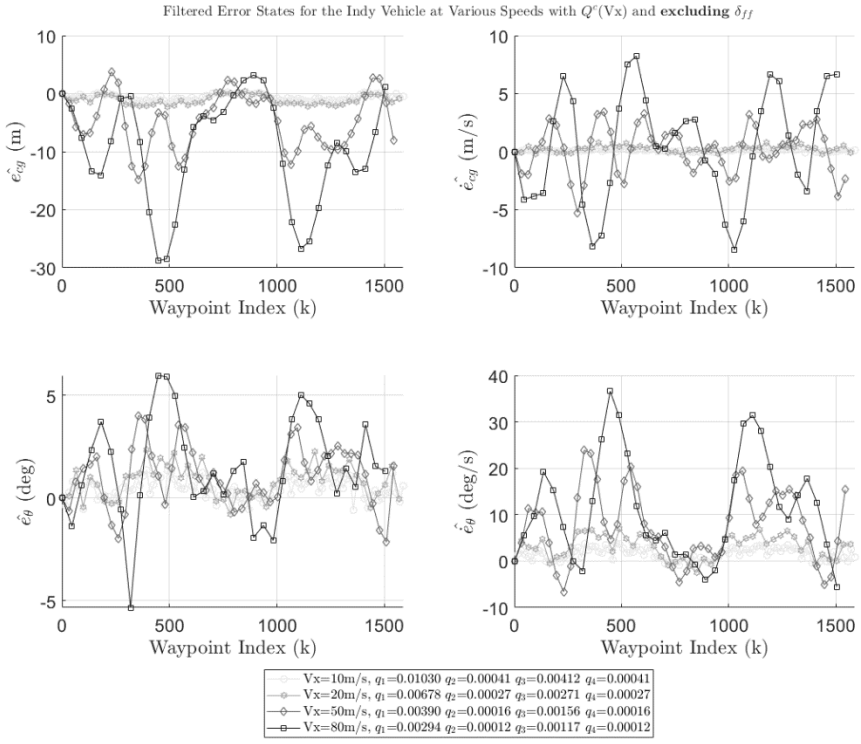

Figure 27: Results of vehicle error states for $Q^c(V_x)$

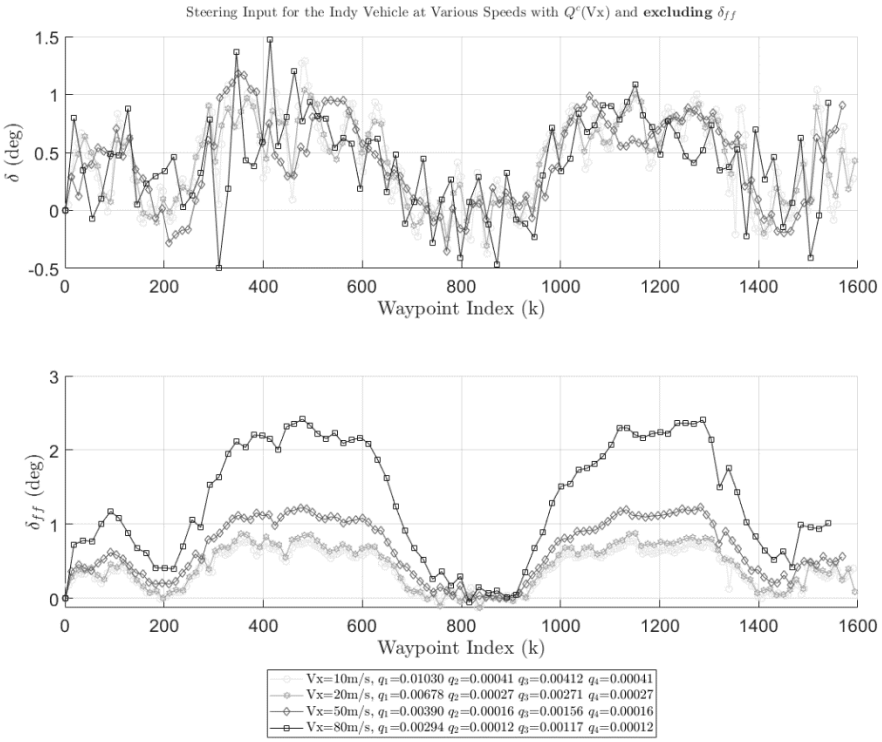

Figure 28: Results of the optimal steering inputs for $Q^c(V_x)$

Now having a more robust controller at a range of operating speeds, the final step in tuning the controller is reducing the steady state errors which is discussed in the next section.

It's worth noting that for some circumstances it may be better to calculate the optimal feedback gain "offline" and use the "gain scheduling" approach for handling a wide range of operating speeds. These gains can be loaded as a lookup table or even create some fitted curves to them to be used online to potentially increase runtime performance of the controller. An example can be seen in Figure 29. The downside is that now the control is limited to a certain state of the system which is technically changing over time and could be accounted for during online calculation of the control gain.

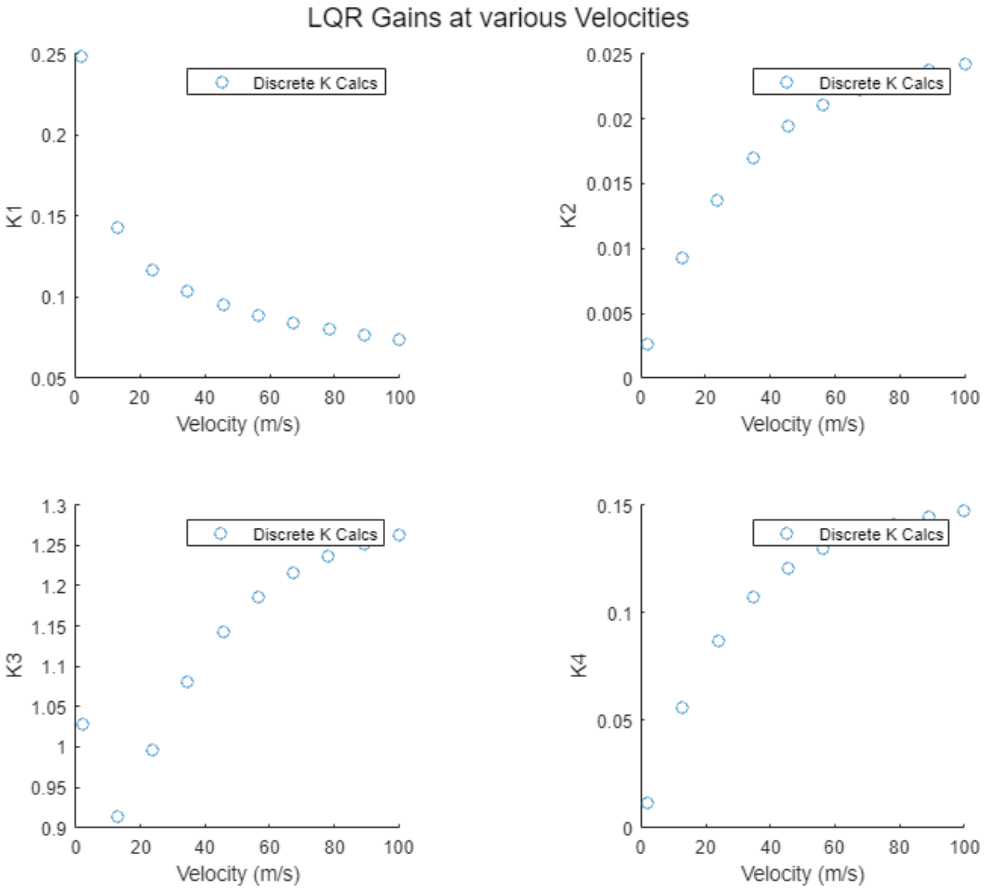

Figure 29: Optimal feedback gains for $Q^c(V_x)$

1.5.5 **Resolving Steady State Error**

Continuing with state space model from eq.(1.25) in its compact form

$$\dot{x} = Ax + B_\delta \delta + B_{\dot{\psi}} \dot{\psi}_{des}$$

Due to the presence of $B_{\dot{\psi}} \dot{\psi}_{des}$, the errors of the system will not all converge to zero when traveling along some path that has a non-zero curvature. The path reference signal into the control block that was previewed earlier in Figure 14 will now be considered to attempt to decrease the tracking errors further by compensating the controller in eq.(1.38) with the feed-

forward term δ_{ff}. The Laplace transform and the final value theorem are used to analyze the steady state of the new proposed closed loop system to reveal the value of δ_{ff}

Letting

$$\delta = Kx + \delta_{ff} \tag{1.40}$$

Then the new closed loop system

$$\dot{x} = Ax + B_\delta\left(Kx + \delta_{ff}\right) + B_{\dot{\psi}}\dot{\psi}_{des}$$

$$\dot{x} = (A + B_\delta K)x + B_\delta\delta_{ff} + B_{\dot{\psi}}\dot{\psi}_{des}$$

Now taking the Laplace transform

$$\mathcal{L}\{\dot{x} = (A + B_\delta K)x + B_\delta\delta_{ff} + B_{\dot{\psi}}\dot{\psi}_{des}\}$$

$$sX(s) = (A + B_\delta K)X(s) + B_\delta\mathcal{L}\{\delta_{ff}\} + B_{\dot{\psi}}\mathcal{L}\{\dot{\psi}_{des}\}$$

$$sX(s) - (A + B_\delta K)X(s) = B_\delta\mathcal{L}\{\delta_{ff}\} + B_{\dot{\psi}}\mathcal{L}\{\dot{\psi}_{des}\}$$

$$\left(sI - (A + B_\delta K)\right)X(s) = B_\delta\mathcal{L}\{\delta_{ff}\} + B_{\dot{\psi}}\mathcal{L}\{\dot{\psi}_{des}\}$$

$$X(s) = \left(sI - (A + B_\delta K)\right)^{-1}\left(B_\delta\mathcal{L}\{\delta_{ff}\} + B_{\dot{\psi}}\mathcal{L}\{\dot{\psi}_{des}\}\right)$$

Now evaluating $\mathcal{L}\{\dot{\psi}_{des}\}$ with the steady state assumption that the vehicle travels with a constant speed V_x along a constant path curvature κ then

$$\mathcal{L}\{\dot{\psi}_{des}\} = \frac{\kappa V_x}{s}$$

And asserting that the sought out δ_{ff} is also constant then

$$\mathcal{L}\{\delta_{ff}\} = \frac{\delta_{ff}}{s}$$

$$X(s) = \left(sI - (A + B_\delta K)\right)^{-1}\left(B_\delta\frac{\delta_{ff}}{s} + B_{\dot{\psi}}\frac{\kappa V_x}{s}\right)$$

$$sX(s) = \left(sI - (A + B_\delta K)\right)^{-1}\left(B_\delta\delta_{ff} + B_{\dot{\psi}}\kappa V_x\right)$$

Now applying the final value theorem which states

$$\lim_{t\to\infty} x(t) = \lim_{s\to 0} sX(s)$$

Then the steady state x_{ss} is

$$x_{ss} = \lim_{t\to\infty} x(t) = \lim_{s\to 0} sX(s) = \lim_{s\to 0}\left(\left(sI - (A + B_\delta K)\right)^{-1}\left(B_\delta \delta_{ff} + B_{\dot{\psi}} \kappa V_x\right)\right)$$

$$x_{ss} = \left(-(A + B_\delta K)\right)^{-1}\left(B_\delta \delta_{ff} + B_{\dot{\psi}} \kappa V_x\right)$$

With

$$K = \begin{bmatrix} k_{e_y} & k_{\dot{e}_y} & k_{e_\psi} & k_{\dot{e}_\psi} \end{bmatrix}$$

Which can be solved symbolically in Matlab to produce

$$x_{ss} = \begin{bmatrix} \frac{\delta_{ff}}{k_{e_y}} \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -\frac{\kappa}{k_{e_y}}\left(\frac{mV_x^2}{(L_f+L_r)}\left(\frac{L_r}{C_f} - \frac{L_f}{C_r}\left(1 - k_{e_\psi}\right)\right) - \left(L_f + L_r - L_r k_{e_\psi}\right)\right) \\ 0 \\ \frac{\kappa}{C_r(L_f+L_r)}\left(-C_r L_f L_r - C_r {L_r}^2 + L_f m V_x^2\right) \\ 0 \end{bmatrix} \quad (1.41)$$

Then

$$\delta_{ff} = \kappa\left(\frac{mV_x^2}{\left(L_f + L_r\right)}\left(\frac{L_r}{C_f} - \frac{L_f}{C_r}\left(1 - k_{e_\psi}\right)\right) + \left(L_f + L_r - L_r k_{e_\psi}\right)\right)$$

Under further inspection

$$\delta_{ff} = \kappa\left(V_x^2\left(\frac{m}{L}\left(\frac{L_r}{C_f} - \frac{L_f}{C_r}\right)\right) + \left(\frac{mV_x^2 L_f}{LC_r} - L_r\right)k_{e_\psi} + L\right)$$

Identifying that

$$K_v = \frac{m}{L}\left(\frac{L_r}{C_f} - \frac{L_f}{C_r}\right)$$

Is known as the understeer gradient, then

$$\delta_{ff} = \kappa V_x^2 K_v + \kappa \left(\frac{m V_x^2 L_f}{L C_r} - L_r \right) k_{e_\psi} + \kappa L$$

Now that δ_{ff} has been found, it can be seen in eq.(1.41) it's not possible for e_ψ to reach zero steady state error but the controller is able to achieve a zero steady state error for e_y. Results for including δ_{ff} with the same $Q^c(V_x)$ are shown in Figure 30, Figure 31, and Figure 32, which dramatically improve path tracking performance even at the higher speeds. With this new capability, future work could include improving the design of the weight functions in the state performance matrix $Q^c(V_x)$ given by eq.(1.40) to further reduce the lateral errors. It can be seen in Figure 61 that the current form of the controller completely breaks down at 100m/s without the aid of δ_{ff} but in Figure 66 shows how it brings the system back to being stable but still has an unacceptable lateral error. Plots of the various paths used for testing can be seen in Appendix A and detailed runs at all the various speeds can also be seen in Appendix B.

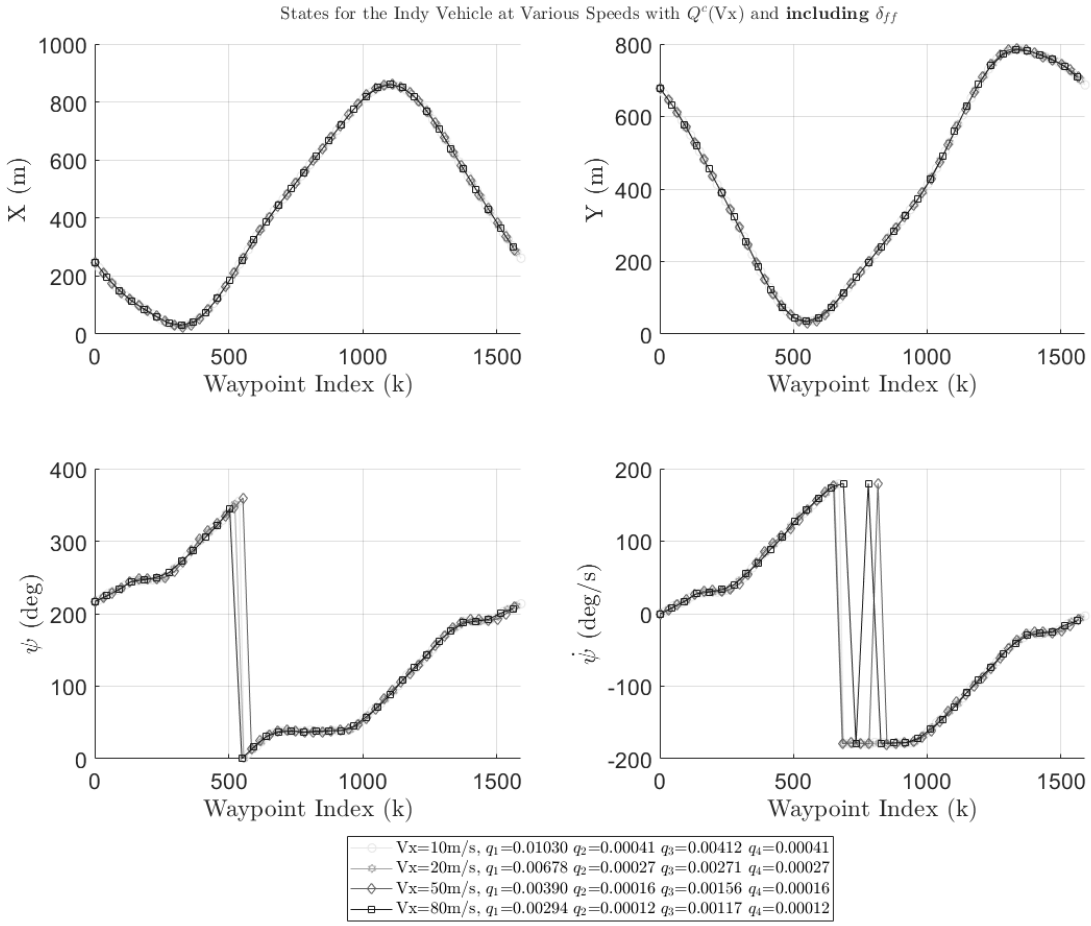

Figure 30: Vehicle states with Q(Vx) and feed-forward

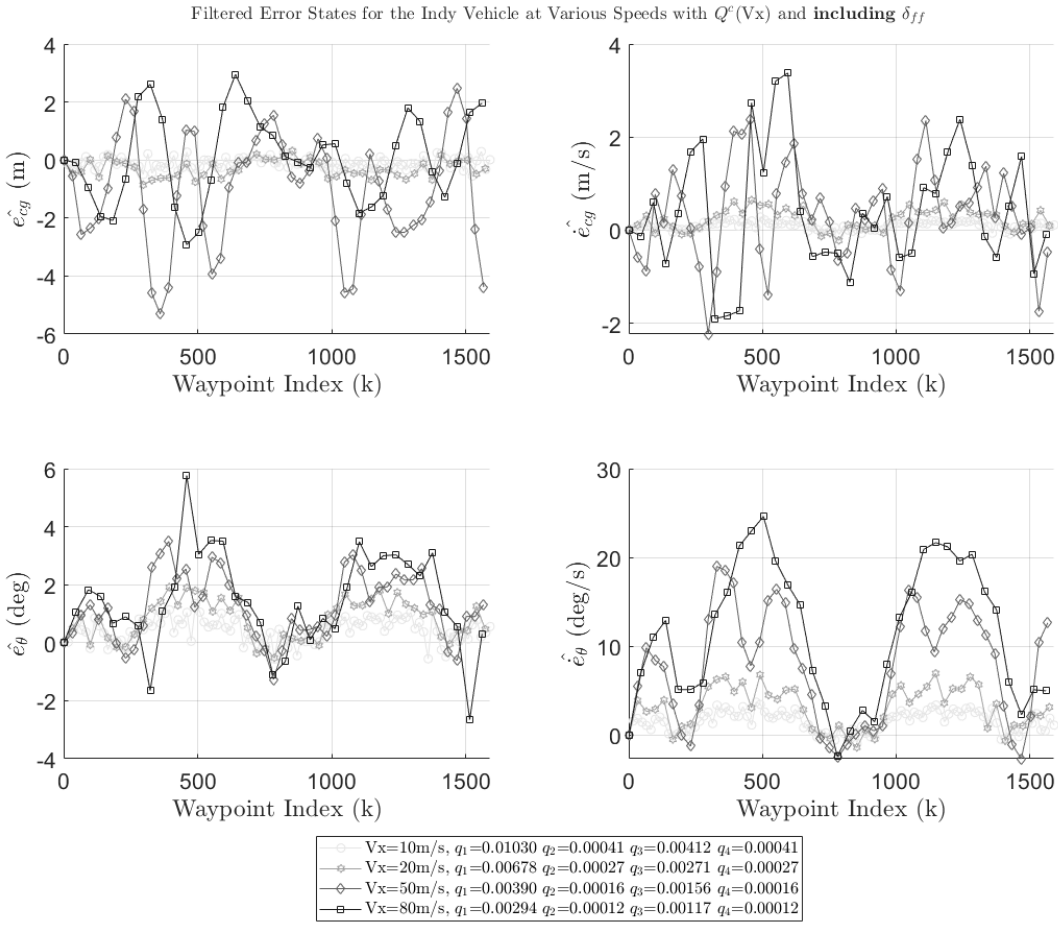

Figure 31: Vehicle error states with Q(Vx) and feed-forward

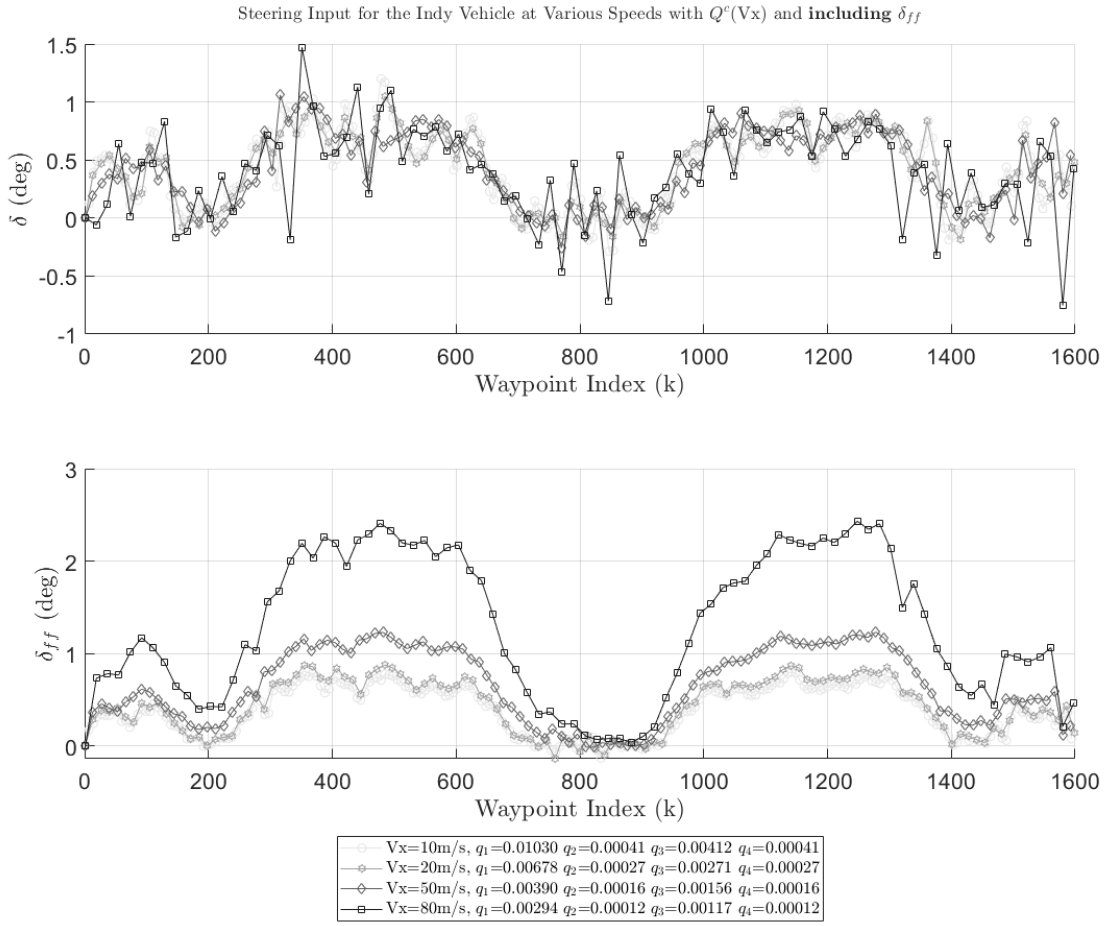

Figure 32: Calculated steering values with Q(Vx) and feed forward

Chapter 2 Implementation

2.1 **Introduction**

For implementing everything in chapter 1, the following sections include the (pseudo) code for the algorithms used provide a general understanding of how one would go about coding them in various languages that they feel comfortable using. The source code on GitLab is all in python3 and Matlab. Once these algorithms are understood comes to how use them which is shown in the autonomous vehicle framework that was created for this thesis using a popular robotics software called ROS2. General knowledge of what ROS2 is and how to use it within the framework is also provided so that the reader may dive into the world of ROS and to show how powerful this software is for robotic applications when it comes to industry ready products or experimenting new ideas. ROS2 was used to implement all the experimental procedures for collecting data and testing the controller in both the simulator and on the physical car. Lastly, the results of the various topics discussed throughout this thesis are presented for validation. It is suggested that the reader has basic Linux skills which will complement all the setup procedures and use developer tools such as GitLab and Docker. Brief introductions to both GitLab and docker are given which as well provide the crucial components of collaboration, version control, and product distribution for the autonomous framework (UCSD Robocar) that was created. It is to be noted that the concepts and ideas discussed here can be applied to many other types of frameworks or projects because of the generality that was maintained throughout the construction of UCSD Robocar.

2.2 **Core Algorithms**

The following sections provide the code for the main algorithms used for path tracking. These algorithms are written in stand-alone format for robust reusability. It is better practice to have code written in object-oriented-programming (OOP) format because it makes functions and variables more accessible throughout the rest of the framework and makes updating current or implementing new algorithms a much smoother process. The following section breaks down the step-by-step process of the core control algorithms that are used in the autonomous vehicle control package software.

2.2.1 **Discrete Parameter-Varying Dynamic Car Model**

This program allows for updating the bicycle car model and then discretized as it is a parameter-varying system. It will need to be updated at every time step at runtime to allow for varying velocity operation.

ALGORITHM 1: DISCRETIZING AND UPDATING DYNAMIC MODEL

INPUT: Longitudinal velocity (V_x)

OUTPUT: Discrete state space matrices $(A_d,\ B_d)$

1 **if** $\boldsymbol{V_x \leq V_{min}}$

2 $V_x = V_{min}$

3 $A, B \leftarrow$ update continuous-time dynamics as function of V_x

5 $A_d, B_d \leftarrow zeros\big(size(A, B)\big)$ initialize each discrete matrix to be same size as continuous matrix

6 **for** $\boldsymbol{k}$ **= 0:max_iteration**

7 $A_d = A_d + \dfrac{(AT_s)^k}{k!}$

8 $B_d = B_d + \dfrac{A^k {T_s}^{k+1}}{(k+1)!} B$

9 **end**

2.2.2 **Parameter-Varying Weight Functions**

This algorithm is for updating the state performance weight matrix that will vary with V_x which allows for achieving design specifications at all possible V_x the vehicle could be at. The control effort weight matrix was set to a constant and read from a configuration file. The coefficients a_i, b_i for each state can be computed offline or a single time upon initialization of the controller.

ALGORITHM 6: CALCULATE WEIGHTS

INPUT: Longitudinal velocity and the size of A $(V_x, size(A))$

OUTPUT: State performance and control effort weights at time sample k $(Q_k,\ R_k)$

$\boldsymbol{Q}$ **= zeros(size(**$\boldsymbol{A}$**))**initialize matrix of zeros with same size as A

1 **for i = 0:length(**$\boldsymbol{Q_k}$**)** run loop for each state

2 a_i, b_i ← read from configuration file for state i

3 $\boldsymbol{Q}(i,i) = a_i V_x^{b_i}$ assign state i performance in its diagonal position

4 ***end***

5 $\boldsymbol{R_k}$ ← **read from configuration file (**does not vary with longitudinal velocity**)**

6 $\boldsymbol{Q_k} \leftarrow \boldsymbol{Q}$

2.2.3 **Cross-Track Error**

This algorithm will calculate the current cross-track error with respect to the center of gravity of the vehicle. It expects that the two closest points in the path to the vehicle are provided but can be easily found by finding the index of the minimum Euclidean distance between the vehicle and the path and then taking the next index in the array as the second closest point. This

assumes that the path provided is orientated such that each successive point is the next point for the vehicle to move towards. It will need to be updated at every time step at run time.

ALGORITHM 7: CALCULATE CTE

INPUT: Closest two points in desired path and vehicle position with respect to its center of gravity $\left((X_{p1}, Y_{p1}), (X_{p2}, Y_{p2}), (X_{cg}, Y_{cg})\right)$

OUTPUT: Lateral error to desired path (CTE)

1. $\vec{P}_{12} = (X_{p2} - X_{p1},\ Y_{p2} - Y_{p1})$ Vector connecting 2 closest points of path to follow
2. $\overrightarrow{\hat{P}_{12}^{\perp}} = \frac{(Y_{p2} - Y_{p1}, -(X_{p2} - X_{p1}))}{\|\vec{P}_{12}\|}$ Unit vector perpendicular to path
3. $\vec{V}_{CG1} = (X_{CG} - X_{p1},\ Y_{CG} - Y_{p1})$ Vector from point 1 to the vehicle position
4. $\overrightarrow{CTE} = (X_{CG} - X_{p12},\ Y_{CG} - Y_{p12})$ cross-track error vector
5. $\boldsymbol{CTE = \vec{V}_{CG1} \cdot \overrightarrow{\hat{P}_{12}^{\perp}}}$ **cross-track error (sign of this product will dictate to left or right of path)**

2.2.4 **Covariance Matrix**

This algorithm solves the RDE/DARE depending on the max iteration specified. If set to 1, the calculation should only be performed once and will be the solution to the RDE. If solved recursively until the previous covariance is equal to the current covariance, then it has solved the DARE.

ALGORITHM 2: SOLVE RDE/DARE

INPUT: discrete system dynamics and weights at sample *k* $(A_k,\ B_k,\ Q_k,\ R_k,\ optional:\ X_{k-1})$

OUTPUT: current and predicted state covariance matrix $(X_k,\ X_{k+1})$

1. **if optional is not none**
2. $\quad X \leftarrow$ initialize as X_{k-1}
3. **else**
4. $\quad X \leftarrow$ initialize as Q_k
6. **for i = 0:max_iteration**

7 | $X = ((I - B(B_k^T X B_k + R_k)^{-1} B_k^T)X$

8 **end**

9 $\boldsymbol{X_k = X}$

10 $\boldsymbol{X_{k+1} = A_k^T X_k A_k + Q_k}$

2.2.5 **Gain**

This algorithm calculates both Kalman and feedback gains that coincide with the solution to the RDE. This algorithm will need to be called at every time step at run time when updating the gains with respect to the state equation matrices.

ALGORITHM 3: CALCULATE GAIN MATRIX

INPUT: discrete system dynamics and weights at sample k
$(A_k, B_k, Q_k, R_k, optional: X_k)$

OUTPUT: gain and variance matrices $(L,\ K,\ X)$

1 **if optional is not none**

2 | $\boldsymbol{X} \leftarrow X_k$

3 **else**

4 | $X_k, X_{k+1} \leftarrow$ call algorithm 2 $(A_k,\ B_k,\ Q_k,\ R_k)$

5 | $\boldsymbol{X} \leftarrow X_k$

6 $\boldsymbol{L = \left(B_k^T X B_k + R_k\right)^{-1} B_k^T X}$ known as the "innovation gain" or "Kalman gain"

7 $\boldsymbol{K = L A_k}$ known as the "Feedback gain"

2.2.6 **Linear Kalman Filter**

This algorithm calculates the current filtered state-estimate of the vehicle and the states covariance matrix for a single time step and will need to be called at every time step at run time to keep updating the estimates based on current measurement data.

ALGORITHM 4: CALCULATE OPTIMAL STATE-ESTIMATE FROM LINEAR SYSTEM

	INPUT:	Current and previous discrete system dynamics, weights, previous state covariance state estimate and input, and current measurement $(sys_k, sys_{k-1}, Q_k, R_k, X_{k\|k-1}, \hat{x}_{k-1\|k-1}, u_{k-1}, y_k)$
	OUTPUT:	optimal state-estimate and variance $(\hat{x},\ X)$
1	L_k, K_k	← call algorithm 5 to get gains $(A_k^T,\ C_k^T,\ Q_k,\ R_k,\ X_{k\|k-1})$
2	L_k, K_k	← $L_k^T,\ K_k^T$ by duality, take transpose to get estimation and innovation gains
3	X_k, X_{k+1}	← call algorithm 4 to get filtered covariances $(A_k^T,\ C_k^T,\ Q_k,\ R_k,\ X_{k-1})$
4	$\hat{x} \leftarrow \hat{x}_{k\|k} = (I_n - L_k C_k)(A_{k-1}\hat{x}_{k-1\|k-1} + B_{k-1}u_{k-1}) + L_k y_k$	
5	$X \leftarrow X_{k+1}$	

2.2.7 **Extended Kalman Filter**

This algorithm behaves the same as the linear Kalman filter but uses the non-linear systems state and output equations when performing updates.

ALGORITHM 5: CALCULATE OPTIMAL STATE-ESTIMATE FROM NON-LINEAR SYSTEM

	INPUT:	discrete system dynamics, previous state covariance, sensor weights $(A_k, B_k, C_k, D_k, X_{k-1}, Q_k, R_k, f_k(x_k, u_k), h_k(x_k), y_k)$
	OUTPUT:	optimal state-estimate $(\hat{x},\ X)$
1	L, K	← call algorithm 5 to get gains $(A_k^T,\ C_k^T,\ Q_k,\ R_k,\ X_{k-1})$
2	L, K	← $L^T,\ K^T$ by duality, take transpose to get estimation and innovation gains
3	X_k, X_{k+1}	← call algorithm 4 to get filtered covariances $(A_k^T,\ C_k^T,\ (F_k Q_k F_k^T),\ (G_k R_k G_k^T),\ X_{k-1})$
4	$\hat{x}_{k\|k} = f_{k-1}(\hat{x}_{k-1\|k-1},\ u_{k-1}) + L(y_k - h_k f_{k-1}(\hat{x}_{k-1\|k-1},\ u_{k-1}))$	
5	$X \leftarrow X_{k+1}$	

2.2.8 **LQG**

This algorithm combines all the algorithms mentioned previously which will perform a single LQG calculation and will need to be called at every time step at run time.

ALGORITHM 8: CALCULATE OPTIMAL STATE-ESTIMATE FEEDBACK CONTROLLER

INPUT: continuous system dynamics, previous discrete system dynamics, weights, previous state covariance state estimate and input, and current measurement $(A, B, C, D, sys_{k-1}, V_x, Q_k^o, R_k^o, X_{k|k-1}^o, \hat{x}_{k-1|k-1}, \hat{u}_{k-1}, y_k)$

OUTPUT: optimal state-estimate feedback control, state, and variance $(\hat{u}_k, \hat{u}_{k-1}, \hat{x}_{k-1|k-1}, X_{k|k-1}^o)$

1 A_k, B_k, C_k, D_k ← call algorithm 1 $(A,\ B,\ C,\ D,\ V_x)$ update CT model speed & convert to DT

2 Q_k^c, R_k^c ← call algorithm 2 (V_x) these are the state performance weights for minimizing control cost

3 K_k^c ← call algorithm 5 $(A_k,\ B_k,\ Q_k^c,\ R_k^c)$ *this is the feedback control gain*

4 $\hat{x}_{k|k}, X_{k|k+1}^o$ ← call algorithm 6 $(sys_k,\ sys_{k-1},\ X_{k|k-1}^o,\ Q_k^o,\ R_k^o,\ \hat{x}_{k-1|k-1}, \hat{u}_{k-1},\ y_k)$

5 $\hat{u}_k = -K_k^c \hat{x}_{k|k}$ **feedback controller to apply to plant**

6 $\hat{u}_{k-1}, \hat{x}_{k-1|k-1}, X_{k|}^o$ ← **update values for next time step** $(\hat{u}_k,\ \hat{x}_{k|k},\ X_{k+1|k}^o)$

2.3 **Human-Machine-Interface**

2.3.1 **Introduction**

An autonomous vehicle framework (*UCSD Robocar*) was created in support of this thesis. UCSD Robocar is built on top of a popular software used for robotic applications called Robot Operating System (ROS and ROS2) was used for controlling the various scaled robot cars. The framework provides flexibility from implementing traditional programming or machine learning techniques to achieve an objective. The framework works with a vast selection of popularly used sensors, controllers and actuators making it a robust framework to use across various platforms. The framework has been tested in ROS simulators as well as on 1/16, 1/10,

1/5 scaled robot cars. For the Go-Kart and Indy vehicles, an industry provided framework was used.

2.3.2 **Development Platforms**

There are 3 main embedded computers that were used to deploy the UCSD Robocar framework on the physical robots and each of them are ARM based computer architectures and belong to the NVIDIA Jetson family. Jetson Nano Jetson Xavier NX, and the Jetson AGX Xavier.

Other computer architectures like X86 from intel and M1 from apple were also compatible and were the main computers used when running simulations of robot behavior under different conditions and controller performance criteria. The host OS on all the Jetson computers use Ubuntu18 which is flashed through NVIDIA's Jetpack image. However, the docker image uses Ubuntu20 which is an OS recommendation (essentially a requirement for not having to deal with package installation issues) for using ROS2 and is discussed in more detail below. Ubuntu in general is a great OS for robotic applications due to the large robotics community that has used it to develop and share their works.

To be able to use different types of computer architectures and ensuring repeatability, a docker image was created that runs Ubuntu20.04 and contains the UCSD Robocar framework and all its software related dependencies which was extremely convenient and efficient when making the transition from the simulator to the actual robot. To get the docker image working, pull the UCSD Robocar docker image from docker hub onto the development computer. This allows for plug-n-play capabilities if all the sensors and hardware are connected to the computer properly.

2.3.3 **ROS Introduction**

ROS is used in this autonomous vehicle application for its built-in ecosystem for receiving and sending sensor and control messages across many different programs in the framework. The framework allows for both ROS-Noetic and ROS2-Foxy to work together through another software tool called ROS bridge or can be used independently per requirement of the application. ROS also has a large community of roboticists who share various packages such as SLAM, sensor drivers, obstacle avoidance etc. that are all open sourced to make use, understanding, and modifying them all in the hands of the developer.

ROS has a unique architecture which consists of various tools and communication methods between numerous programs. In this introduction, only the concepts of nodes, topics, launch files and parameters will be discussed. However, there are other core concepts such as actions and services that can provide more robust functionality of what a robot/system is capable of. This is meant to be a brief introduction as to what ROS is and to demystify its usage by exposing the main ideas behind it.

2.3.3.1 **Nodes**

Nodes can be the source code or be the place where source code is imported to be used. Nodes initiate the potential communication between various other nodes that might be running in parallel. For any program that needs information or data from another program, a node must be initialized! Without the creation of the node, the communication is severed from the rest of the programs. Nodes can communicate (send data) with each other over a shared topic. If this topic is not set up properly, there will not be any communication between the nodes.

2.3.3.2 **Topics**

Topics provide the means for 2 or more nodes to share data with one another. For at least 2 nodes to start talking to one another, at least one topic must be made, one node must publish to a topic to send data, and another node must subscribe to that same topic to receive that data. A simple example is shown in Figure 33. These topics can pass various types of data structures from simple integers to custom defined data types which is very convenient and powerful when attempting to create a framework of code to properly share data within itself.

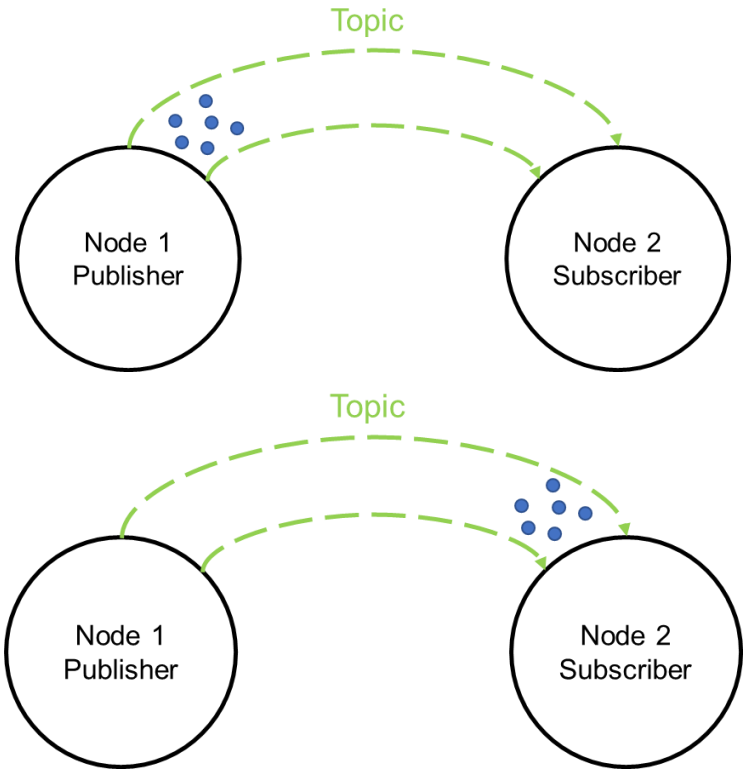

Figure 33: ROS Topics with Nodes

2.3.3.3 **Launch Files**

These are one-stop-shop programs that can run numerous nodes simultaneously which can be very powerful and convenient. For example, some robotic systems could have many

different nodes for various scenarios or environments that the robot could be in. They can also load parameters into nodes to make programs much more robust and minimize user error. With ROS2, Logic can also be implemented in these files because they're in python. An example is shown in Figure 34.

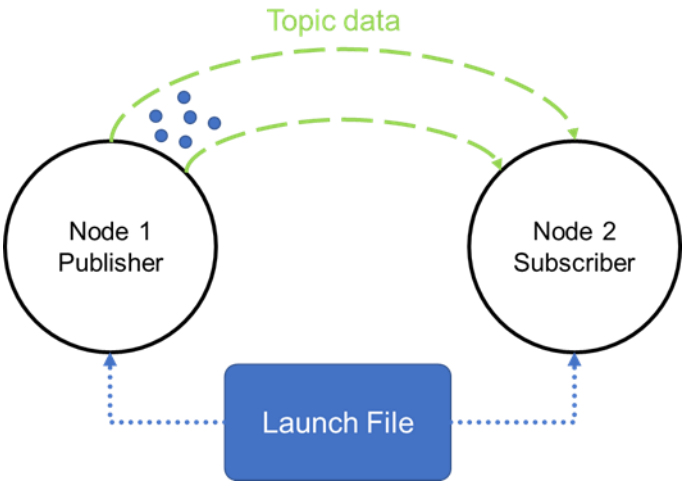

Figure 34: ROS Launch files with nodes

2.3.3.4 **Parameters**

Parameters in ROS are incredibly useful even with simple robotic applications. Parameters can define physical robot limitations, behavior, information about the environment the robot is in, and so much more. Parameters can be set inside of nodes and is possible to update at run time in the terminal! They can always be viewed via the terminal as well for verification. A simple example of using parameters is given in Figure 35. This example shows using the same node (source code) and only updating a set of user defined parameters to achieve different behavior or updating system properties when in different environments. This was exhausted on the 3 different platforms used for testing the LQG controller and when changing track conditions.

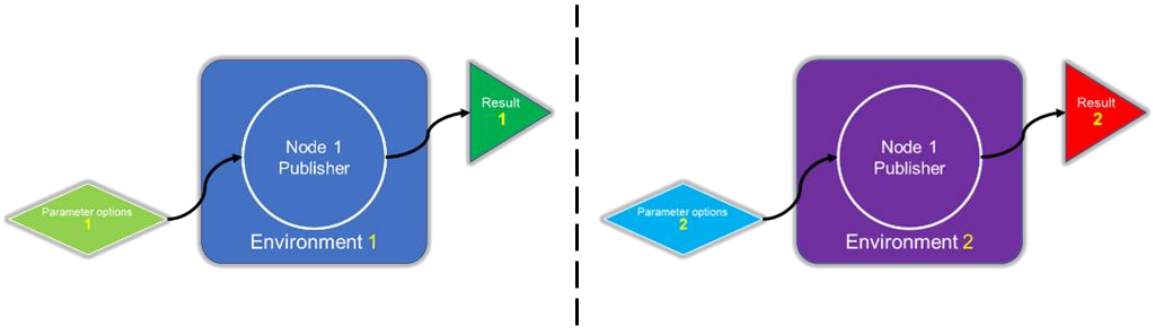

Figure 35: ROS Parameters in different environments

2.3.4 **UCSD Robocar Framework Breakdown**

Now having a basic understanding of ROS, The Autonomous framework that was created for this thesis is broken down into the packages and nodes that must have been made to implement all the theory from part 1. Figure 36 gives a high-level overview of how the framework was structured. This piecewise structure allows for plug-in type capabilities which makes it a robust solution for testing on different vehicle platforms, experimenting with new controllers, or path planning techniques.

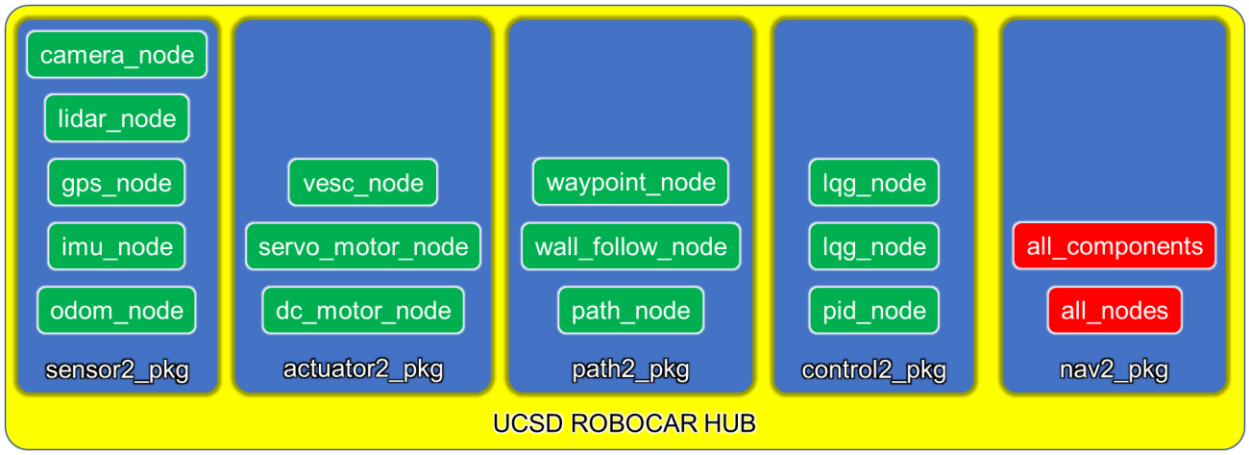

Figure 36: UCSD Robocar framework scheme

The navigation package is essentially the node manager of the UCSD Robocar framework because it keeps track of all the names and locations of any node/launch file from the other packages used in the framework. This includes sensor and actuator drivers as well as navigation and control algorithms. This makes using the framework even simpler by removing the requirement of remembering all the different launch files and package names. This package dynamically builds a user-defined launch file at run-time through 2 different config files, 1 for hardware equipped to the robot and the other for the set of desired nodes to use for a given scenario. Direct application of this is shown later in **Error! Reference source not found.**.

The sensor package contains all the required nodes/launch files needed to use the sensors that are equipped to the car. The sensors include cameras, LiDAR, GPS, IMU, and wheel encoders.

The actuator package contains all the required nodes/launch files needed to use the actuators that are equipped to the car. The drivers include sensor-less brushed DC motors via pulse-width-modulation (PWM), sensored brushless DC motor and servo motors via PWM.

The path package contains all the required nodes/launch files needed to create trajectories for the car to follow in a pre-built map as well as in simulations. The path planners include waypoint following via GPS, bug algorithms via LiDAR scans, lane guidance via camera, Hector SLAM via LiDAR.

The control package contains all the required nodes/launch files needed to control the car with various methods such as manual joystick, manual keyboard, PID, and LQG

2.3.4.1 **Using the Framework**

The first thing to do is update config files. The framework is setup such that most things a user would need to change or modify can be done in a config file. If more intricate modifications must be done, its recommended to use what currently works and create a new node to integrate new behavior into the robot. This is explained more in **Error! Reference source not found.**. The following sections go over how to modify the config files to achieve a certain behavior such us turning on sensors, manual control, lane guidance, etc.

2.3.4.2 **Updating Parameters**

Hardware configuration is as simple as flipping a switch. Since the launch files in ROS2 are now in python, we can dynamically build launch files. Meaning there is no need to have several different “car configurations” that may have different hardware on them and instead have a single launch file that can launch any needed component by changing a single number (that number is explained below). There is only one file to modify and all that needs to be changed is either putting a “0” or a “1” next to the list of hardware in the file. To select the hardware that the robot has and that is required for the application, put a “1” next to it otherwise put a “0” which means it will not activate.

Modify and save the car config with the sensors and actuators on the robot and then recompile.

From the terminal

```
source_ros2
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/car_config.yaml
build_ros2
```

Very similar to hardware configuration, the nodes to use for a desired application is also as simple as flipping a switch. There is only one file to modify and all that needs to be changed is either putting a “0” or a “1” next to the list of robot behaviors/features. To select the desired behaviors/features that the robot is going to perform for the application, put a “1” next to it otherwise put a “0” which means it will not activate.

Modify and save the node config and then recompile.

From the terminal

```
source_ros2
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/node_config.yaml
```

```
build_ros2
```

Vehicle parameters spanning from mass and length to sensor noise covariances, and controller performance weights are all configurable outside of the source code. This means that the user should never need to modify the source code to update any given parameter. Below are the commands to update any parameter for the vehicle and controller performance.

From the terminal

```
source_ros2
gedit src/ucsd_robocar_hub2/ucsd_robocar_control2_pkg/config/car_config.yaml
build_ros2
```

Being able to manually control the vehicle is very useful and convenient when either collecting data, troubleshooting hardware, or debugging. Below are the required commands to get the car moving in manual mode.

Controls

- A “deadman” switch is enabled which means that button ***must*** be pressed and held (LB on logitech) down in order for commands to be sent to the robot’s motors.
- The joysticks on the controller are what control the robot to move forwards/backwards and turn.

From the terminal

```
source_ros2
```

Modify the hardware config file to turn on the **vesc_with_odom**

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/car_config.yaml
```

Then modify the node config file to activate **all_components** and **f1tenth_vesc_joy_launch** launch files

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/node_config.yaml
```

Then rebuild and launch

```
build_ros2
ros2 launch ucsd_robocar_nav2_pkg all_nodes.launch.py
```

2.3.4.3 **Sensor Visualization**

After selecting the hardware that's equipped on the robot, let's visually verify that the sensors are working. The current config file that is launched will display laser scan and image data. If the robot has more sensors to visualized, they can be added through RVIZ.

Modify the hardware config file to turn on the **sensors** plugged in and want to visualize.

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/car_config.yaml
```

Then modify the node config file to activate **all_components** and **sensor_visualization** launch files

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/node_config.yaml
```

Then rebuild and launch

```
build_ros2
ros2 launch ucsd_robocar_nav2_pkg all_nodes.launch.py
```

2.3.4.4 **Data Collection**

To collect data being broadcasted over the topics that are actively being published, turn on whichever nodes needed to publish that topic information but make sure that the *rosbag_launch* option in the node_config is also turned on which is the switch for data collection. This will record ALL topics to the “rosbag” which is a unique file type to ROS. Then

a package called bagpy is used to convert the data into csv format which is useful for viewing/analysis.

Modify the hardware config file to turn on any sensors equipped and needed for data collection

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/car_config.yaml
```

Then modify the node config file to activate only *all_components*, *rosbag_launch* launch files

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/node_config.yaml
```

Then rebuild and launch

```
build_ros2
ros2 launch ucsd_robocar_nav2_pkg all_nodes.launch.py
```

Figure 37 shows a diagram of all the nodes that need to be activated during the system identification process which consists of all the sensors, actuators, and control methods.

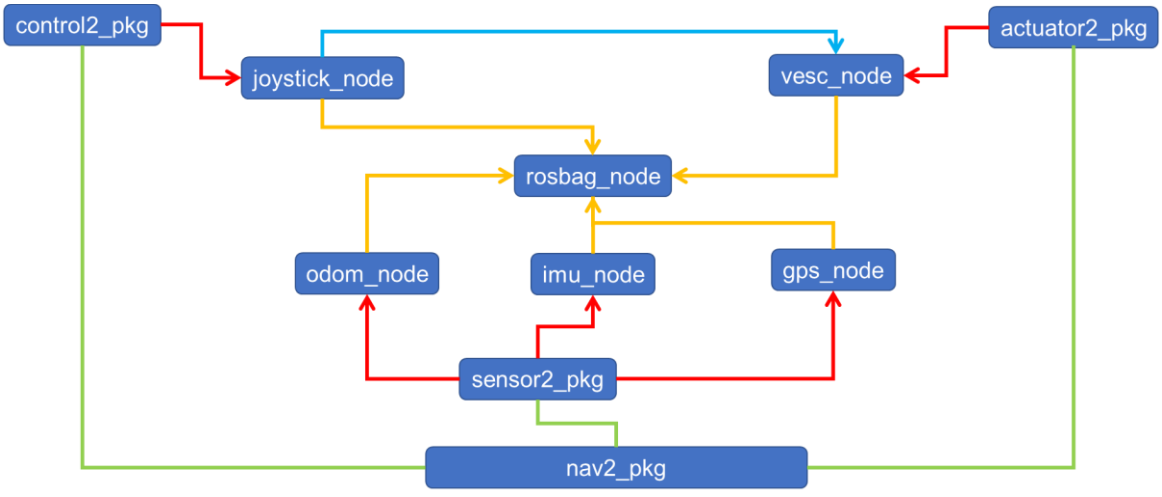

Figure 37: UCSD Robocar system ID node tree

2.3.4.5 **Running Simulator**

A lightweight ROS2 simulator created by the F1 Tenth community using RVIZ was used for various scenarios such as model validation, experiment repeatability and general experimentation. The simulator uses a 2D dynamic bicycle-car model to simulate how the car would move in an environment. There are several default maps available in the simulator, but it is also possible to load in custom maps that could have been generated via SLAM techniques or by CAD drawings.

NOTE: For the example below, the joystick is used as the controller so the user will need a controller plugged into their computer. For manual control, the path planner is not activated.

NOTE: Only use the simulator on the X86 docker image and not the Jetson.

Then modify the node config file to activate only the *simulator* and *f1tenth_vesc_joy_launch* launch files

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/node_config.yaml
```

Modify the f1 tenth simulator config file to update the map (if needed)

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/f1_tenth_sim.yaml
```

Then rebuild and launch

```
build_ros2
ros2 launch ucsd_robocar_nav2_pkg all_nodes.launch.py
```

2.3.4.6 **Autonomous Mode with LQG**

After verifying that all the sensors/hardware are working properly, system ID has been completed, and weighting matrices have been resolved, tested controller in the simulator, it is now time to test out the LQG controller on the physical robot.

From the terminal (if haven't done already)

```
source_ros2
```

Modify the hardware config file to turn on the *vesc_with_odom*

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/car_config.yaml
```

Then modify the node config file to activate *all_components* and *f1tenth_vesc_joy_launch* launch files

```
gedit src/ucsd_robocar_hub2/ucsd_robocar_nav2_pkg/config/node_config.yaml
```

Then rebuild and launch

```
build_ros2
```

```
ros2 launch ucsd_robocar_nav2_pkg all_nodes.launch.py
```

Figure 38 shows a diagram of all the nodes that need to be activated during the autonomous operation which consists of everything from the system identification process as well as the path planner.

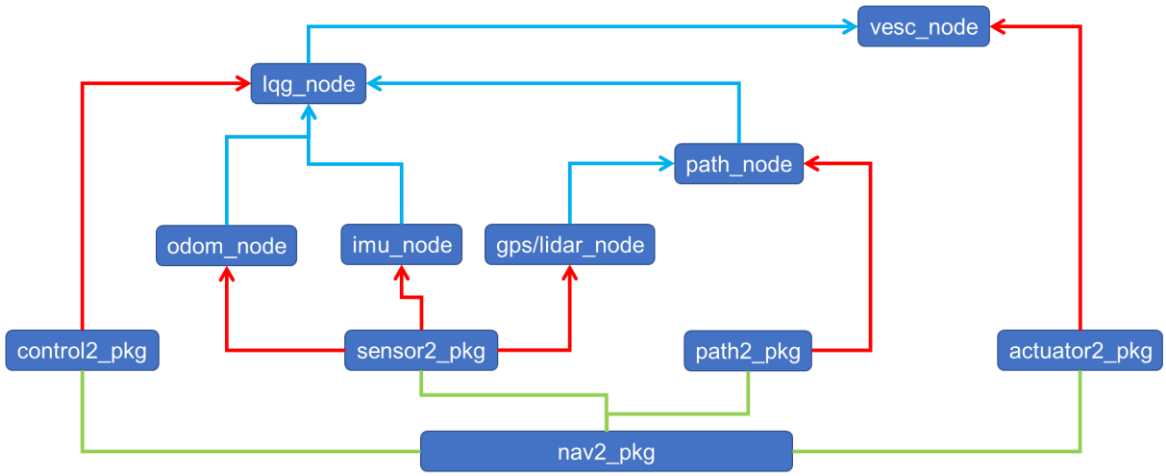

Figure 38: UCSD Robocar Autonomous node tree

2.4 **Experimental Procedures**

2.4.1 **Sensor Calibration**

2.4.1.1 **PWM to Steering Wheel Angle**

This experiment was to convert the pulse-width-modulation (PWM) signal data (input data type to steering column motor on the car) to road wheel angle. The experiment flow goes as follows at zero speed:

- From center position to max left/right limits of steering, record PWM values and measure wheel angle deflection from the vehicle's longitudinal axis directly (using geometry)
- Repeat a minimum of 3 trials

Now at low speed in manual mode:

- Find $bias$ by finding the PWM value that puts the wheel angles in the straight position and that the path taken by the robot follows a straight line for at least 3m

An example of finding the bias can be seen in Figure 39. Then Least squares can be performed to determine the gain (slope) and offset/bias (y-intercept) of the linear relationship.

$$\delta = Kpwm + bias$$

Now the vehicle can be controlled with wheel angles vs PWM.

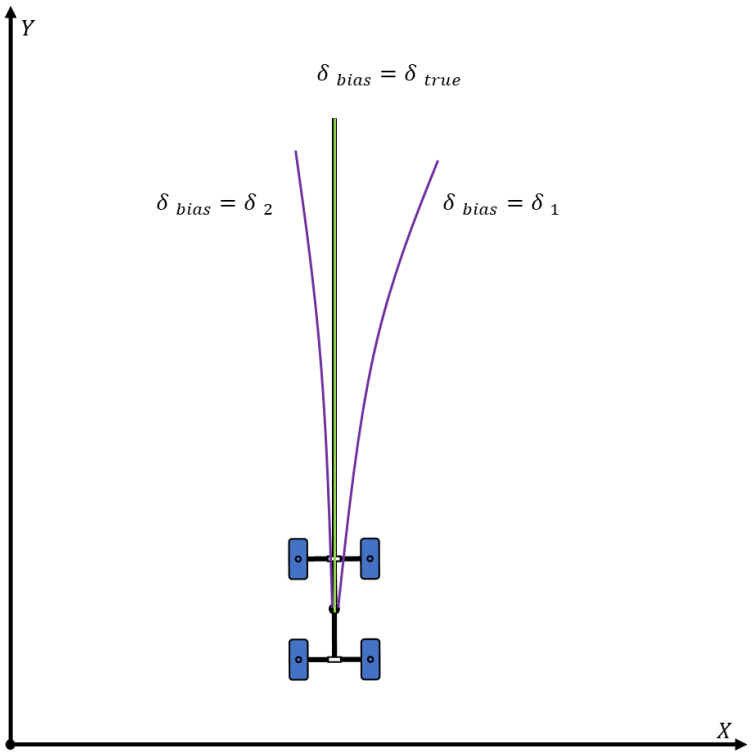

Figure 39: PWM to Steering Wheel Angle

2.4.1.2 **Steering Wheel Angle to Road Wheel Angle**

This experiment was to convert the normalized input servo motor angle to the road wheel angle of the car. The experiment flow goes as follows for a constant speed

- Lock servo angle at some angle (θ_i)
- Measure the wheel angle (δ_i)
- Repeat a minimum of 3 trials

This was done for several discrete positions of the vehicles steering range

Then Least squares can be performed to determine the gain (slope) and offset/bias (y-intercept) of the linear relationship.

$$\delta = K\theta + bias$$

Now the vehicle can be controlled with wheel angles vs servo motor position.

2.4.1.3 **RPM to Vehicle Speed**

This experiment was to convert the input motor RPM (revolutions per minute) to the longitudinal velocity of the car. The experiment flow goes as follows for a particular input motor RPM value:

- Lock steering angle in straight position
- Have a “buffer” zone set to not measure transient
- Measure Δt over the pre-defined distance of travel
- Repeat a minimum of 3 trials

This was done for several discrete rpm values of the DC motor to characterize the full relationship. An example of the setup can be seen in Figure 40. Then Least squares can be performed to determine the gain (slope) and offset/bias (y-intercept) of the linear relationship.

$$V_x = K\omega + bias$$

Now the vehicle can be controlled with vehicle speeds vs motor RPM.

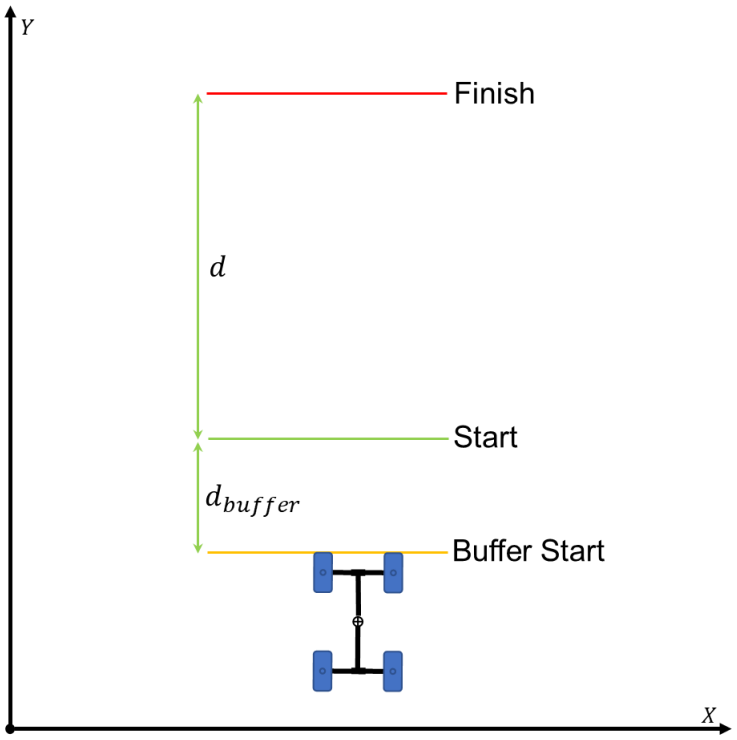

Figure 40: RPM to longitudinal velocity

The gain in this experiment relates to the effective gear ratio which is solved analytically as follows

$$\omega_{motor} N_{motor} = \omega_{clutch} N_{clutch}$$

$$\omega_{clutch} = \omega_{motor} \frac{N_{motor}}{N_{clutch}}$$

$$V_{wheel} = \omega_{clutch} R_{wheel}$$

$$V_{wheel} = \left(\omega_{motor} \frac{N_{motor}}{N_{clutch}}\right) R_{wheel}$$

$$V_{wheel} = \omega_{motor} \left(\frac{N_{motor}}{N_{clutch}} R_{wheel}\right) = \omega_{motor} GR_{effective}$$

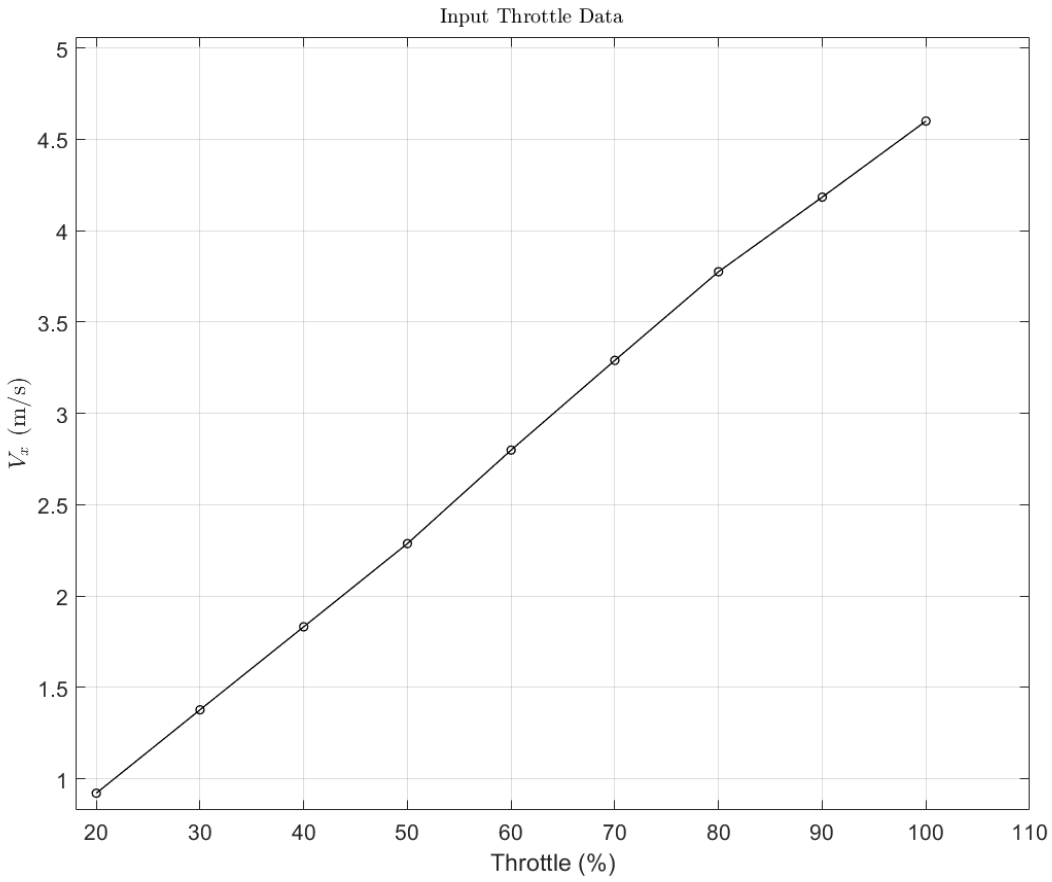

Figure 41: Speed to throttle correlation data

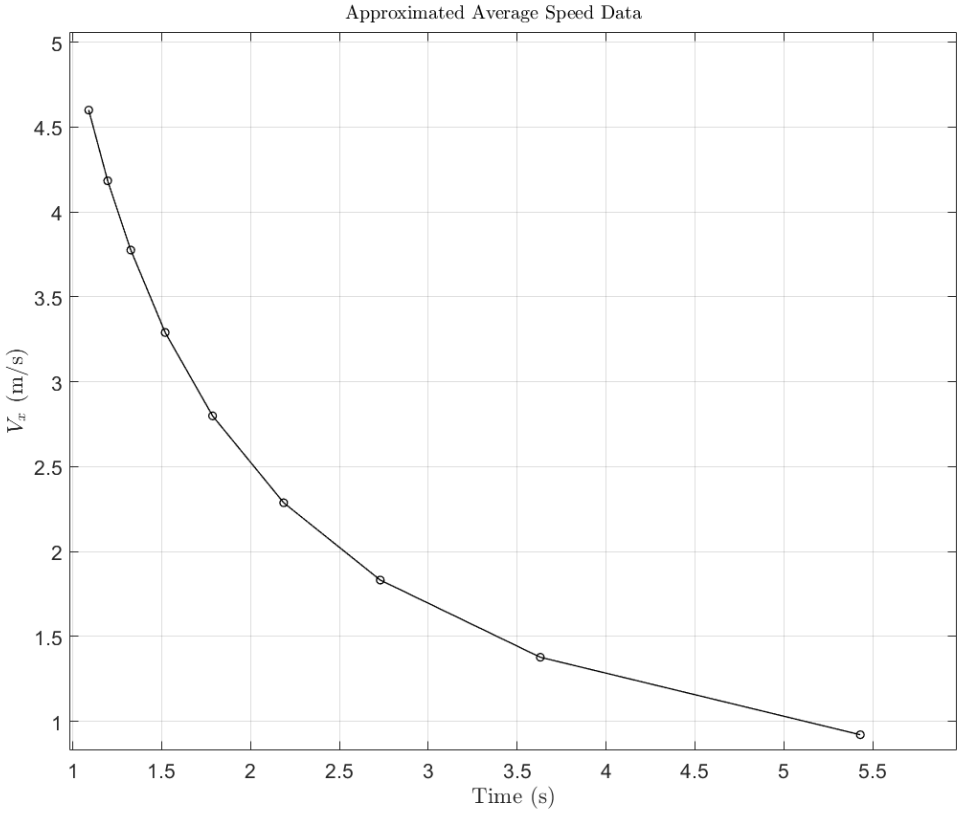

Figure 42: Measured speed over time

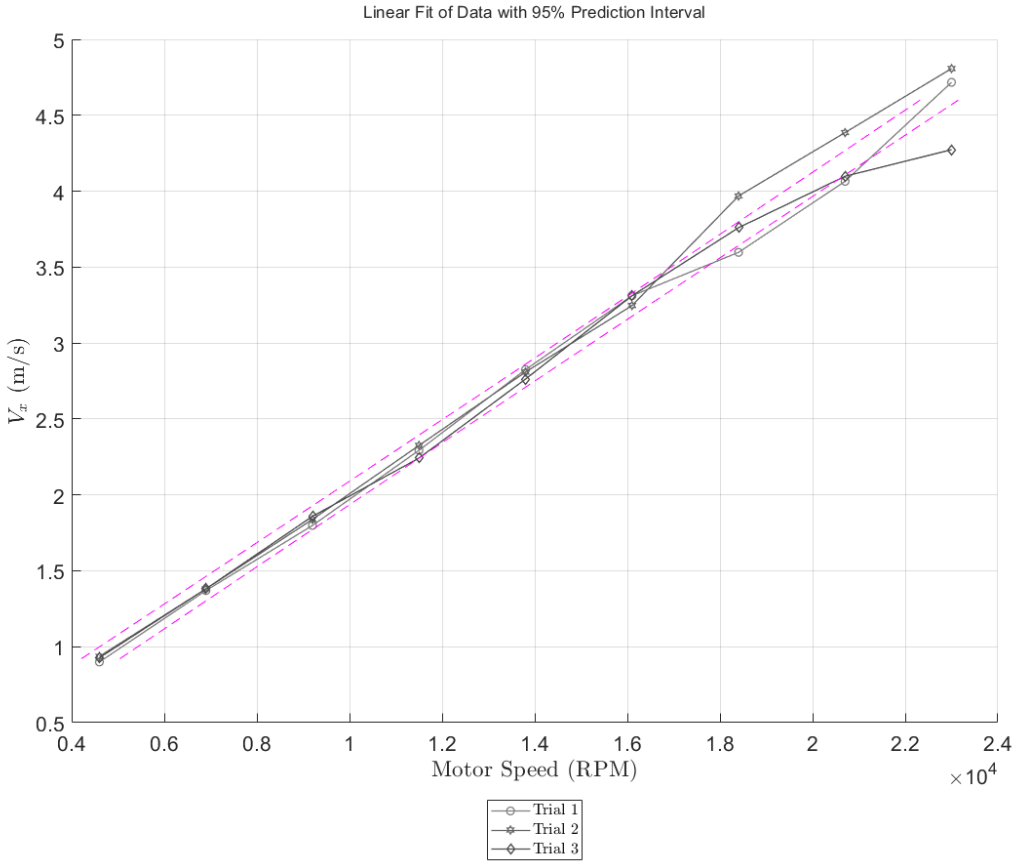

Figure 43: Correlation from RPM to longitudinal velocity

2.4.2 **Parameter Measurements**

Most parameters can be estimated well by measuring the mass of the vehicle. Depending on what kind of scales are available, the front and rear axles can be weighted or each individual tire and then summing them to get the total mass. Once the mass distribution is known, the distances from the CG are estimated by the weight distribution and the actual length of the vehicle which was measured simply with a measuring tape. Then the moment of inertia is calculated using the mass distribution and rotation axis distances.

2.4.2.1 **Tire Stiffness Coefficients**

This experiment was done by collecting data from a specified steering angle reference and constant speed. The reference signals were sinusoidal and varied in amplitude and frequency to make sure that the system was continuously excited which is a requirement for system ID and to have several independent data sets that can then be combined to get a more robust approximation for various lateral tire forces acting on the vehicle. The experiment flow goes as follows:

- Set constant vehicle speed, amplitude, and frequency of steering angle (max steering angle, rotation speed)
- Collect, steering, GPS, and IMU data for length of experiment
- Repeat a minimum of 3 trials

Then with this data, least squares using eq.(1.31) can be performed to approximate these parameters. The steering input and sensor measurements can be seen in Figure 45, Figure 46, and Figure 47. The models given by eq.(1.17) and eq.(1.18) can then be used for validation which the results can be seen in Figure 48 and Figure 49 respectively.

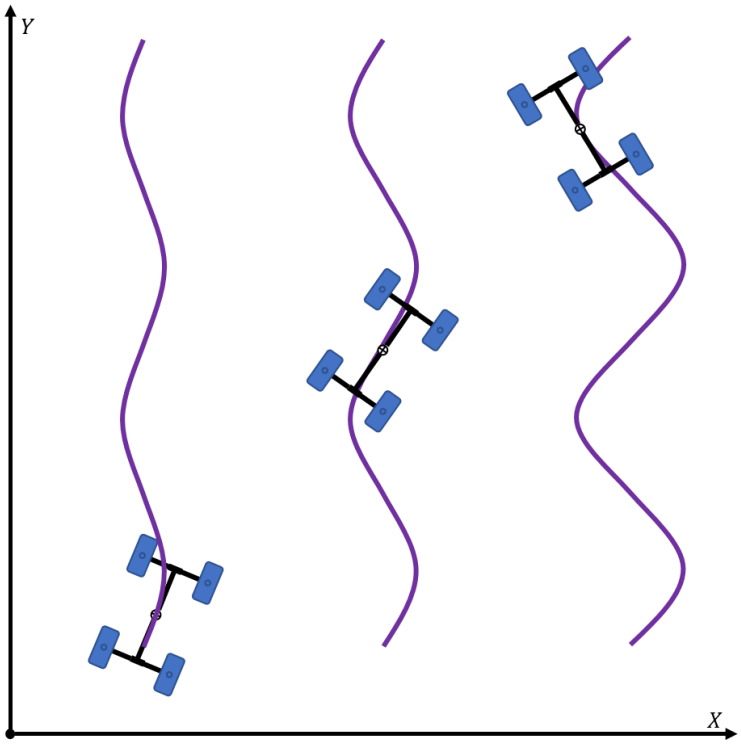

Figure 44: Tire Stiffness Coefficients Experiment

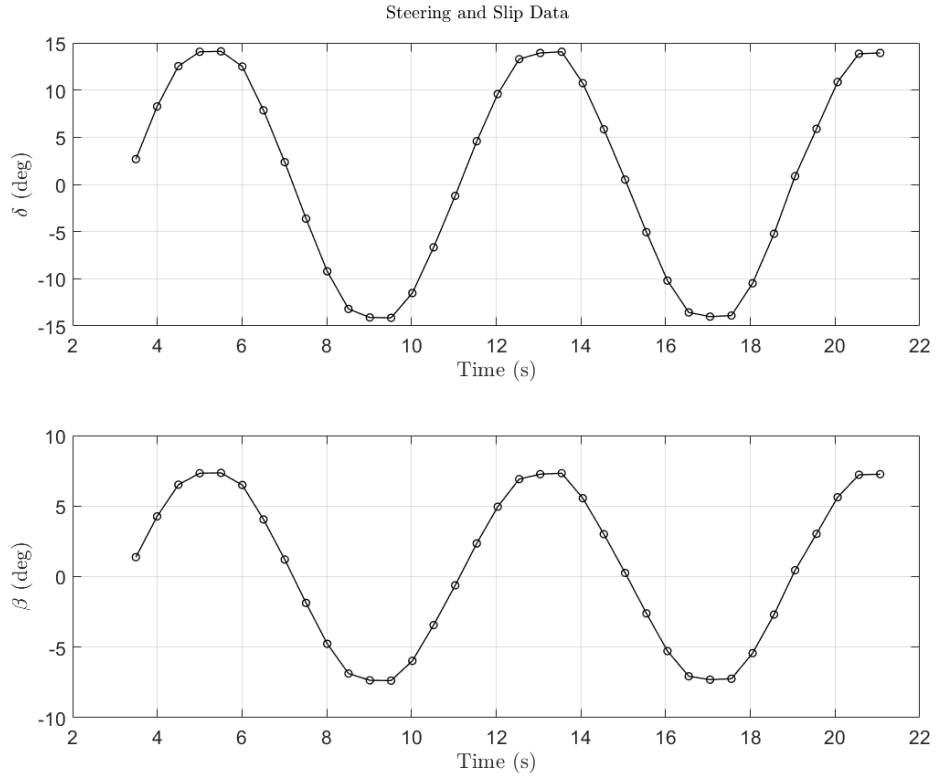

Figure 45: Steering and slip angle measurements

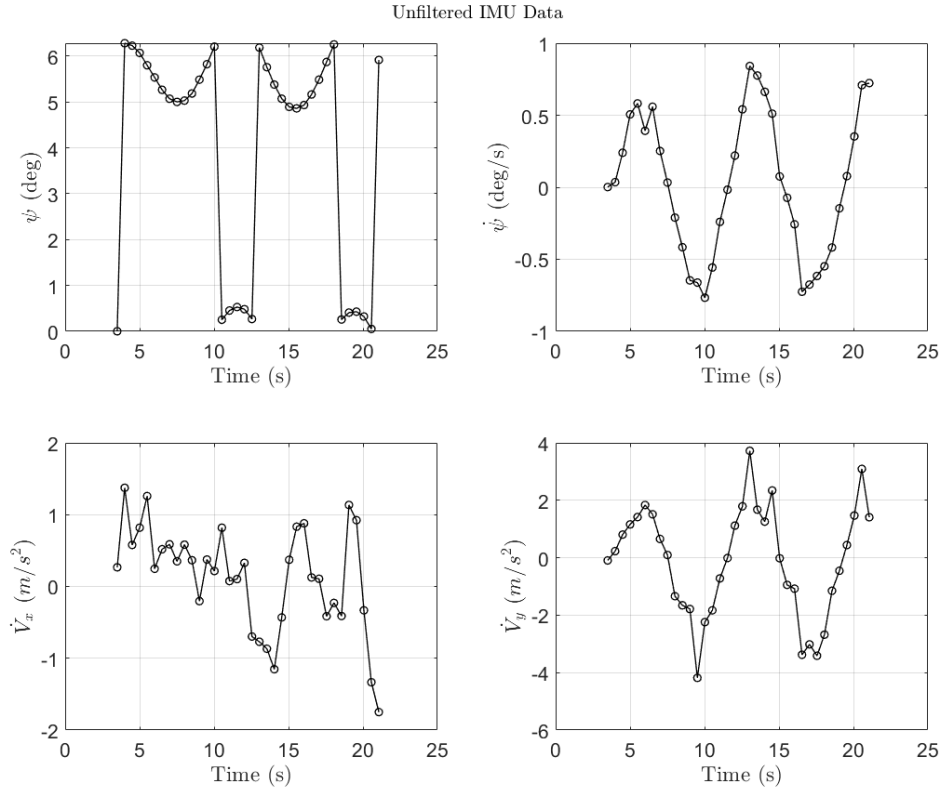

Figure 46: IMU measurements

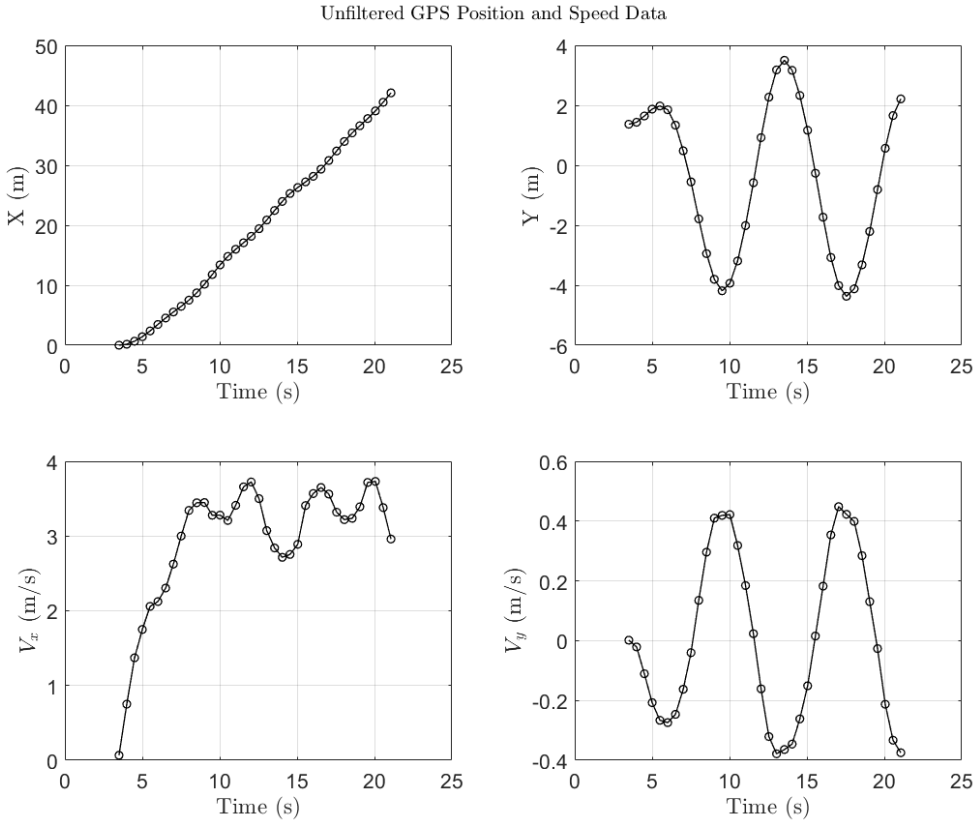

Figure 47: GPS measurements

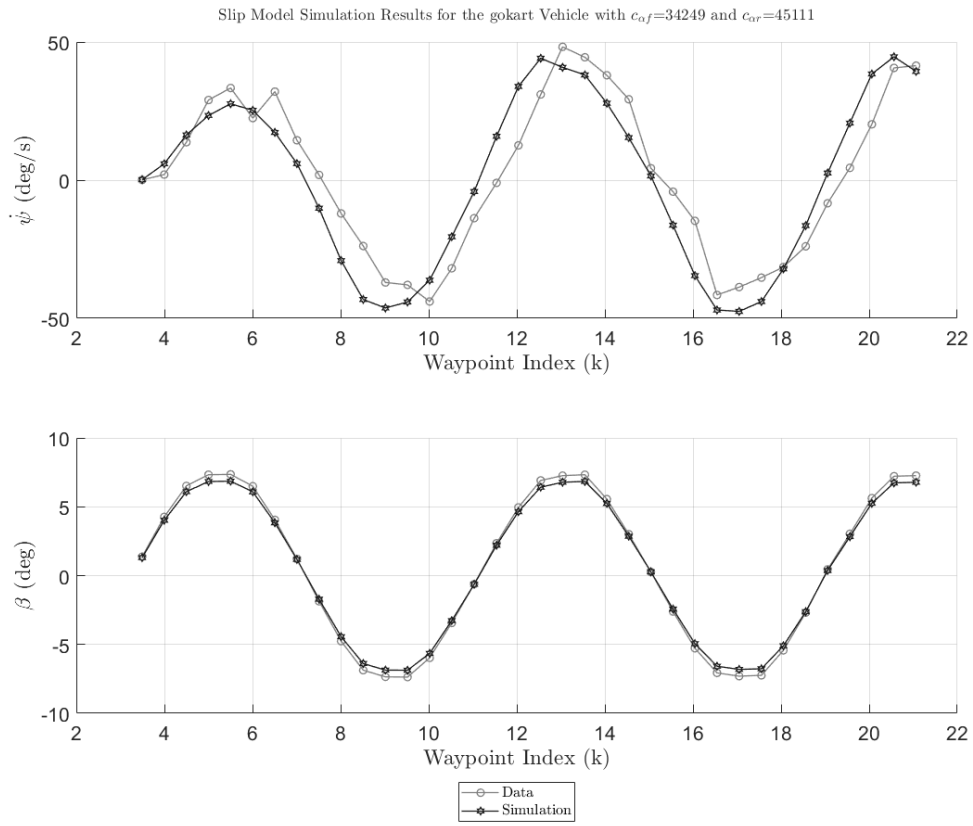

Figure 48: Slip model simulation results from ls

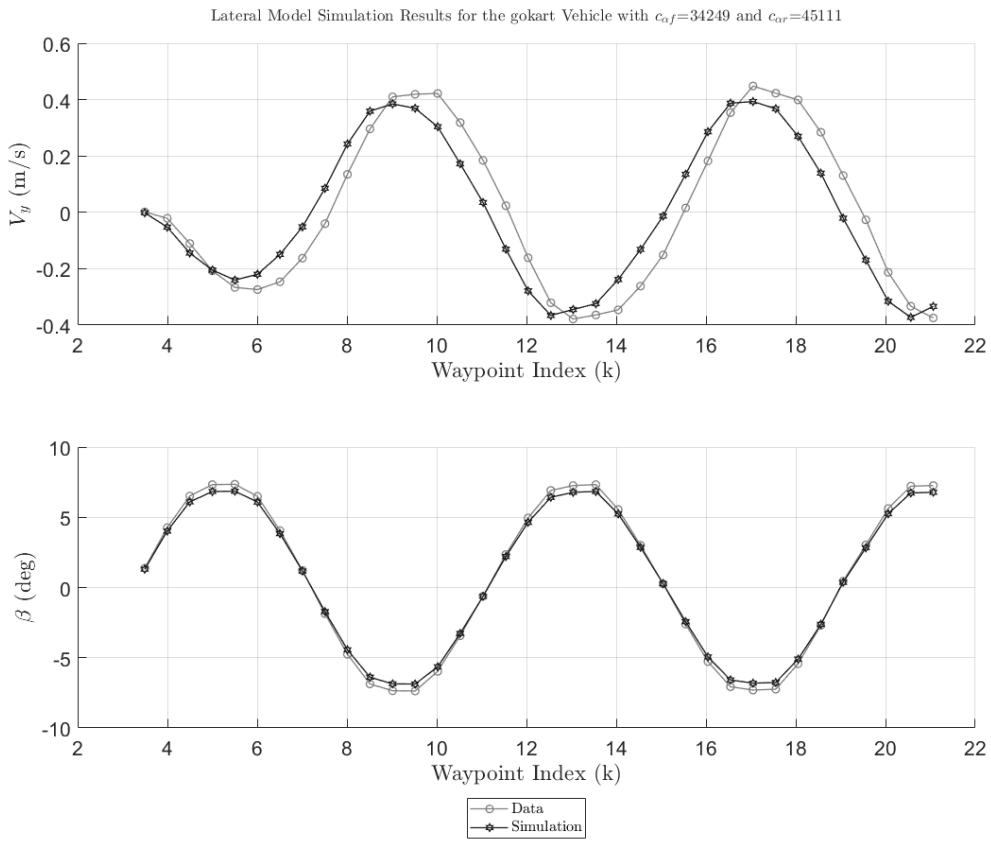

Figure 49: Lateral model simulation with

Appendix A Path Catalog

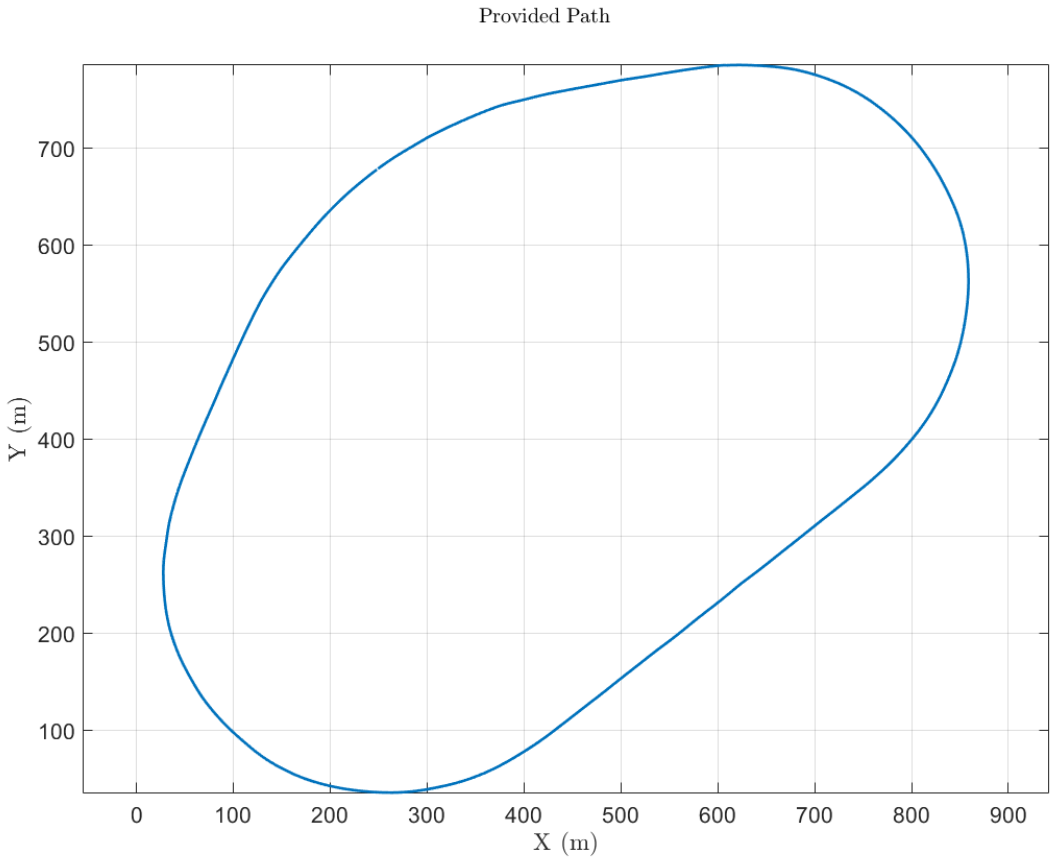

Figure 50: Las Vegas Motor Speedway (LVMS) track

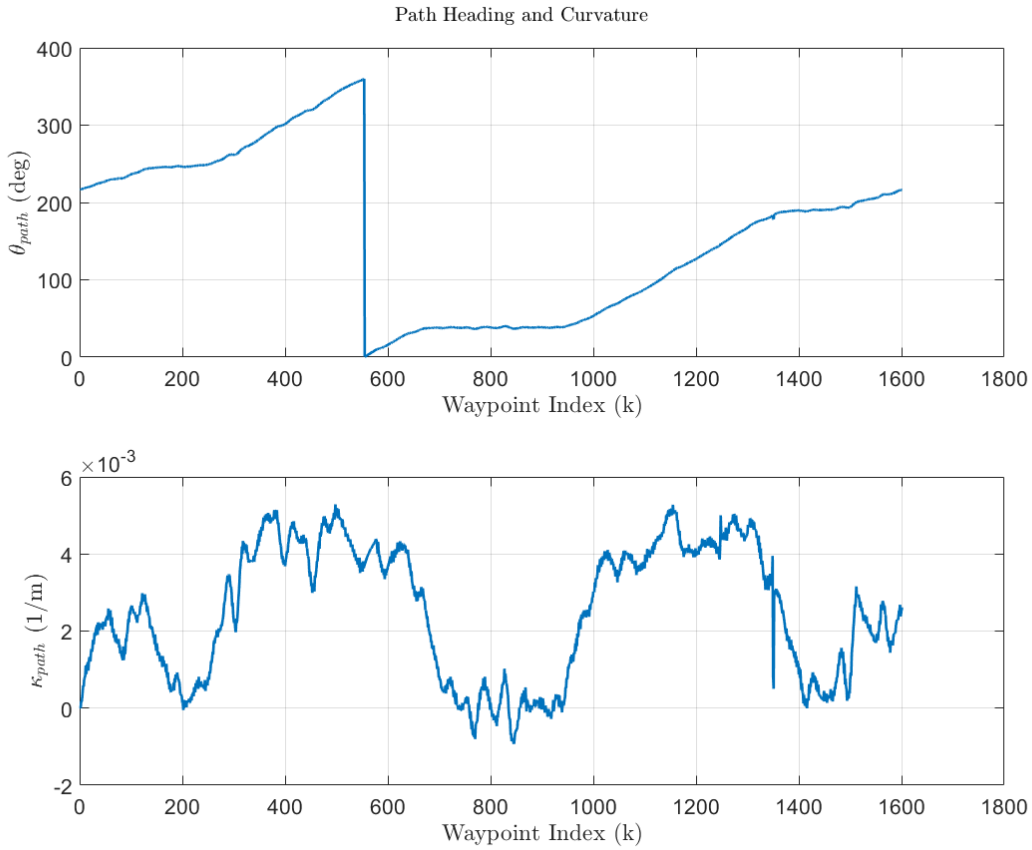

Figure 51: LVMS track characteristics

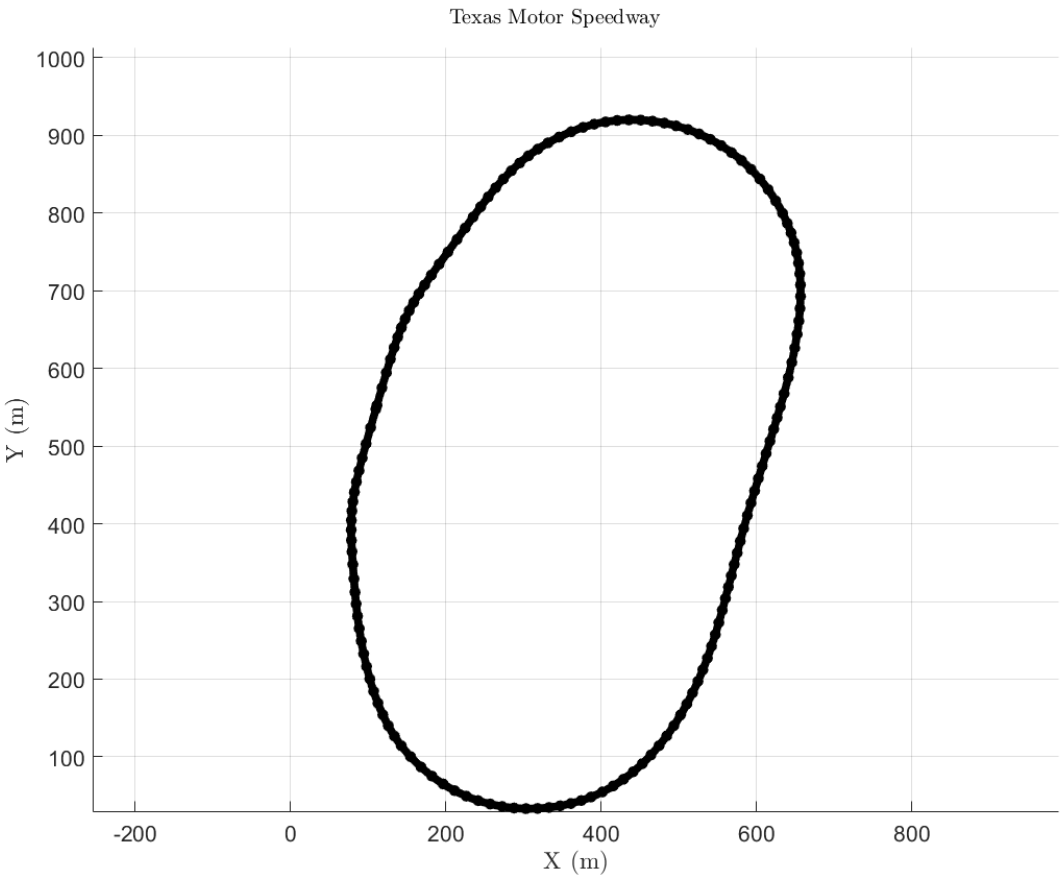

Figure 52: Texas Motor Speedway (TMS) track data

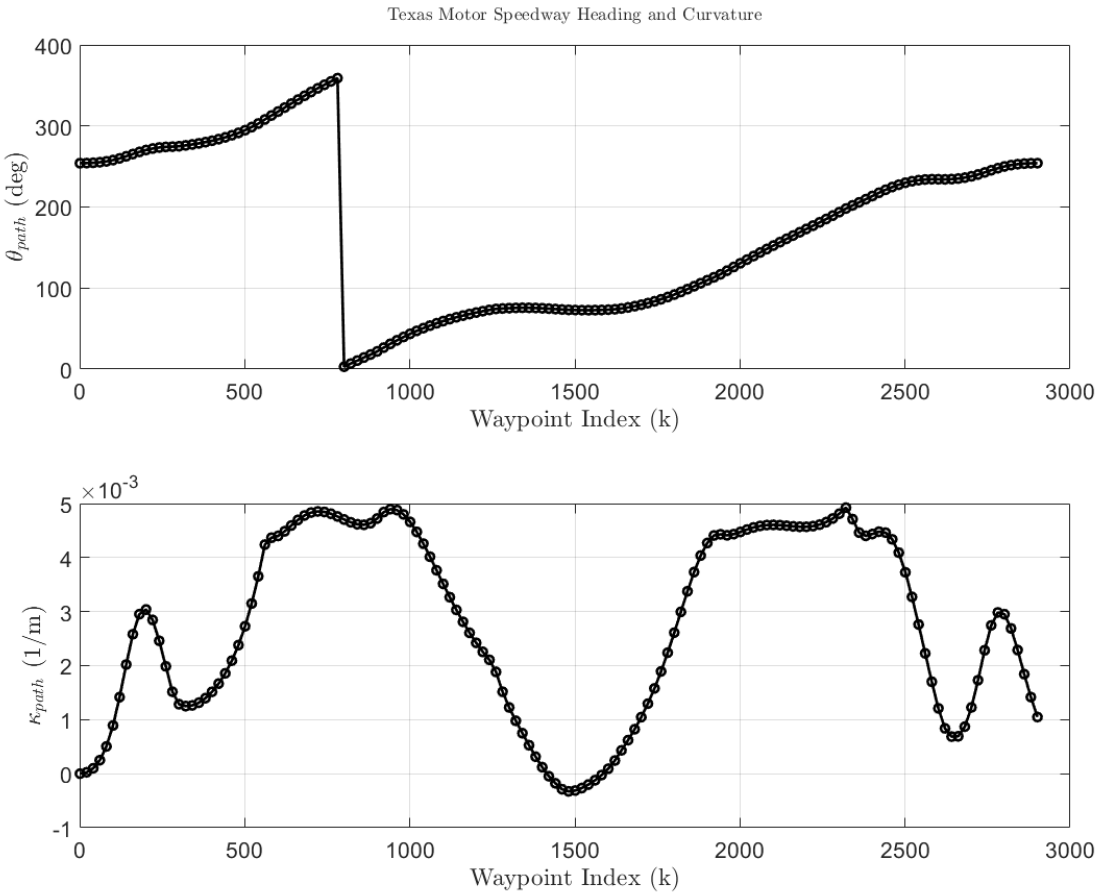

Figure 53: TMS track characteristics

Figure 54: Purdue track data

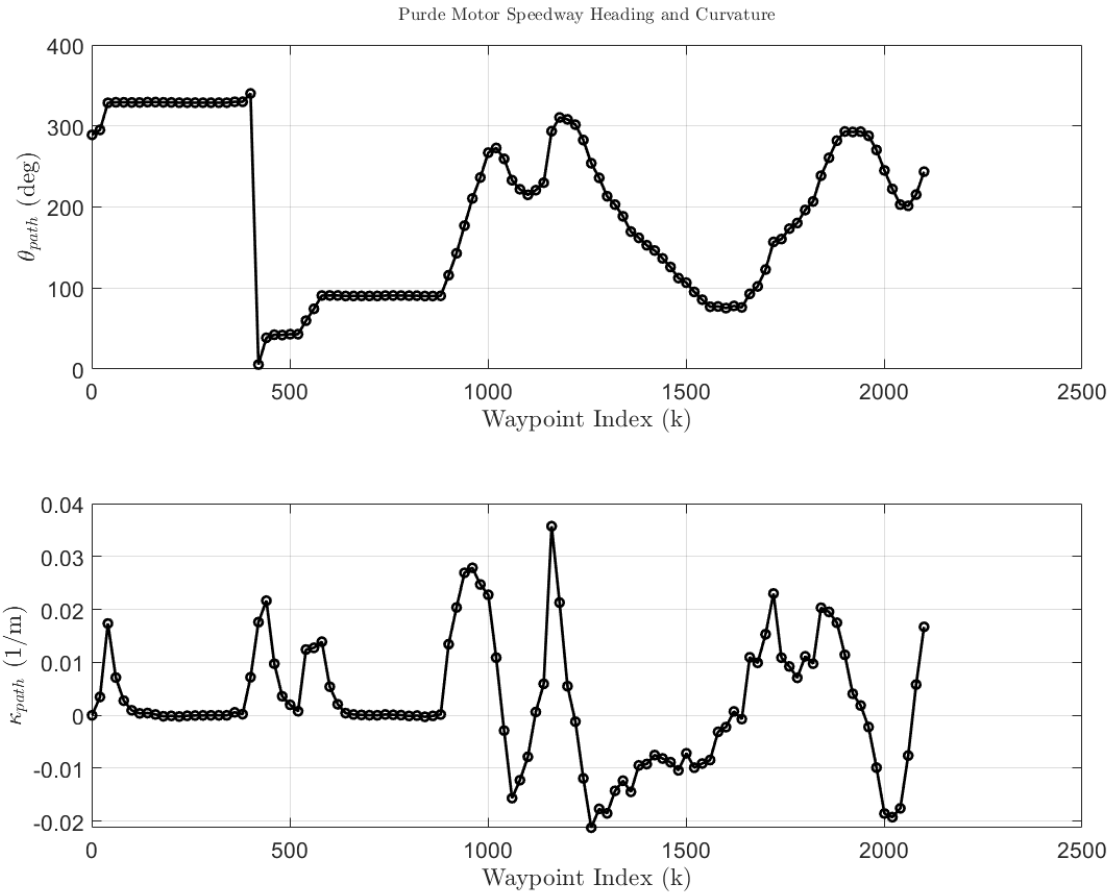

Figure 55: Purdue track characteristics

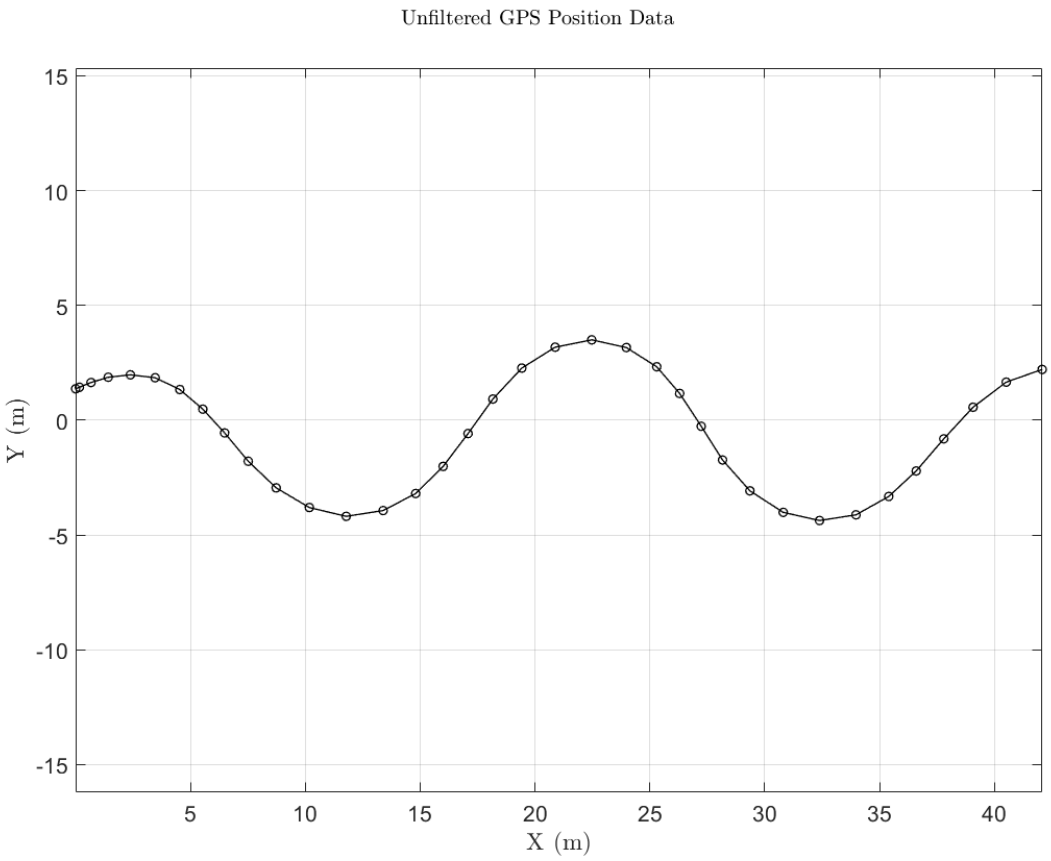

Figure 56: System ID path 1

Appendix B Controller Performance

B.1 Without feedforward

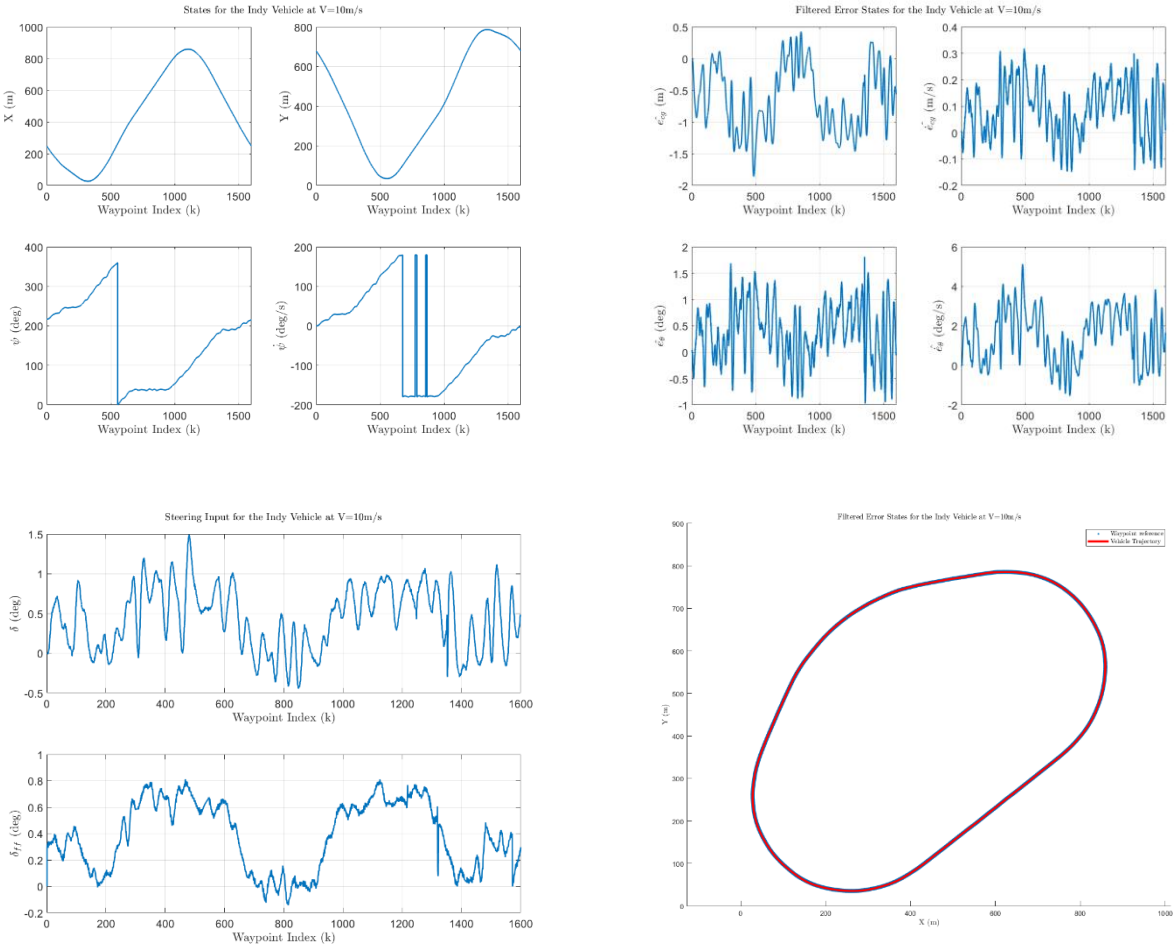

Figure 57: LVMS without δ_{ff} and V_x=10m/s

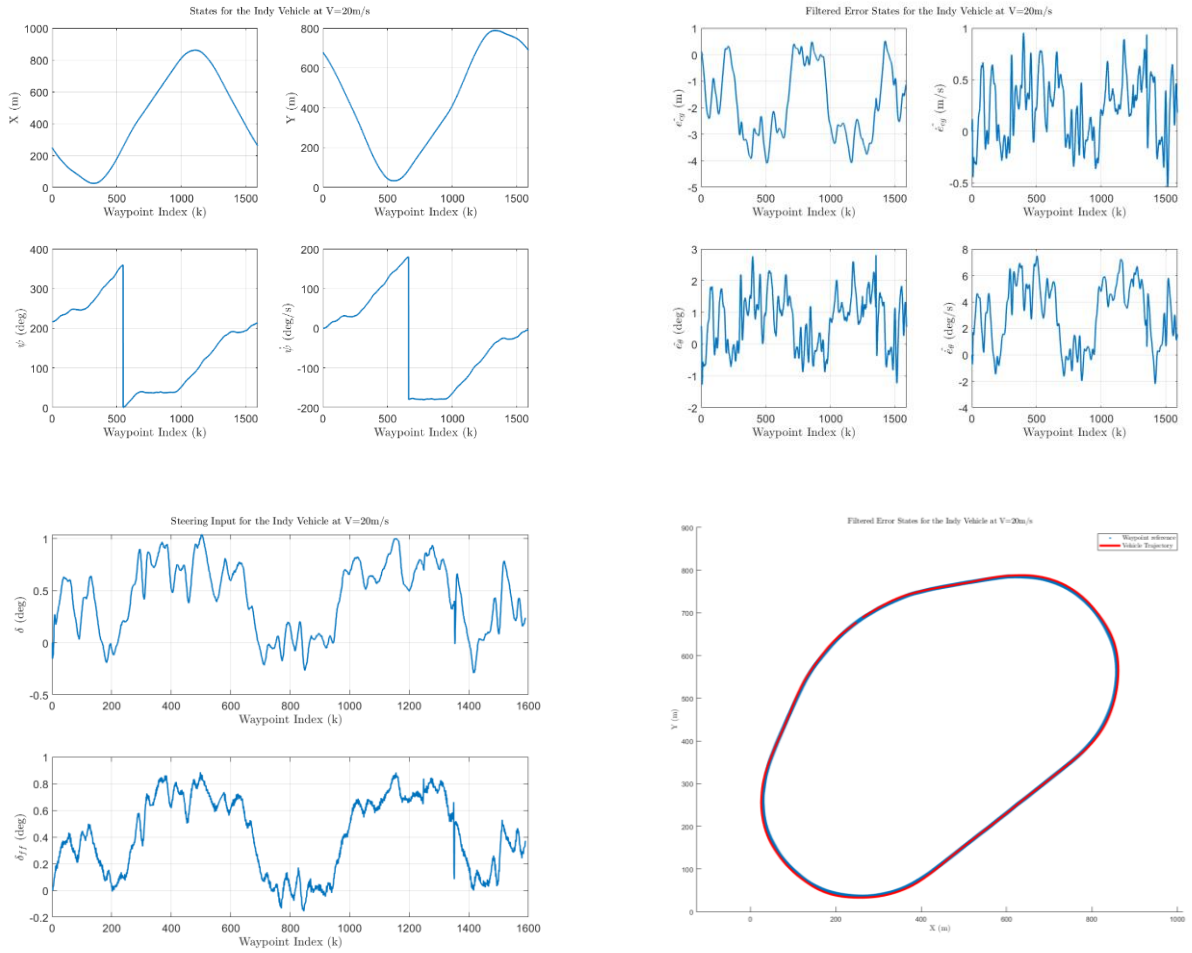

Figure 58: LVMS without δ_{ff} and V_x=20m/s

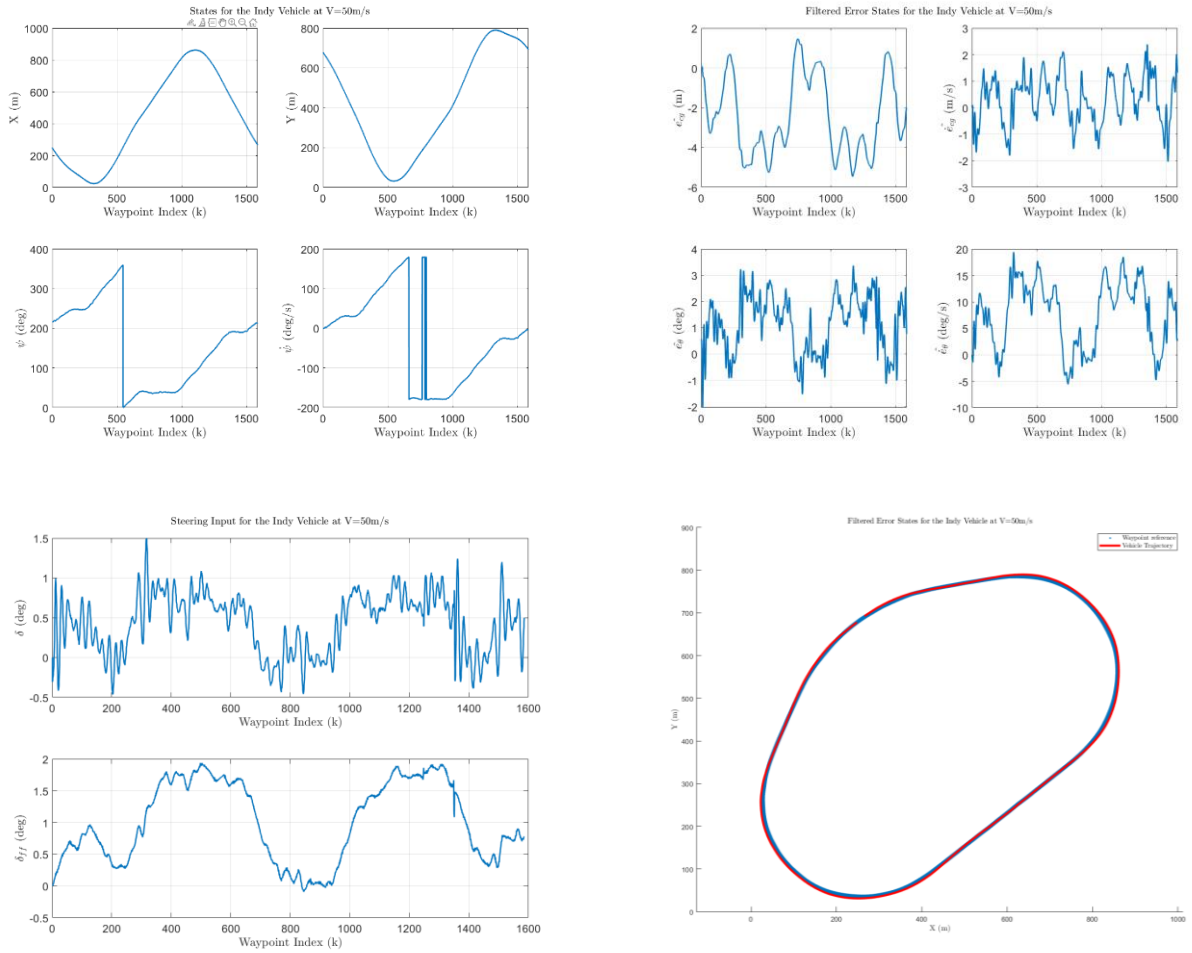

Figure 59: LVMS without δ_{ff} and V_x=50m/s

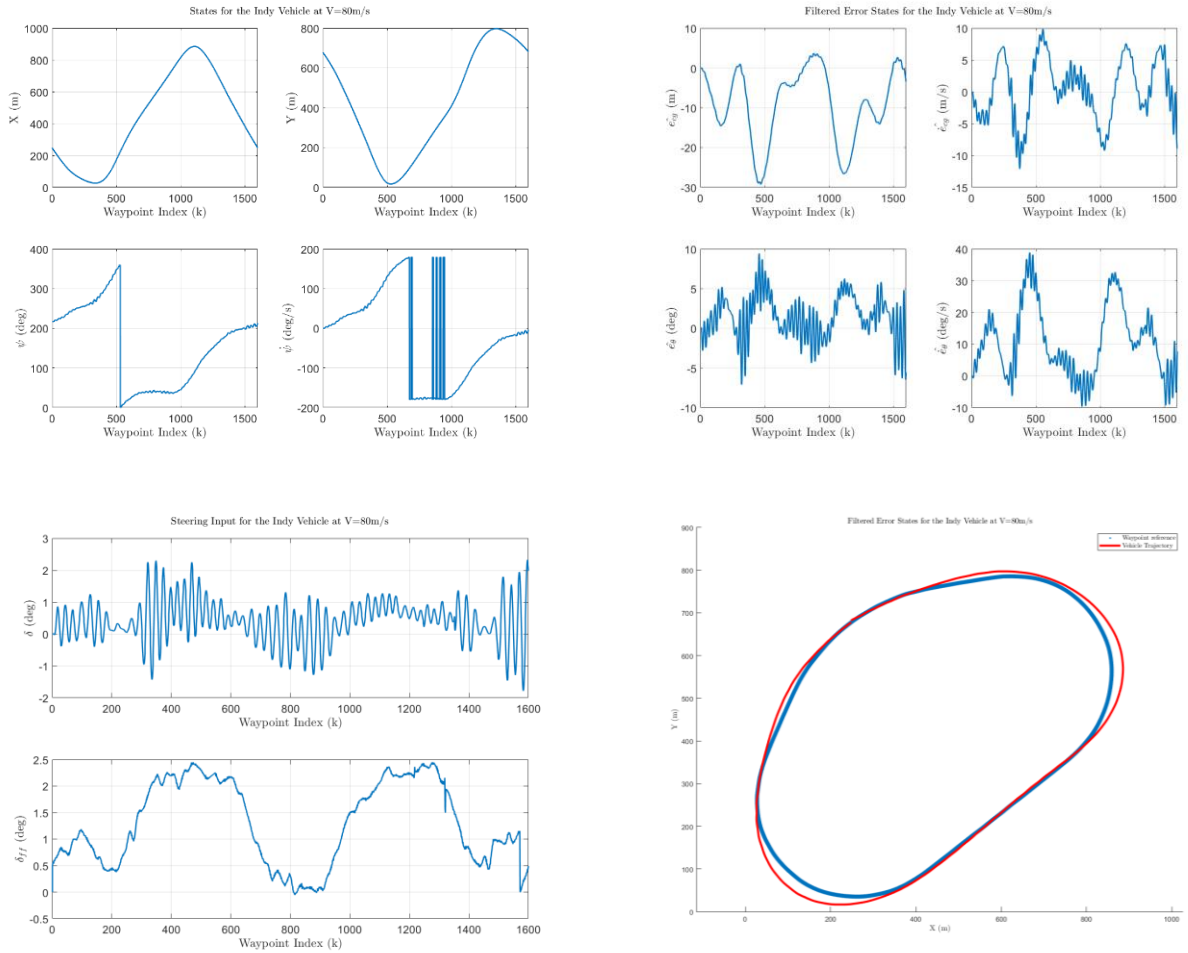

Figure 60: LVMS without δ_{ff} and V_x=80m/s

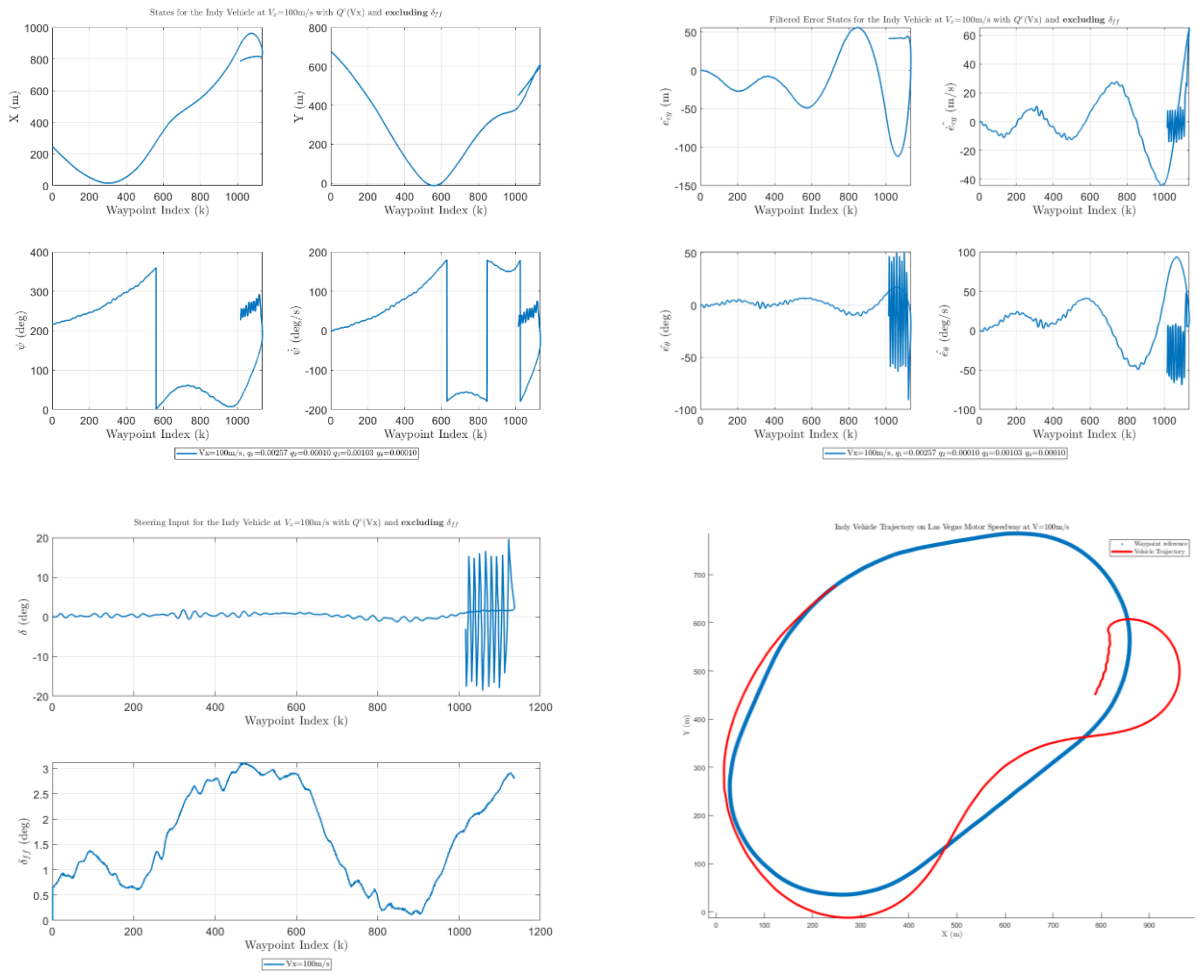

Figure 61: LVMS *without* δ_{ff} and V_x=100m/s

B.2 With feedforward

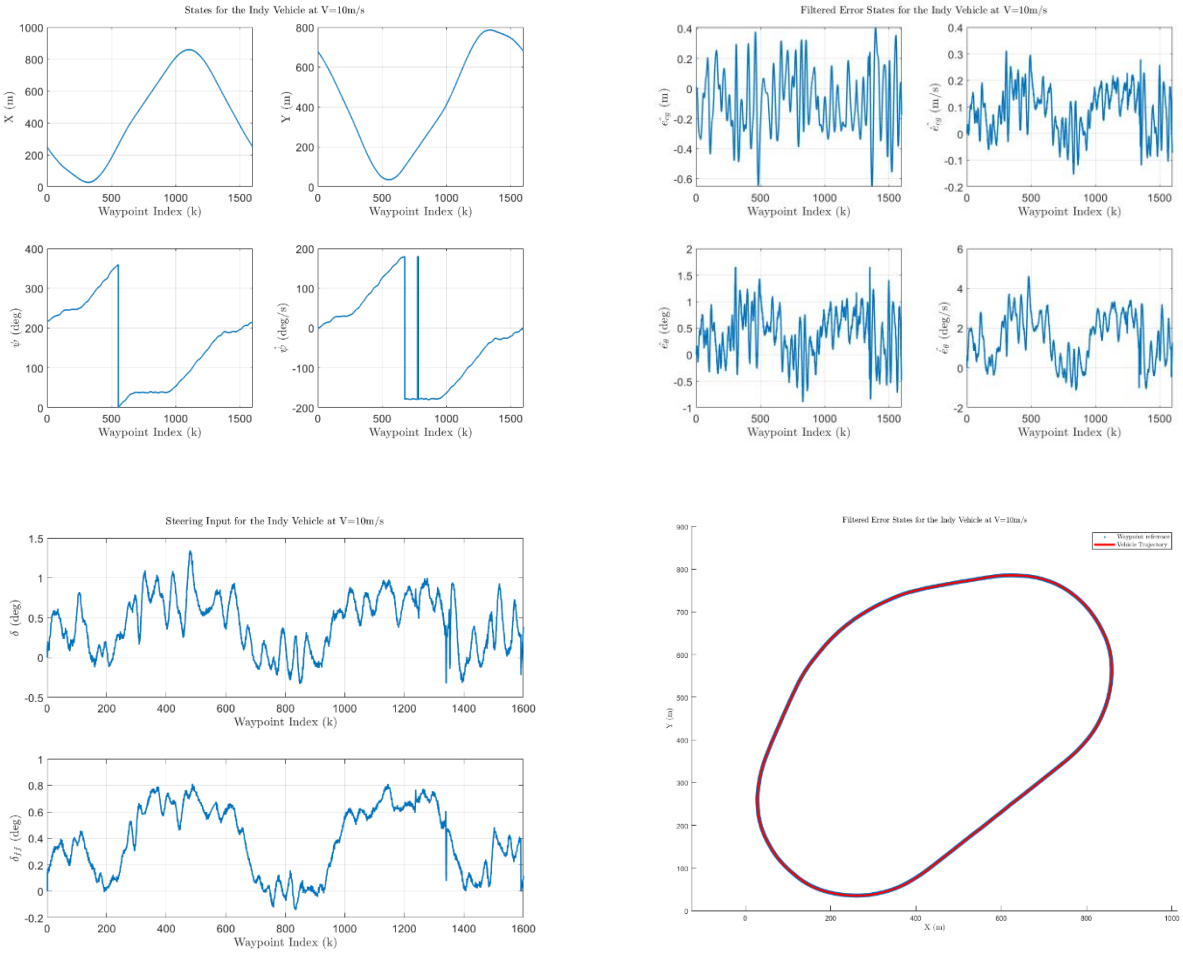

Figure 62: LVMS with δ_{ff} and V_x=10m/s

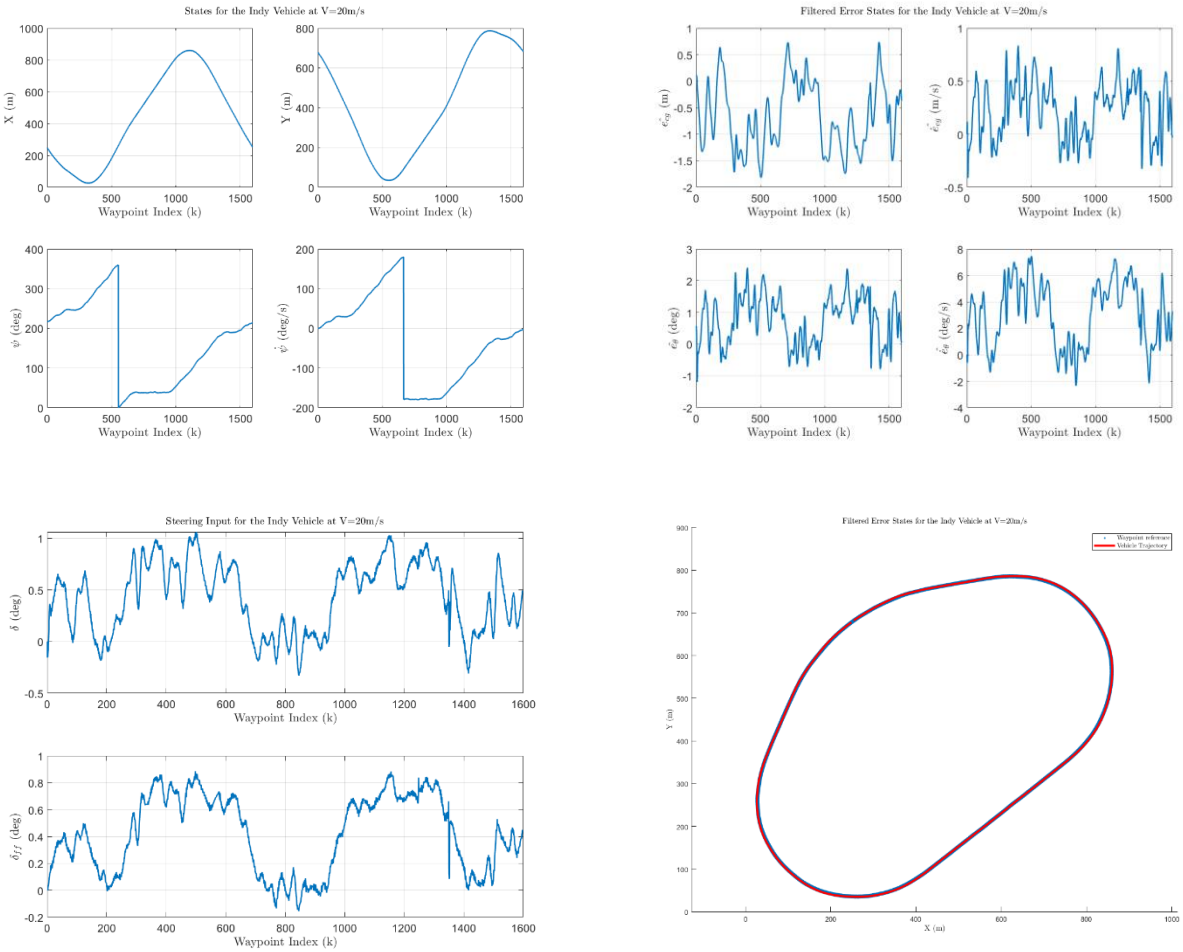

Figure 63: LVMS with δ_{ff} and V_x=20m/s

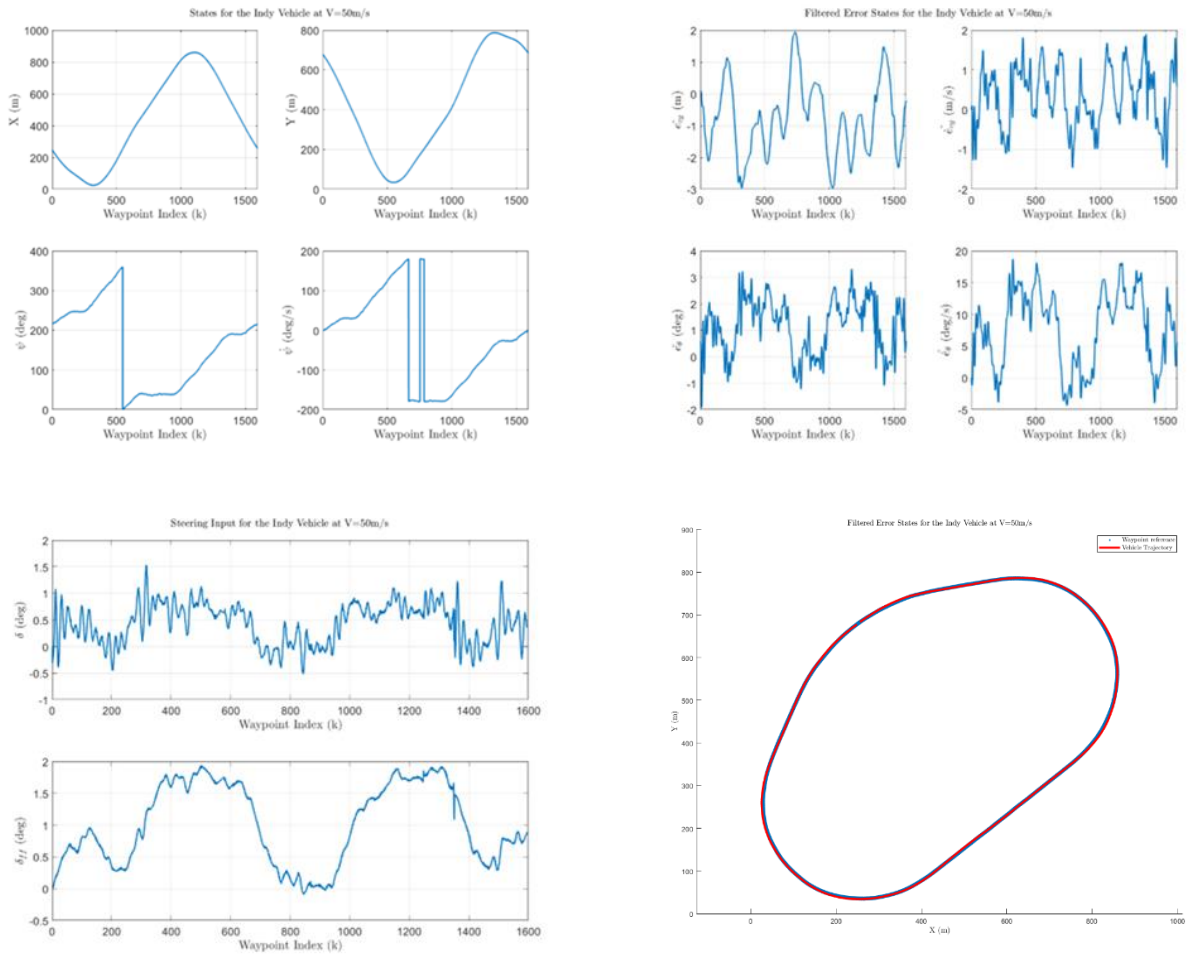

Figure 64: LVMS with δ_{ff} and V_x=50m/s

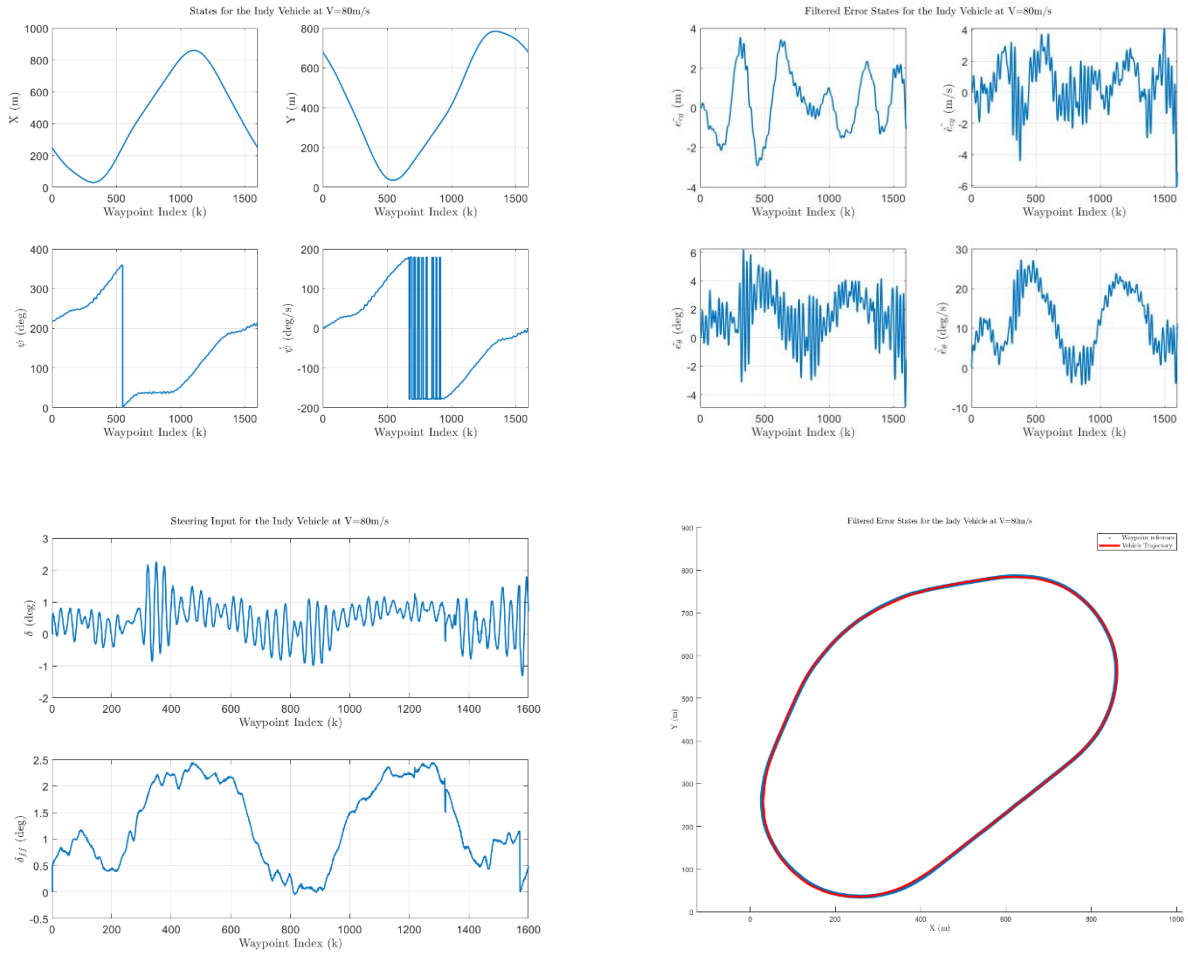

Figure 65: LVMS with δ_{ff} and V_x=80m/s

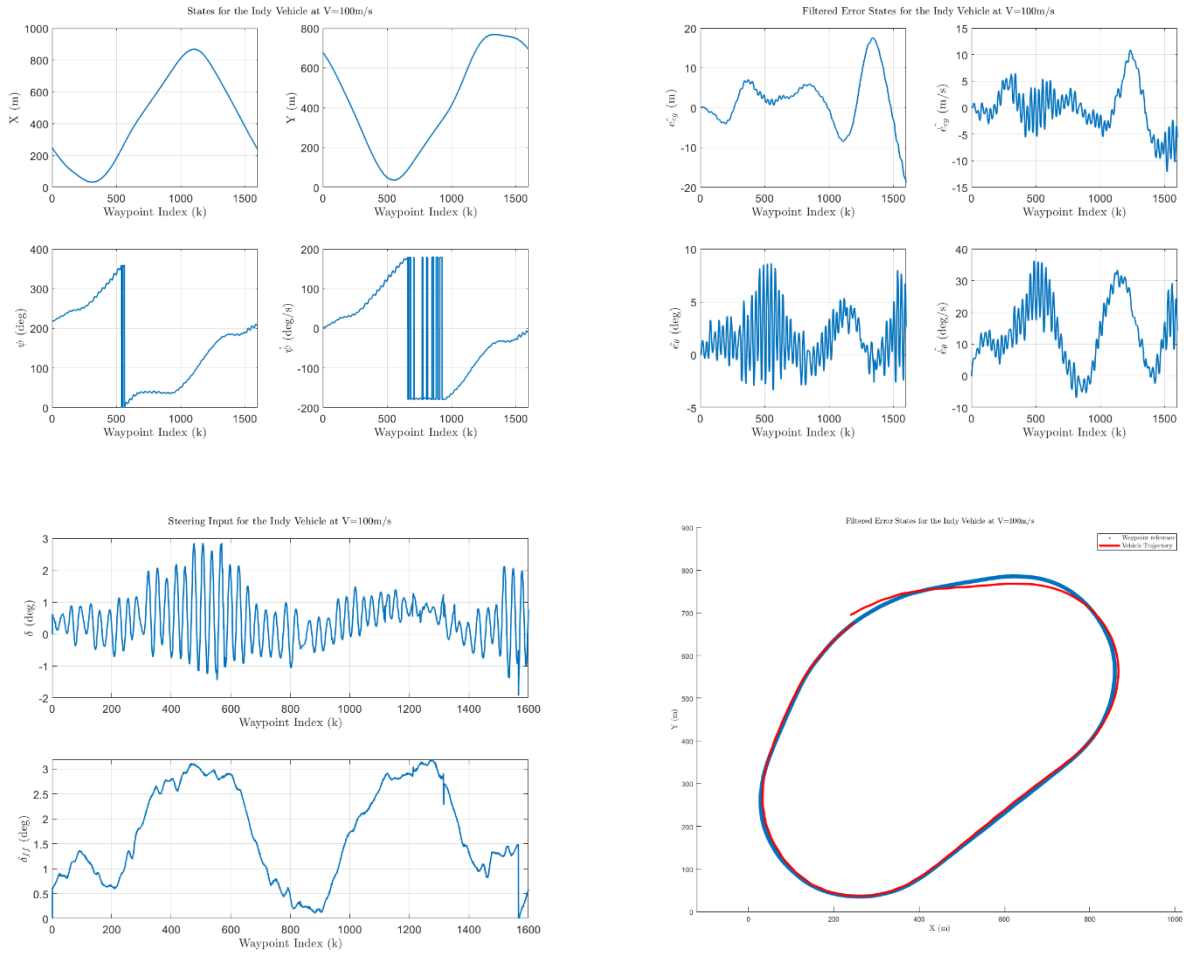

Figure 66: LVMS *with* δ_{ff} and V_x=100m/s

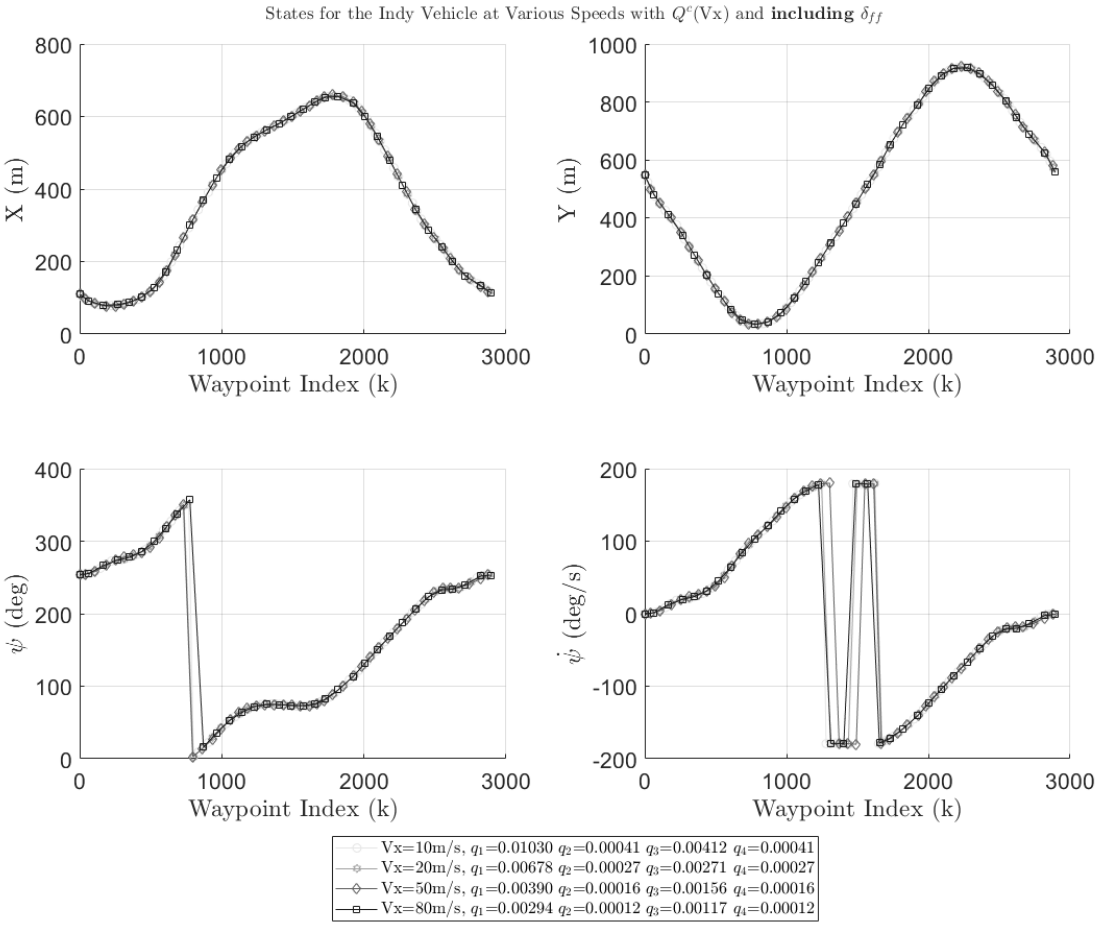

Figure 67: TMS Indy vehicle states with δ_{ff} and V_x= [10:80]m/s

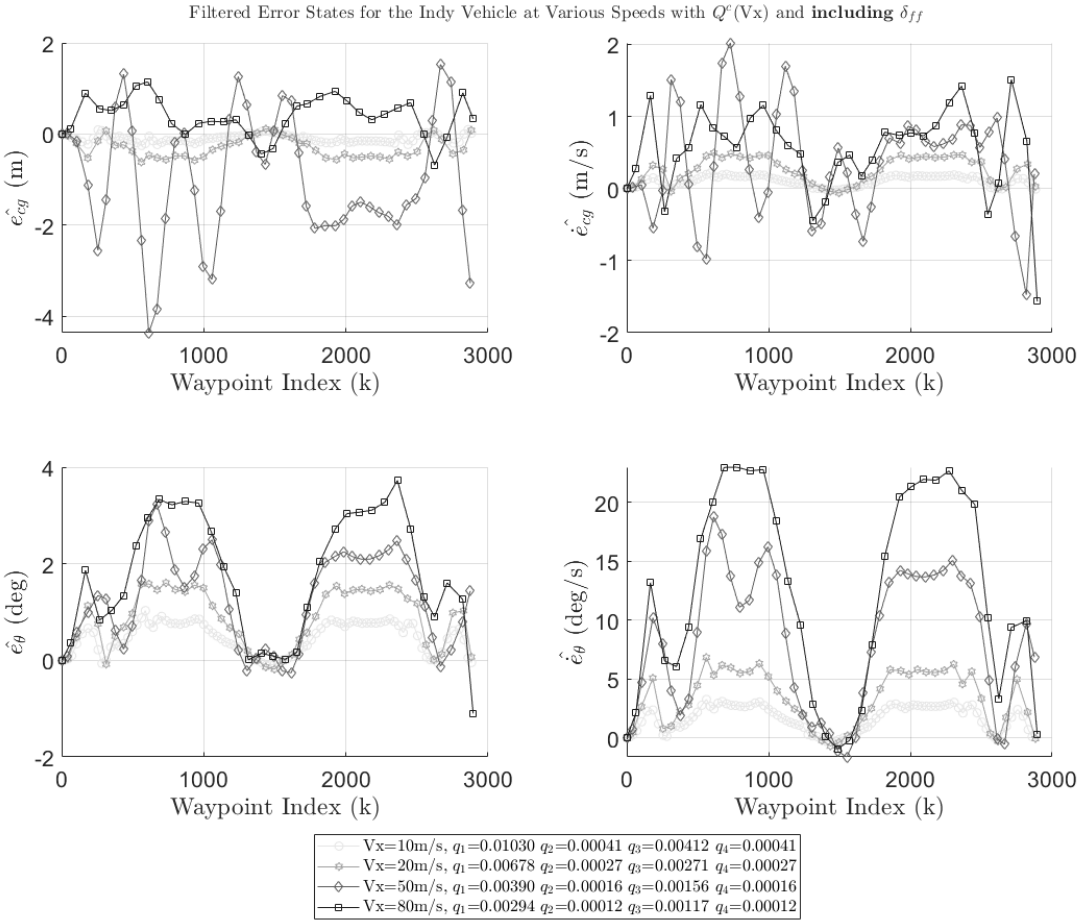

Figure 68: TMS Indy vehicle error states with δ_{ff} and V_x= [10:80]m/s

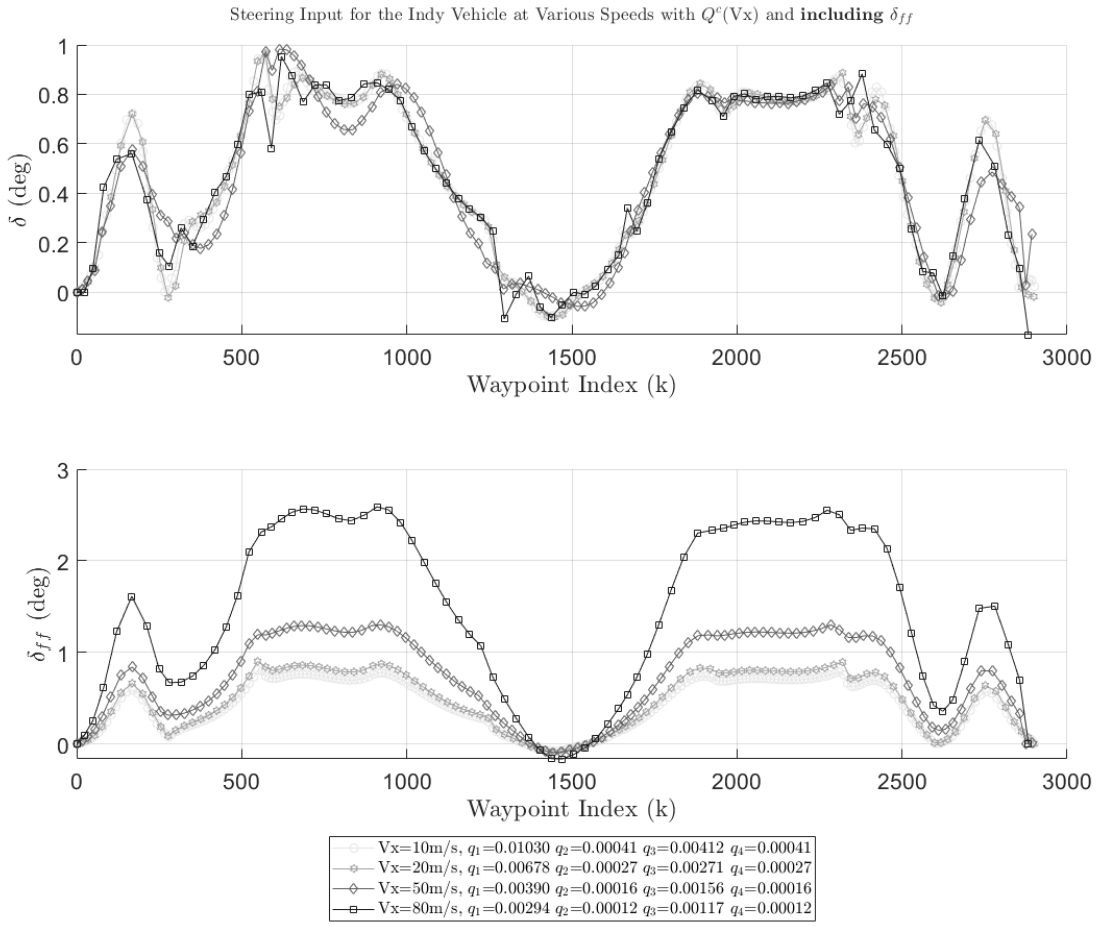

Figure 69: TMS Indy vehicle optimal steering inputs with δ_{ff} and V_x= [10:80]m/s

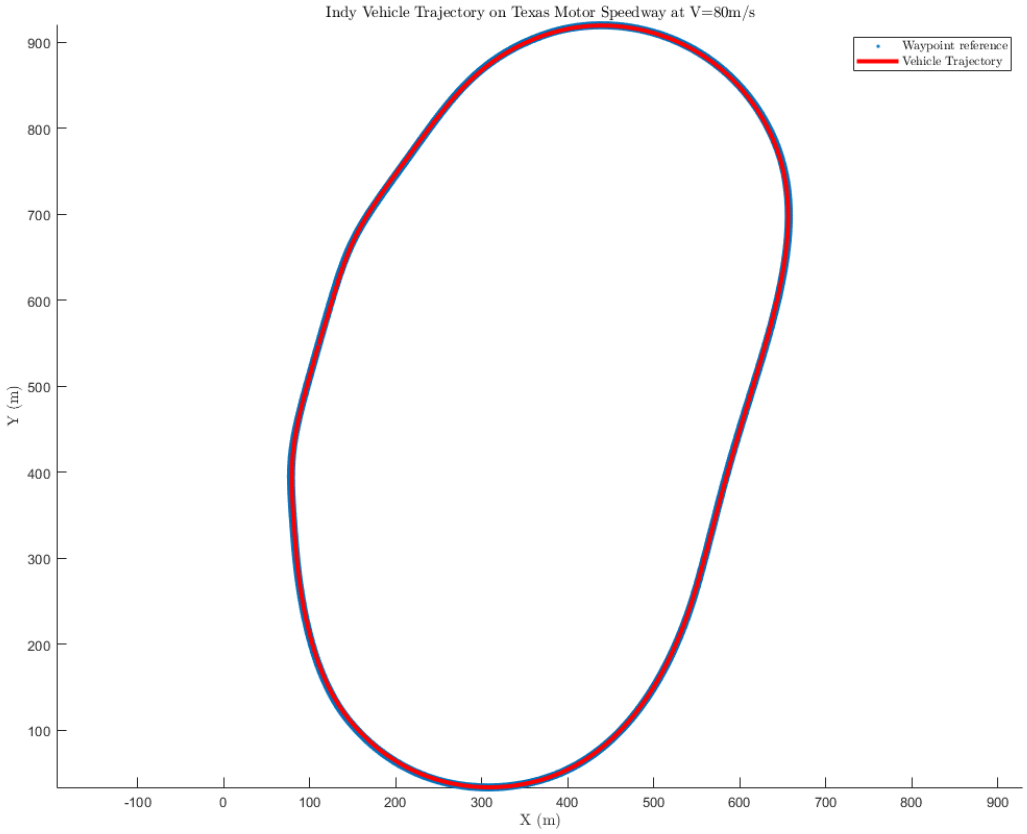

Figure 70: TMS Indy vehicle trajectory at V_x= 80m/s

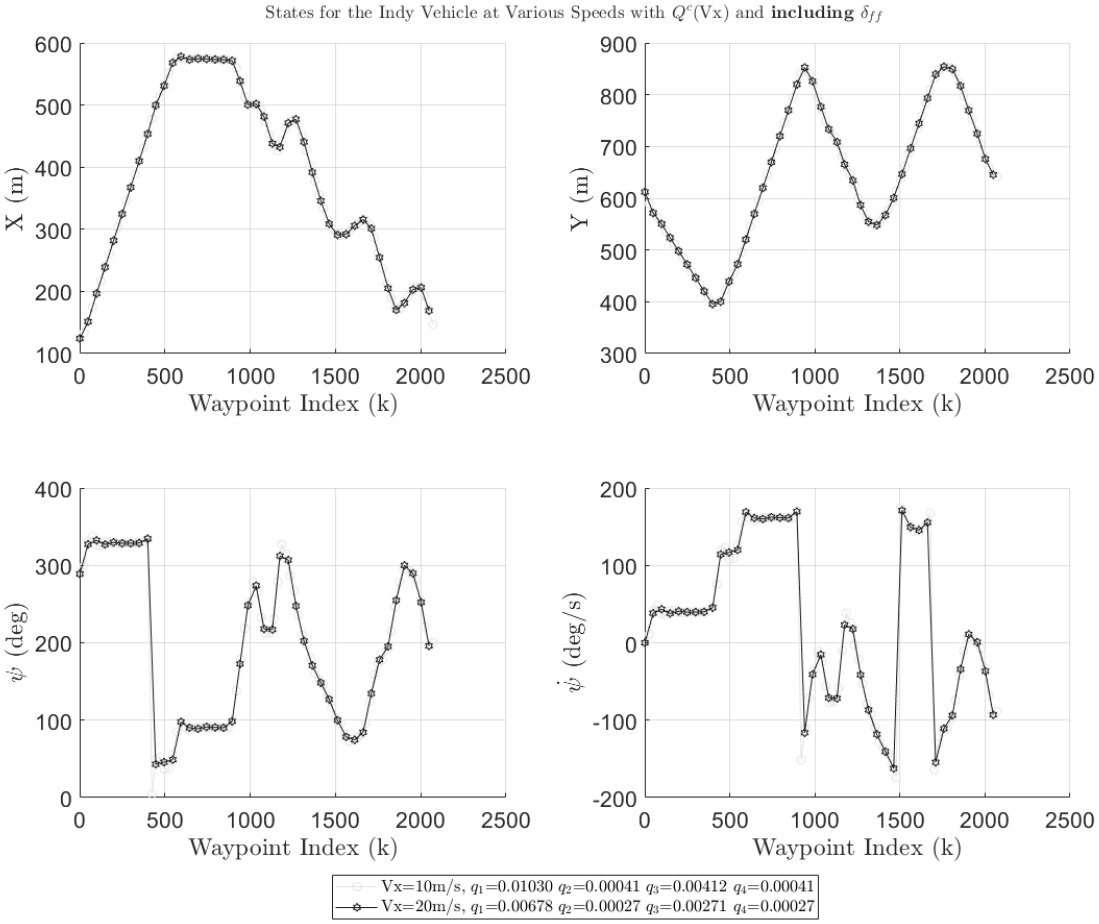

Figure 71: Purdue Indy vehicle states with δ_{ff} and V_x= [10:20]m/s

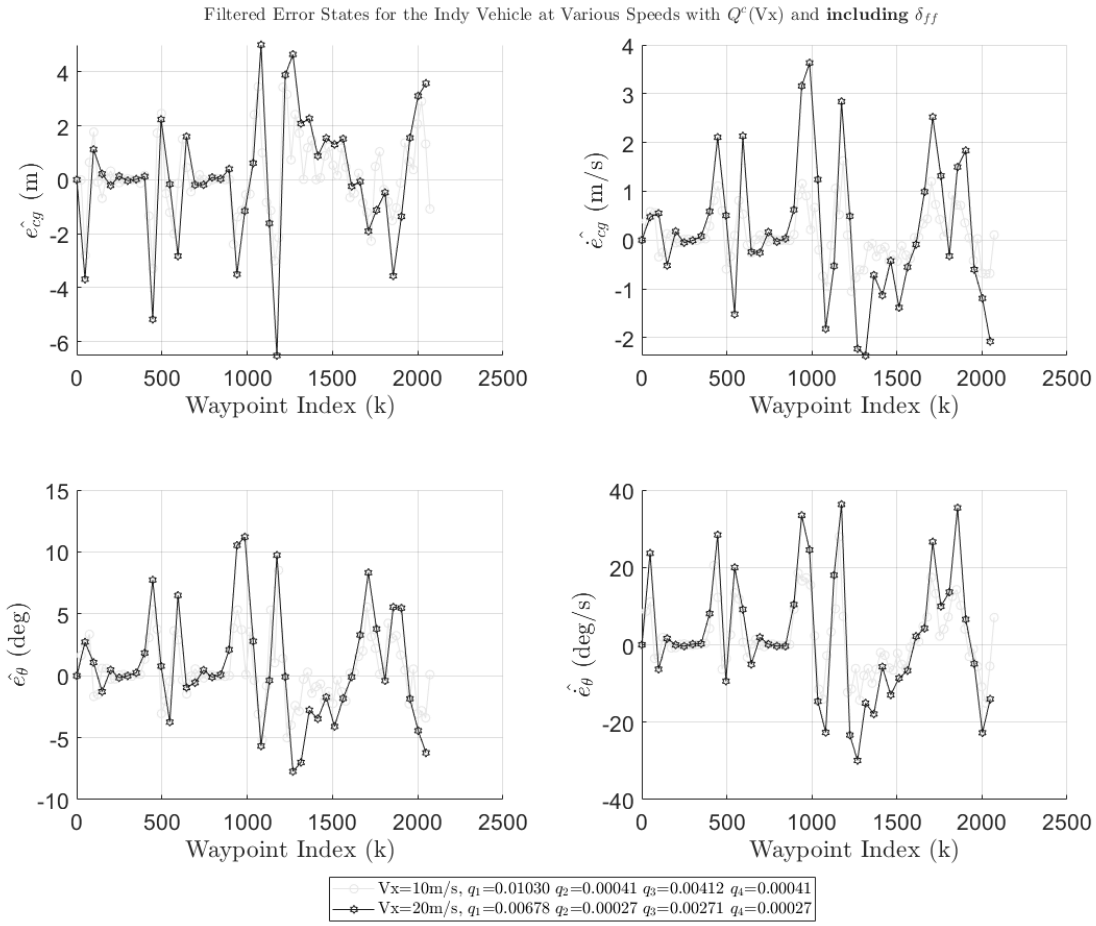

Figure 72: Purdue Indy vehicle error states with δ_{ff} and V_x= [10:20]m/s

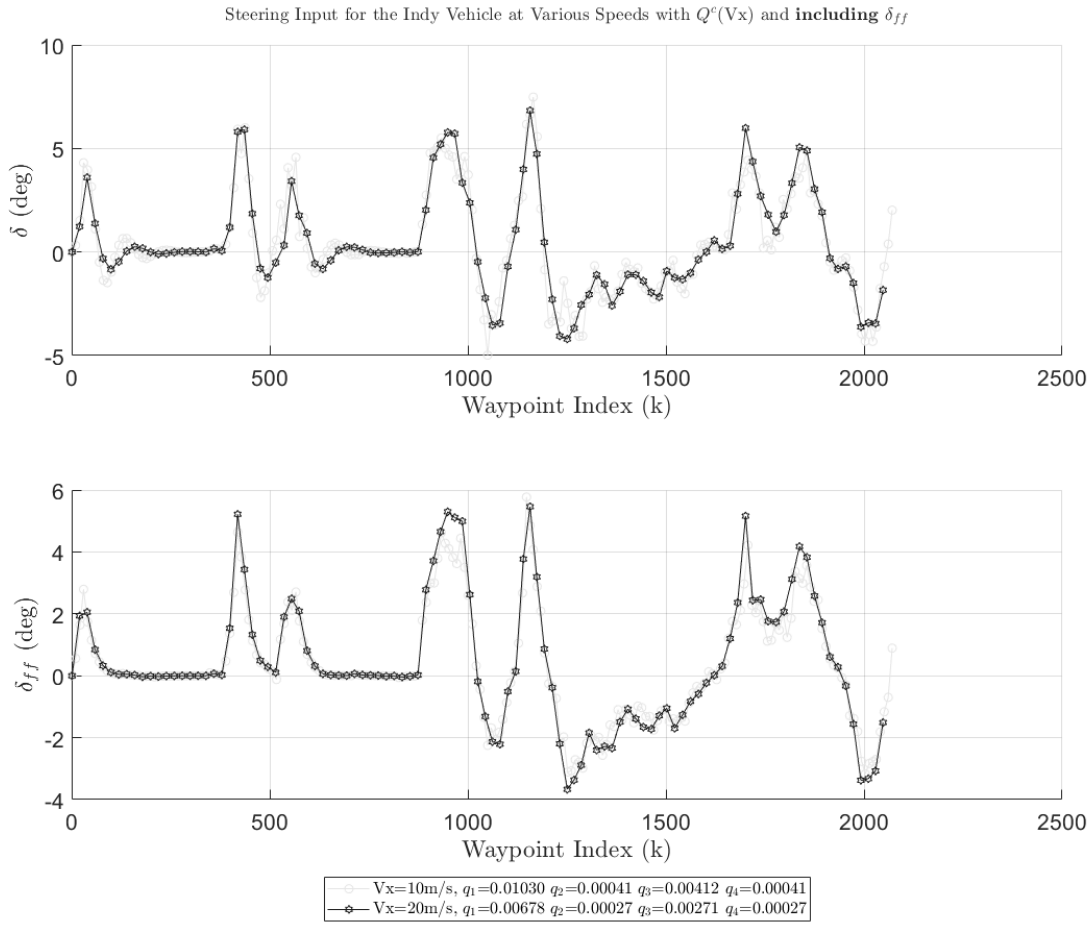

Figure 73: Indy vehicle optimal steering inputs with δ_{ff} and V_x= [10:20]m/s

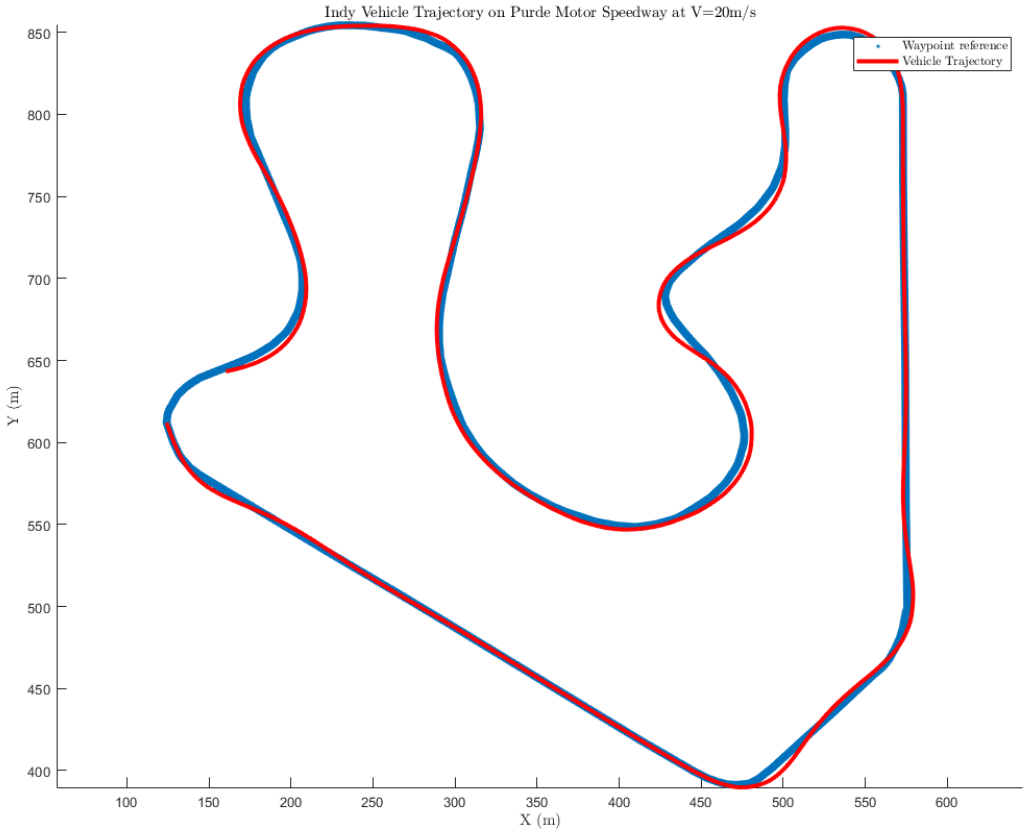

Figure 74: Purdue Indy vehicle trajectory at V_x= 20m/s

References

1. Rajesh Rajamani. Vehicle Dynamics and Control. Spinger, 2006.

2. Joseph D. Rounsaville. Methods for Calculating Relative Methods for Calculating Relative Cross-Track Err ack Error for ASABE/ISO or for ASABE/ISO Standard 12188-2 from Discrete Measurements, University of Kentucky, Joseph S. Dvorak Timothy S. Stombaugh, University of Kentucky 2016

3. Adarsh Patnaik, Manthan Patel, Vibhakar Mohta, Het Shah, Shubh Agrawal, Aditya Rathore. Design and Implementation of Path Trackers for Ackermann Drive based Vehicles. 2020

4. Peng, H., Tomizuka, M. Preview control for vehicle lateral guidance in highway automation. Journal of Dynamic Systems Measurement & Control-Transactions of the Asme, Vol. 115, No. 4, pp. 679-686, Dec 1993

5. Jihan Ryu. State and Parameter Estimation for Vehicle Dynamics Control using GPS. PhD thesis, Stanford University, 2004

6. Jihan Ryu. Automatic Steering Methods for Autonomous Automobile Path Tracking, PhD thesis, Carnegie Mellon University 2009

7. Paul Yih. Steer-by-Wire: Implications for Vehicle Handling and Safety. PhD thesis, Stanford University, 2005

8. M. Doumiati, A. Victorino, A. Charara, D. Lechner, A method to estimate the lateral tire force and the sideslip angle of a vehicle: Experimental validation, in: American Control Conference (ACC), 2010, IEEE, 2010, pp. 6936–6942.

9. V. Balakrishnan1 and L.Vandenberghe. Connections Between Duality in Control Theory and Convex Optimization. American Control Conference, 21 June 1995.

10. Fredrik Gustafsson. Sensor fusion for accurate computation of yaw rate and absolute velocity. Linkoping University, Sweden 2001

11. D. M. Bevly, R. Sheridan, and J. C. Gerdes. Integrating ins sensors with GPS velocity measurements for continuous estimation of vehicle sideslip and tire cornering stiffness. In Proceedings of the 2001 American Control Conference, pages 25–30, Arlington, VA, 2001.

12. A. Gelb Joseph F. Kasper, Raymond A. Nash, Charles F. Price, Arthur A. Sutherland, Applied Optimal Estimation, The M.I.T. Press, Cambridge, Massachusetts, 1974

13. F.L. Lewis, D. Vrabie, and V.L. Syrmos, Optimal Control, 3rd edition, John Wiley 2013

14. F.L. Lewis, Applied Optimal Control and Estimation: Digital Design and Implementation, Prentice-Hall, New Jersey, TI Series, Feb. 1992.