**Development of an Automatic Steering Function on a Vehicle Driving Simulator**

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Junmin Wang, Supervisor

Dr. Raul G. Longoria, Supervisor

# Development of an Automatic Steering Function on a Vehicle Driving Simulator

**by**

**Vibhav Gaur, B.S.**

**THESIS**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**MASTER OF SCIENCE**

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2020

To my parents

# Acknowledgments

I would like to express my deepest gratitude to my parents, whose support has been instrumental in my education - thank you, Mom and Dad.

I would like to thank my supervisors Dr. Junmin Wang for his advice and attention at every stage of the project, and Dr. Raul G. Longoria for his guidance and feedback for this work.

Finally, thanks are in order to Zejiang Wang and Adrian Cosio, without whose help this work would not be possible.

# Development of an Automatic Steering Function on a Vehicle Driving Simulator

Vibhav Gaur, M.S.
The University of Texas at Austin, 2020

Supervisors: Dr. Junmin Wang
Dr. Raul G. Longoria

The subject of this thesis is the development of an Automatic Steering function on a vehicle driving simulator. A model-free controller based on algebraic derivative estimation is developed to drive the steering wheel motor and therefore implement autonomous driving functionality in the driving simulator. Due to the pandemic of 2020, limited lab access resulted in the inability to tune the implemented controller on the actual system. Presented in this work is a proof-of-concept of the feasibility of such a system, with the final step being the tuning of the implemented controller.

**Keywords:** Driving simulator, Model-free control, Algebraic Derivative Estimation, Automotive Simulation Models (ASM)

# Table of Contents

**Acknowledgments** **v**

**Abstract** **vi**

**List of Tables** **ix**

**List of Figures** **x**

**Chapter 1. Introduction** **1**

1.1 Objectives . . . . . 1

1.1.1 Impact of the 2020 pandemic . . . . . 1

1.2 Motivation and justification . . . . . 2

**Chapter 2. System description** **5**

2.1 Driving simulator hardware description . . . . . 5

2.1.1 Driving simulator components . . . . . 5

2.1.2 Integration with the software . . . . . 7

2.1.3 Control loader description . . . . . 9

2.2 Driving simulator software description . . . . . 9

2.2.1 Software hierarchy . . . . . 10

2.2.2 Automotive Simulation Models (ASM) . . . . . 11

2.2.3 The ASM simulation loop . . . . . 14

2.2.4 Communication between the hardware and software . . 16

2.2.5 Driver model in ASM . . . . . 18

2.2.6 Steering modes in Automotive Simulation Models (ASM) 19

**Chapter 3. Model-free control** **22**
3.1 Ultra-local model . . . . . 22
3.2 Control law and closed-loop dynamics . . . . . 24
3.3 Real-time estimation of the model parameter $\phi$ . . . . . 25
3.3.1 Estimator of order 2 . . . . . 26
3.3.2 Estimator of order 3 . . . . . 28
3.4 Discrete real-time implementation of algebraic derivative estimator . . . . . 28

**Chapter 4. Controller Development** **30**
4.1 Test of feasibility . . . . . 30
4.2 Motor model . . . . . 31
4.3 Control design . . . . . 32
4.4 Simulation results . . . . . 33
4.4.1 Effect of estimation parameters . . . . . 35
4.4.2 Effect of controller parameter $\alpha$ . . . . . 38

**Chapter 5. Conclusions and Future Work** **40**

**Appendices** **41**

**Appendix A. Algebraic Derivative Estimation** **42**

**Appendix B. Simulink block diagrams** **45**

**Bibliography** **47**

# List of Tables

2.1 Boolean flags used for selecting steering input mode . . . . . 21

4.1 Values of motor parameters . . . . . 31

# List of Figures


1.1 The system functionality needed to study human driver interaction with the autonomous system. . . . . 3
1.2 Picture showing the mechanical position limiter on the steering column (orange spacer) . . . . . 4

2.1 The Motion base . . . . . 6
2.2 Layout of the motion base and the screen . . . . . 7
2.3 Network structure of the simulator system (MB stands for Motion Base)[8]. . . . . 8
2.4 The steering feedback Control Loading (CL) motor: eF-DDM-20 by E2M Technologies [12] . . . . . 9
2.5 Hierarchy of the software used to run the simulator [14] . . . . 11
2.6 Top page of the ASM Vehicle Dynamics Simulink model. . . . 12
2.7 Schematic of the vehicle model as implemented in the `MDL` block in Simulink [2] . . . . . 13
2.8 Diagram showing the signal flow with the human in the loop . 15
2.9 Signal flow from the hardware to the software [8] . . . . . 16
2.10 Screenshot showing the default self-aligning torque signal being sent into the `To E2M CL` block. . . . . 17
2.11 Screenshot showing the default self-aligning torque signal being sent into the `From E2M CL` block. . . . . 18

3.1 Visual intuition for the ultra-local model. . . . . 24

4.1 Control loop for the steering wheel position control problem. . 32
4.2 Comparison of MFC vs. PID when the reference is a noisy signal. 33
4.3 Algebraic derivative estimation for the case shown above. . . . 34
4.4 Comparison of a derivative estimator of order 2 vs. a derivative estimator of order 3. . . . . 36
4.5 Comparing the effect of estimation window size on the quality of the algebraic estimation. . . . . 37

4.6 Comparing the effect of changing $\alpha$ on the system response and controller performance. . . . . 38

B.1 Simulink implementation of the model-free controller algorithm. 45

B.2 Simulink block diagram of the model-free controller simulation. 46

# Chapter 1

# Introduction

## 1.1 Objectives

The objective of this thesis is to implement automatic steering on a Cruden vehicle driving simulator using Automotive Simulation Models (ASM), a vehicle model created by dSPACE. The control methodology used for this purpose is model-free control, a novel approach developed in [5]. The purpose for the creation of this functionality in the driving simulator is to study the interaction of human drivers with autonomous driving systems.

### 1.1.1 Impact of the 2020 pandemic

The COVID-19 pandemic of 2020 caused research lab closures across the United States. Due to this reason, this work fell short of its intended objective of implementing automatic steering on the driving simulator. While the control algorithm was implemented on the system, the lab was shut down before the controller could be tuned for the desired performance on the driving simulator. As a result, the data shown in section 4.4 are from the same control algorithm but applied to a simulated motor model. This work, in its current form, serves as a proof-of-concept for the intended objective, with the tuning of the system as the last remaining step.

## 1.2 Motivation and justification

The Cruden driving simulator is an invaluable tool in studying human interaction with an automobile due to its ability to load different types of vehicles, traffic scenarios, roads, and environments without the need for expensive real world testing. A simulator test also offers a higher safety standard by virtue of it being virtual. With autonomous vehicles steadily gaining in popularity, it is important to scientifically study the interaction of human drivers with autonomous vehicle systems. Specifically, the points when control of the vehicle is switched from the human driver to the autonomous driving controller. Therefore, the need for an automatic steering system on the simulator arises.

In its default implementation, the Cruden driving simulator is capable of giving steering and motion feedback to the human driver in normal driving scenarios. As such, there is no support to simulate autonomous driving situations in the driving simulator. This necessitates the need to implement a custom automatic steering feature in the simulator to study the handover of control by the human to the computer and vice versa. For instance, some useful tests that this will enable include human driver reaction times when disengaging the automatic steering, human driver biometric scanning during different traffic scenarios, and evaluating the performance of various steering control strategies for the edge case of switching from human vehicle control. To summarize the requirements: a system feature needs to be implemented that allows the human driver to push a button to switch the vehicle from hu-

man driver steering to automatic steering and vice versa. This is illustrated in figure 1.1.

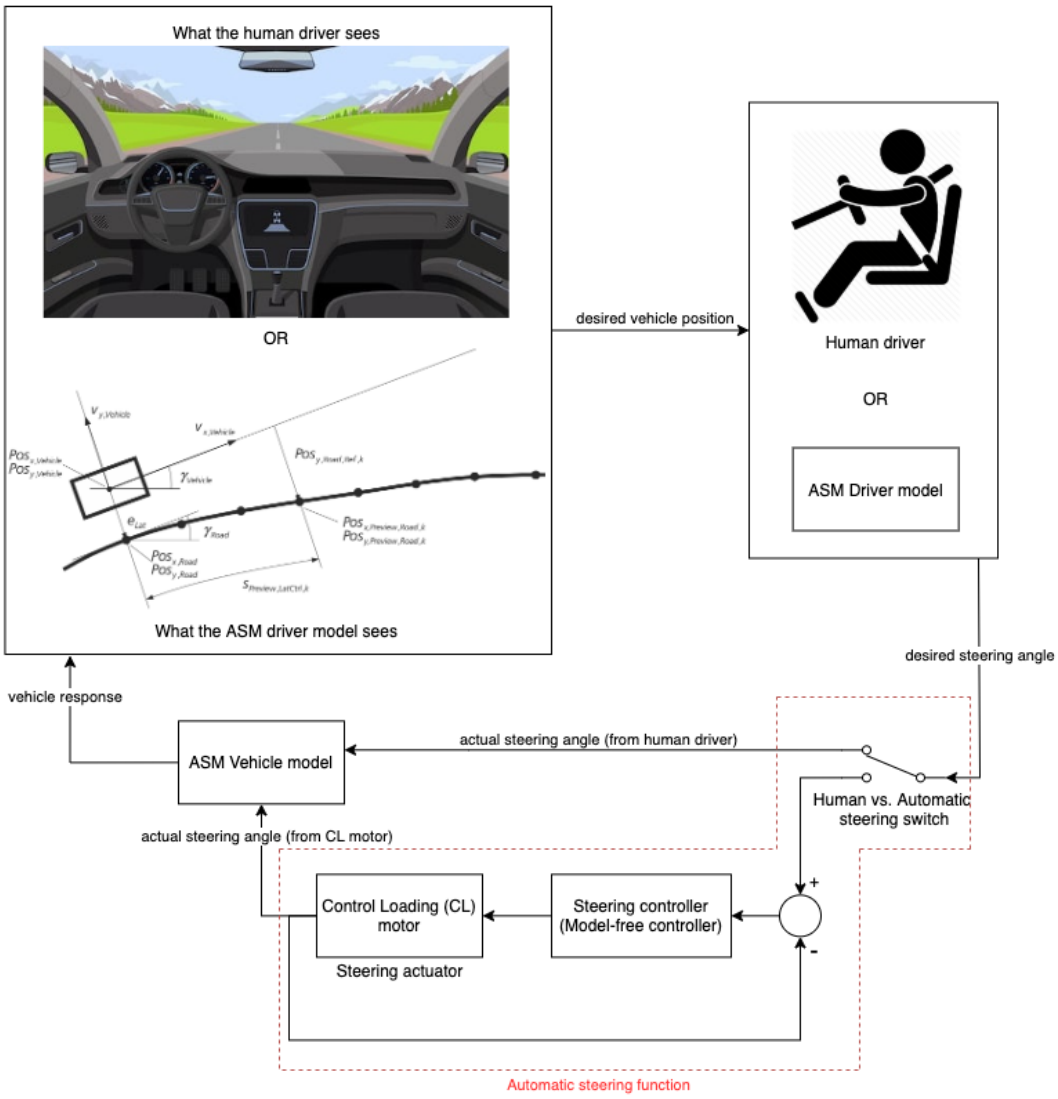



Figure 1.1: The system functionality needed to study human driver interaction with the autonomous system.

The model-free control method was selected for this project as system identification proved to be difficult on the steering wheel of this driving simulator due to mechanical limits on the position of the steering wheel. Additionally, the unpredictable friction profile of the steering wheel and motor assembly meant that frequency domain identification would yield poor results. Additionally, the novelty of the model-free control method also was a factor in this decision.

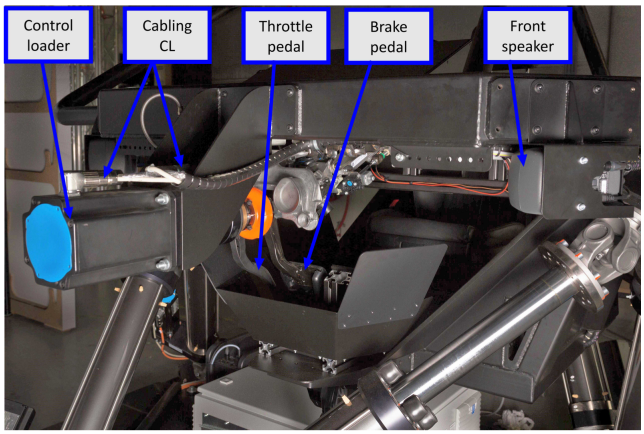



Figure 1.2: Picture showing the mechanical position limiter on the steering column (orange spacer)

# Chapter 2

# System description

## 2.1 Driving simulator hardware description

The Cruden driving simulator system hardware is composed of a motion base, a cylindrical projection screen, three projectors, ten PCs, 4 speakers, and equipment for their connection. The following section is aimed at making the reader reasonably familiar with the hardware of the simulator while not diving too deep in the specifics of each component. Each relevant component is described in a way that is pertinent to the project and therefore, should the reader want more detail about anything discussed in the following section, they should refer to the manufacturer's documentation.

### 2.1.1 Driving simulator components

**Motion base**

The Motion base is a 6 degree-of-freedom hexapod platform on which the simulator cab is mounted. It has 6 electric actuators that provide movement in 3-dimensions to mimic the motion of the vehicle being simulated. It also houses the steering wheel, steering wheel feedback actuator (the Control Loading motor), and the pedal box assembly.

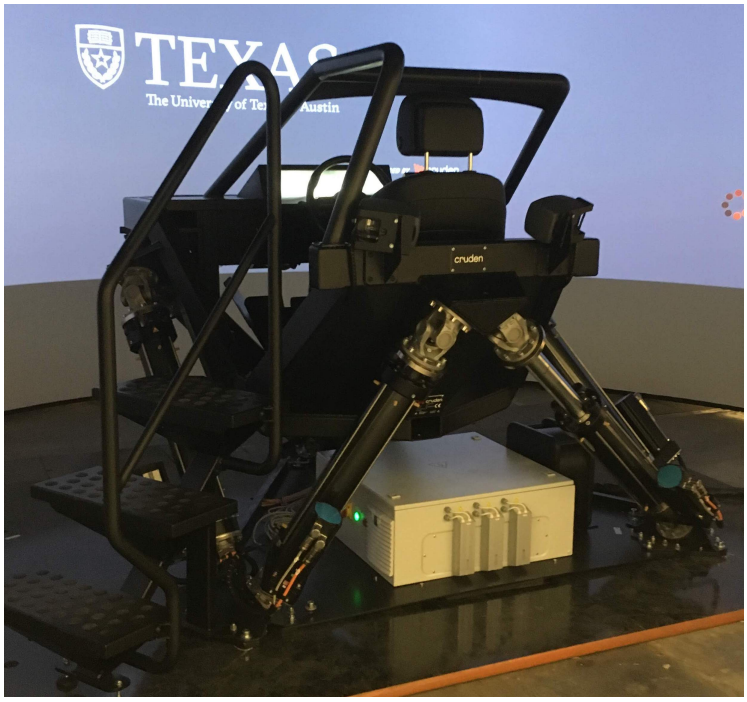



Figure 2.1: The Motion base

### Display system

The display system consists of three projectors with their positioning calibrated in order to produce one contiguous image on the 210° field-of-view cylindrical screen. In addition to the projection, the simulator cab is equipped with several screens that serve as the instrument cluster, the driver's and outer rearview mirrors, and a programmable touchscreen.

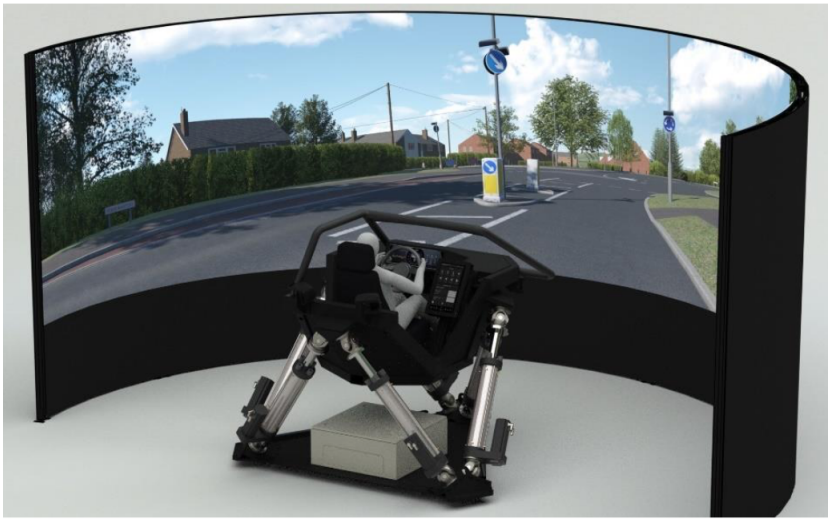


Figure 2.2: Layout of the motion base and the screen

### 2.1.2 Integration with the software

The simulator hardware is integrated with the software described in section 2.2 by a total of 10 networked computers. These computers serve specific functions with regards to the operation of the simulator. The computer network is shown in figure 2.3 and the roles of the computers are as follows [8]:

- Operator (OP) - starts the simulation software and the master computer for the simulator.
- Master (M) - starts the other computers on the network.

- Image Generators (IG) - computers that generate the graphics to be sent to the mirror screens or the projectors.
- Spectator (S) - allows spectators to see the reaction of the car and the driver (not used in this project).
- Physics (EPN) - run the Simulink vehicle model described in section 2.2.2, which handles the physics of the simulation.
- SCALEXIO - the real-time computer on which the generated code is built. This is the computer used to run a simulation with Hardware-in-the-Loop (HiL) (not shown in figure 2.3).

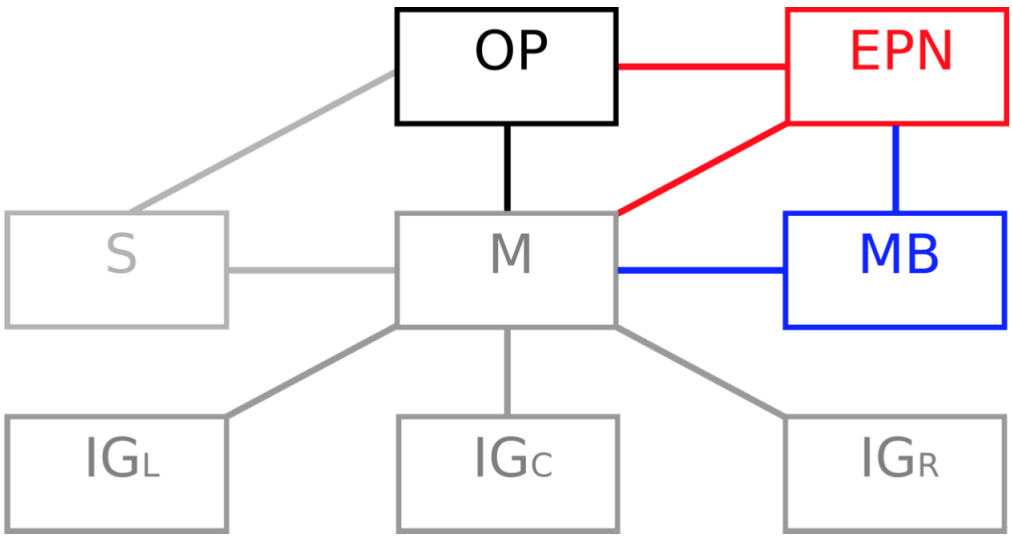



Figure 2.3: Network structure of the simulator system (MB stands for Motion Base)[8].

### 2.1.3 Control loader description

The force feedback on the steering wheel is achieved by means of the eF-DDM-20 CL motor manufactured by E2M Technologies in The Netherlands. This direct-drive, multi-turn actuator offers high-fidelity in reproducing smooth steering feedback torque and finds application in many different kinds of simulators. This serves as the actuator for the control problem posed for this project.

Figure 2.4: The steering feedback CL motor: eF-DDM-20 by E2M Technologies [12]

## 2.2 Driving simulator software description

The hardware described in section 2.1 uses Automotive Simulation Models (ASM) as the software to simulate the vehicle in real-time. This section provides a useful overview about the software and its organization in order

to save the reader the time and effort required to go through the significant amount of documentation available for this software. This is by no means a comprehensive guide on how to use ASM. Instead, this is a distillation of some useful information that pertains to this project from the ASM documentation such as [2], [3], and [4]. The reader should approach this section as a primer on the software so as to better understand the methodology described in later chapters.

### 2.2.1 Software hierarchy

Figure 2.5 shows the overview of the software layers that run the simulation. The base layer is made in Simulink that contains the vehicle model and the control algorithms - this is the layer where custom controllers are implemented. This model is parameterized by the dSPACE ModelDesk software, which provides a graphical user-interface to load vehicles with different properties (e.g. sedan, truck, etc.). Once the model is fully parameterized, ConfigurationDesk is used to generate code that can be sent to the SCALEXIO computer to run the simulation in real-time. Finally, ModelControl and Panthera - software written by Cruden - are used to interface between the dSPACE real-time computer and the Cruden hardware.

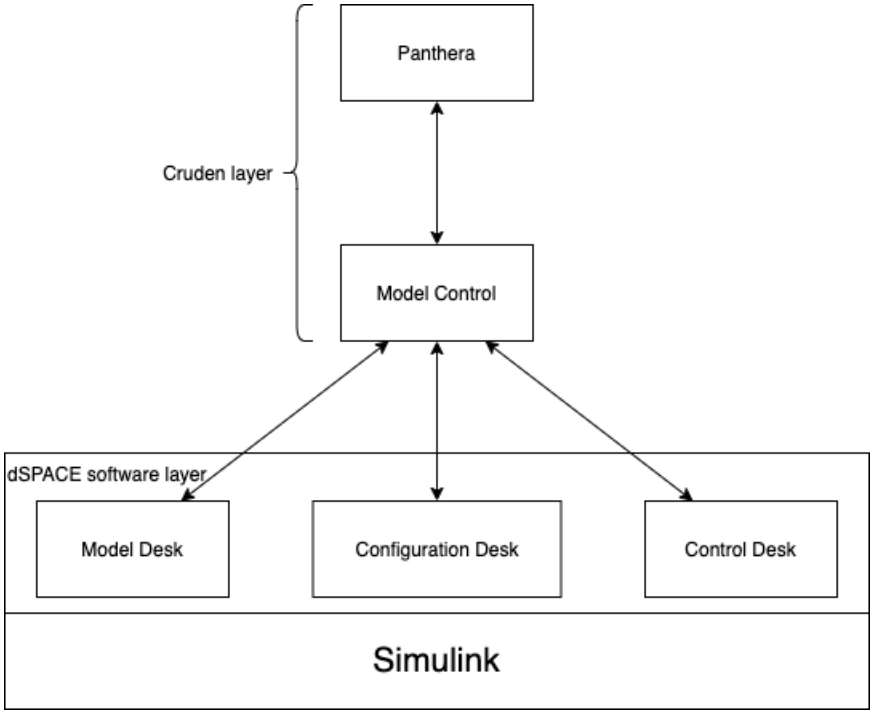



Figure 2.5: Hierarchy of the software used to run the simulator [14]

### 2.2.2 Automotive Simulation Models (ASM)

ASM is a high-fidelity virtual vehicle software developed by dSPACE. The model uses Simulink as the platform of implementation and uses "open" blocks, which means that the model implementation is visible to users down to the level of built-in Simulink blocks. This allows the user to modify and customize the vehicle model to develop their own systems and then test them on a fully simulated vehicle in real-time. In addition to the vehicle models, ASM also provides software to create roads and environments, traffic scenarios, and driving maneuvers, all of which can be used to create more realistic testing within the simulation. The models used for the purposes of this thesis are the

ASM Vehicle Dynamics with Traffic.

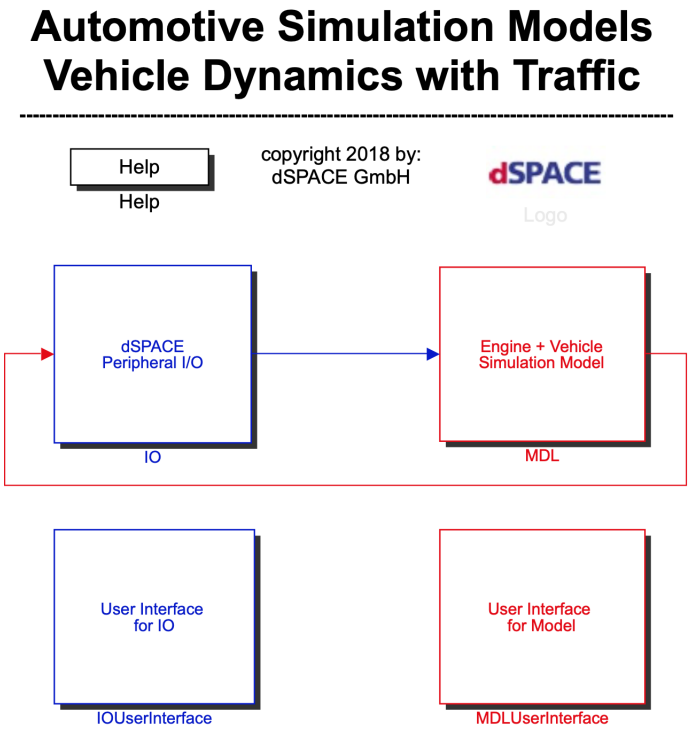



Figure 2.6: Top page of the ASM Vehicle Dynamics Simulink model.

ASM Vehicle Dynamics is a nonlinear multibody vehicle system with support for vertical, longitudinal, and lateral vehicle dynamics. The model is divided into five main subsystems as shown in figure 2.7 where each is linked with the others and provides inputs and outputs for the other subsystems:

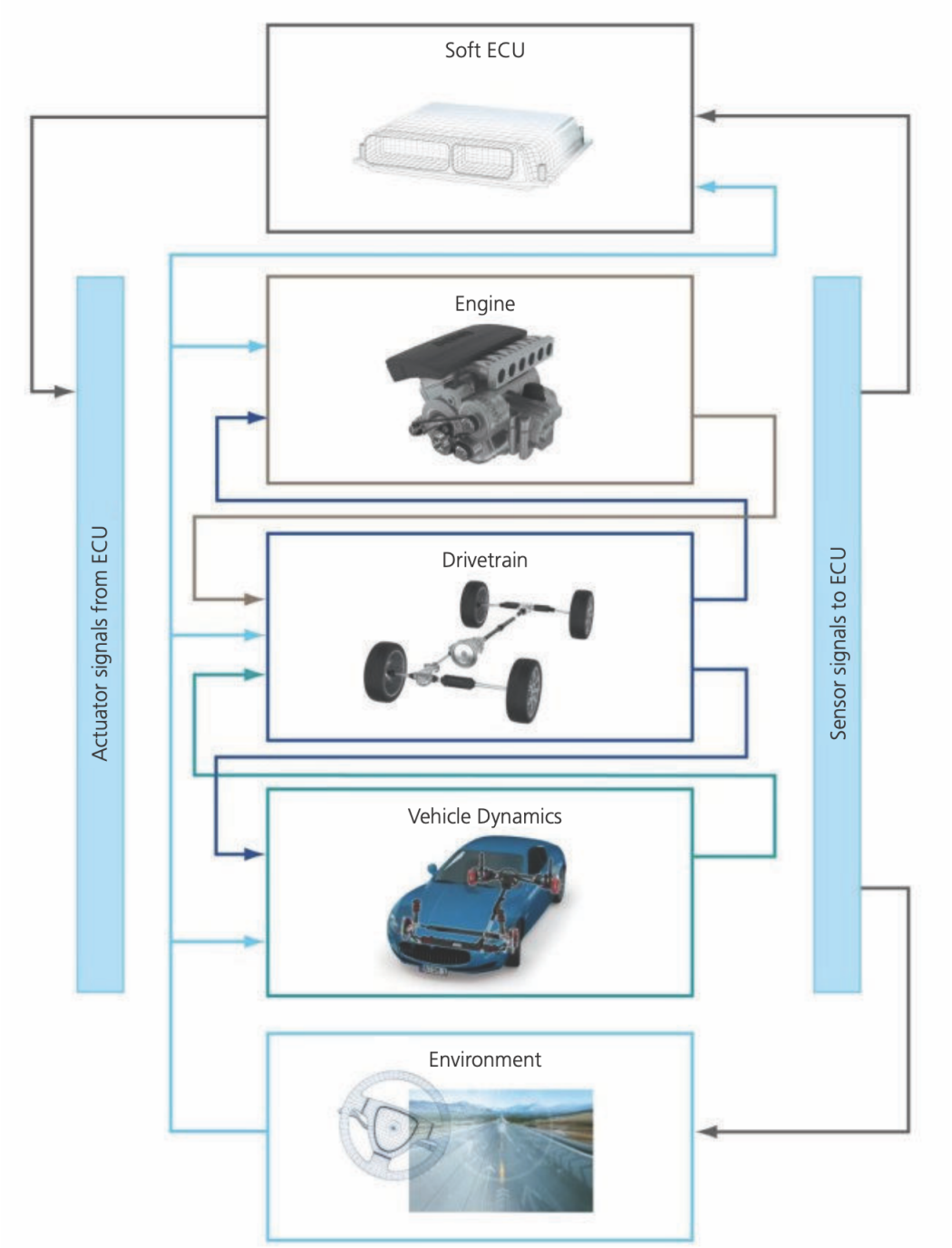



Figure 2.7: Schematic of the vehicle model as implemented in the `MDL` block in Simulink [2]

The subsystems of interest for this project are the Vehicle dynamics and Environment subsystems. The Vehicle dynamics susbsystem contains the blocks that determine the steering dynamics while the Environment subsystem contains the driver model in ASM (explained in section 2.2.5).

### 2.2.3 The ASM simulation loop

The ASM model has access to many signals available from the Cruden driving simulator hardware. For instance, the CL motor (used to provide steering wheel feedback) provides the vehicle model with signals such as steering wheel position, steering wheel velocity, and steering wheel force (or torque). These are signals from the physical world that are generated by the person in the driver's seat by interacting with the steering wheel. The values of these signals are sent from the `IO` block to the `MDL` block for each iteration of the simulation loop. These signals are sent to various subsystems to act as inputs into the vehicle model. The model then computes the response of each vehicle subsystem to the aforementioned inputs, resulting in the total vehicle response. This response is used by the middleware from Cruden to generate the graphics images and animations, in addition to rendering the environment, which are then projected on to the screens. In addition, the `MDL` block generates the signals needed to send to the driving simulator hardware, namely the motion base actuators and the CL motor, to simulate the vehicle motion and steering feedback respectively, completing the simulation loop with the human driver in it.

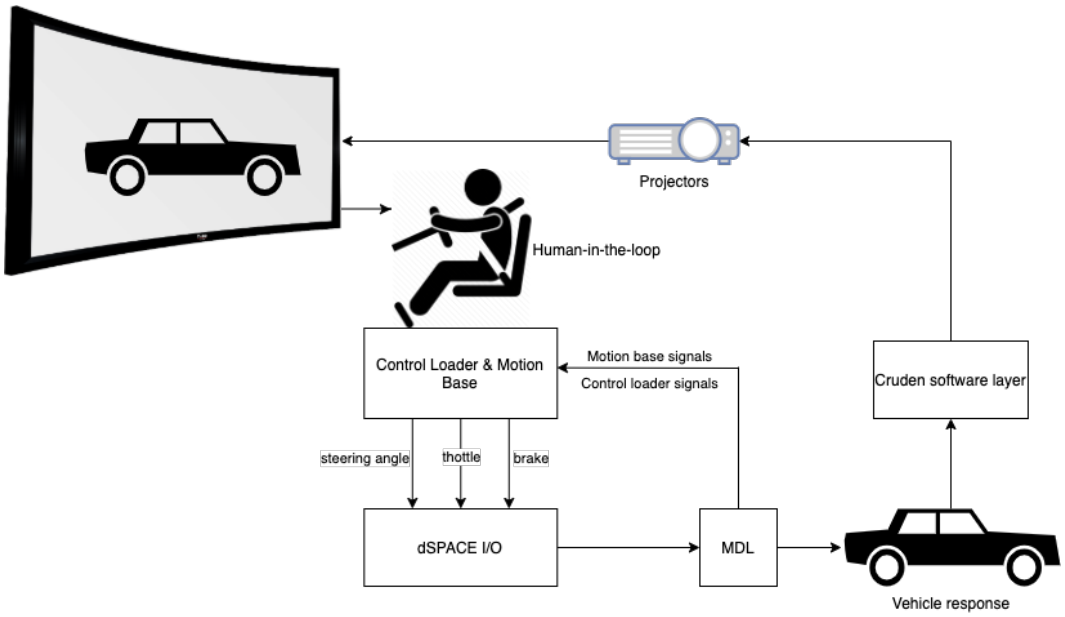



Figure 2.8: Diagram showing the signal flow with the human in the loop

As a part of the vehicle response, the model generates the steering feedback torque which occurs due to the self-aligning moment experienced by the tires and the force experienced at the wheels due to the road. In a real vehicle, these forces are transmitted through the suspension and steering linkages back to the driver (with the exception of steer-by-wire systems, which artificially generate those forces at the steering wheel). In the Cruden driving simulator these forces are transmitted through the CL motor to provide a realistic feeling to the driver. The details of how these signals are sent to and from the model are discussed in the following section.

### 2.2.4 Communication between the hardware and software

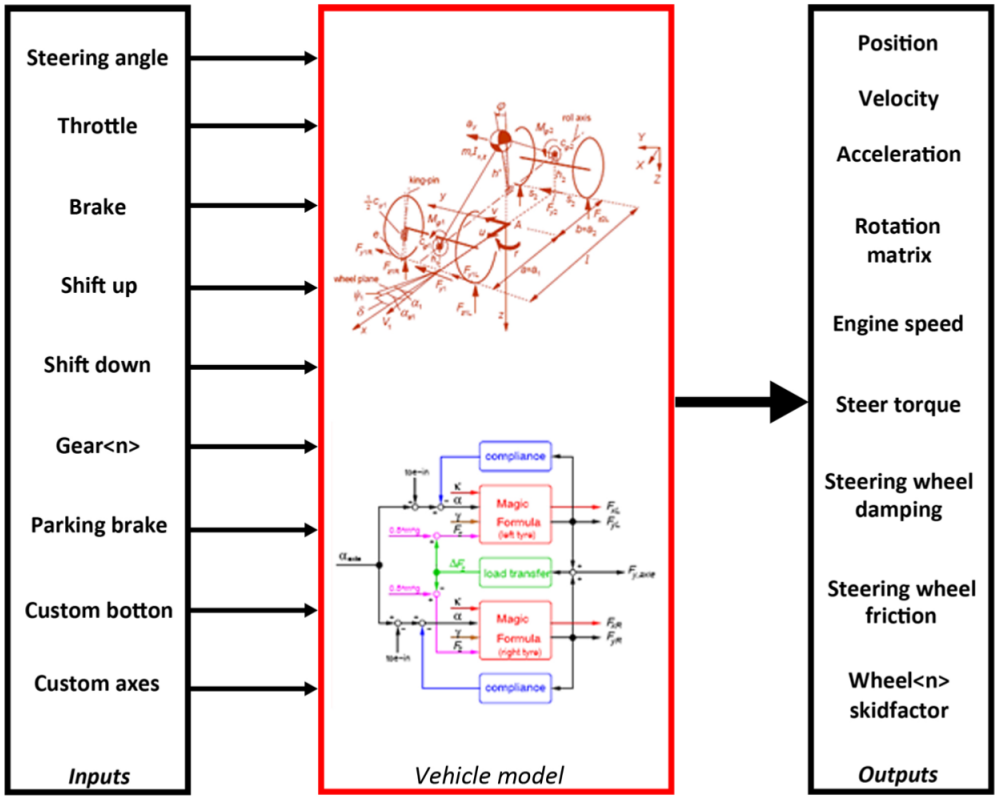



Figure 2.9: Signal flow from the hardware to the software [8]

Figure 2.9 gives a schematic for the signals available for the software from the hardware layer. As previously mentioned, the vehicle model has access to signals such as steering angle, throttle and brake position etc. from the hardware in the simulation loop. The block used to command the CL motor is the `To E2M CL` block inside the `IO` block of the Simulink model (figure

2.10). This block takes in signals like steering wheel friction and inertia, steering wheel position limits, and the torque demanded from the CL motor. The value of steering wheel torque to be fed back to the driver is calculated in the `STEERING_3DOF_VARIABLE_RATIO` block inside the `VehicleDynamics` subsystem in the `MDL` block. This opens up a possibility to exploit the CL hardware to perform automatic steering with the torque demand signal being generated by a custom controller.

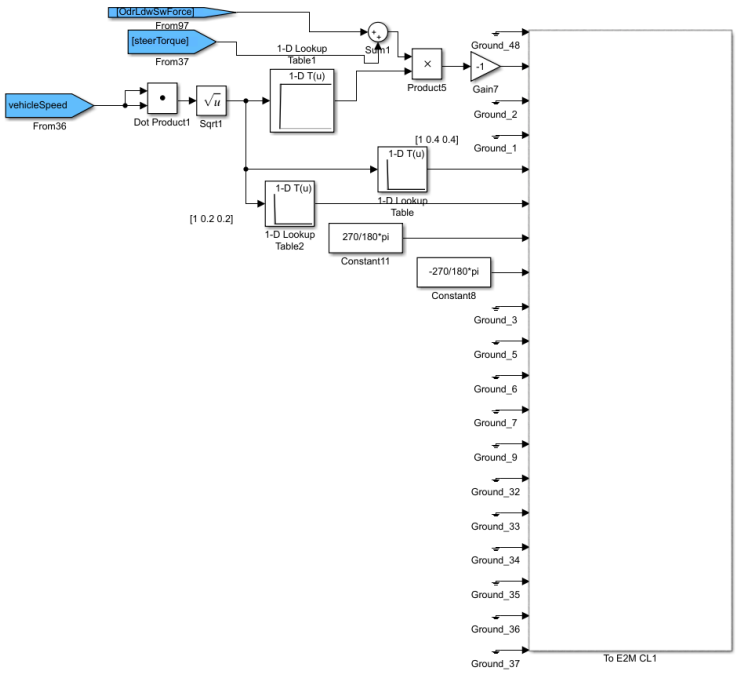



Figure 2.10: Screenshot showing the default self-aligning torque signal being sent into the `To E2M CL` block.

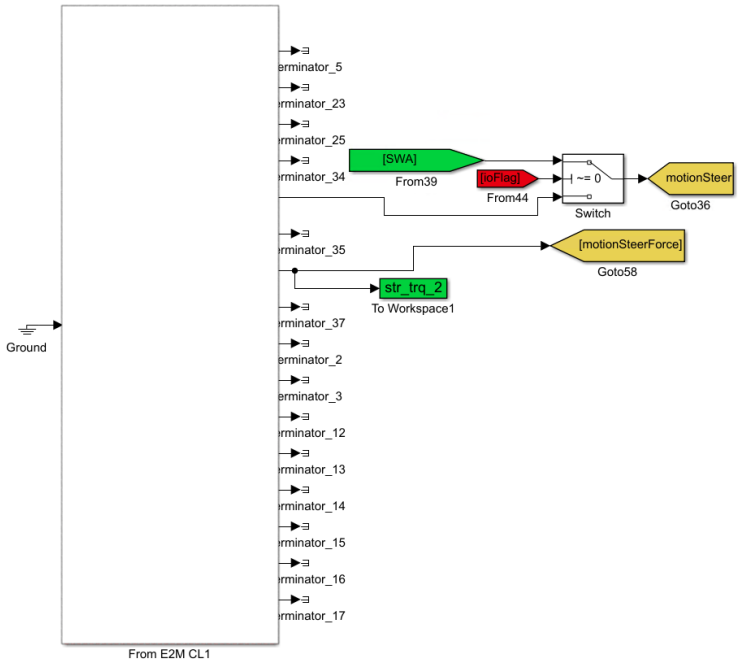



Figure 2.11: Screenshot showing the default self-aligning torque signal being sent into the `From E2M CL` block.

Conversely, the `From E2M CL` block inside `IO` provides the position, velocity, and force signals from the steering wheel which can be used as measurements for any custom controller (figure 2.11).

### 2.2.5 Driver model in ASM

ASM has an internal driver model implemented for simulation cases without any hardware where external driver inputs are unavailable. Depending on the steering mode selected (section 2.2.6), one can run a purely software simulation (no hardware-in-the-loop). In such cases, the internal ASM driver model produces signals such as steering angle, throttle position, and brake

position. The job of the driver is divided into two separate controllers - the longitudinal controller and the lateral controller. The longitudinal controller is responsible for generating the throttle and brake signals while the lateral controller generates the steering signal. There are two implementations of the lateral controller - `LATERAL_CONTROL1`, used for most normal driving conditions, and `LATERAL_CONTROL2`, used for stationary applications like driving in a circle at constant speed. Both lateral and longitudinal controllers use preview information about the road in the ASM model to generate their respective signals [4].

For the purpose of automatic steering control design on the Cruden driving simulator the output of the ASM driver model was used as the reference signal. This reference signal can be used to compute the error that is fed into the controller described in chapter 3. To access the ASM driver steering position reference signal while the simulation is run with hardware-in-the-loop (as in the case of the Cruden simulator), the Simulink model must be modified in a way that allows the driver to change the `Steer_Mode` flag on the fly mid-simulation. The system of flags used by ASM to implement different steering modes is discussed in the following section.

### 2.2.6 Steering modes in ASM

ASM allows for varying degrees of HiL simulation: from no hardware in the loop, to the system described in section 2.1. In order to manage and redirect the input signals when hardware is connected, ASM uses a system of

boolean or numerical flags to set modes that allow an external (human) driver to provide the steering signal instead of the internal ASM driver described in section 2.2.5.

For the purposes of this project, three flags are of importance: the `Steer_Mode`, `LatDriver`, and `LatCtrl1_Enable` flags. The `Steer_Mode` flag is utilized to internally set a steering mode for the simulation while the `LatDriver` flag is used to select the specific internal lateral controller in the ASM driver model described in 2.2.5. The `LatCtrl1_Enable` flag is utilized to enable/disable the `LATERAL_CONTROL1` block. The `Steer_Mode` flag can take one of three values, each corresponding to a different source for the steering signal:

- `1|Stim` - The steering signal is acquired from a source external to the ASM model. This is the default mode for the Cruden driving simulator since the steering signal comes from the physical steering wheel being controlled by the human driver.
- `2|Driver` - The steering signal is acquired from the driver model implemented within ASM. The ASM driver model is explained in section 2.2.5.
- `3|Fix` - Holds the last steering angel value.

The `LatDriver` flag can take two values, corresponding to different implementations of the lateral controller within ASM:

- `1|LatCtrl1` - Activates the `LATERAL_CONTROL1` block which is appropriate for most driving applications (which causes the `LatCtrl1_Enable` flag to be set to 1 or true).
- `2|LatCtrl2` - Activates the `LATERAL_CONTROL2` block which is appropriate for few special conditions like circular tracks.

The values that these flags can take and their behaviour is summarised in table 2.1 below.

Table 2.1: Boolean flags used for selecting steering input mode

| **Flag** | **Value** | **Meaning** |
|---|---|---|
| `Steer_Mode` | 1 → `Stim` | External stimulus |
| | 2 → `Driver` | ASM driver |
| | 3 → `Fix` | Fixed input |
| `LatDriver` | 1 → `LatCtrl1` | Lateral controller 1 |
| | 2 → `LatCtrl2` | Lateral controller 2 |
| `LatCtrl1_Enable` | 1 → Enabled | Lateral controller 1 enabled |
| | 0 → Disabled | Lateral controller 1 disabled |

# Chapter 3

# Model-free control

Model-free Control (MFC) is a relatively new control concept introduced in 2008 by Michel Fliess and Cedric Join [5]. The aim of this method is to do away with the need for an accurate system model. The term *model-free control* has been used in different contexts in literature - sometimes referring to "classical" PID controllers, to robust and adaptive control systems. However the approach in [5] is rather different from the above mentioned techniques. The core ideas of the MFC approach are real-time algebraic derivative estimation and the ultra-local model, described in the following sections. The resultant controller performs well without the need for a model of the plant and is very robust against signal noise.

## 3.1 Ultra-local model

For the sake of simplicity, the system developed here is assumed to be single-input single-output (SISO) with input variable $u$ and output variable $y$. The plant with an unknown model of arbitrary complexity can be replaced by the following *ultra-local model*

$$y^{(\nu)}(t) = \phi + \alpha u(t) \tag{3.1}$$

The terms of equation 3.1 are explained as follows:

- $y^{(\nu)}$, $(\nu \geq 1)$ is the $\nu$-order time derivative of the output $y$. The value of $\nu$ is chosen by the control designer, however a value of 1 or 2 are known to provide good performance for most systems in practice.

- $\phi$ is a continuously updated quantity representing the unknown dynamics of the plant. Additionally, all disturbances in the system are also captured by this term. $\phi$ is estimated in real-time through techniques shown in section 3.3.

- $\alpha$ is a non-physical, tunable parameter, chosen in a way that $\alpha u$ and $y^{(\nu)}$ are of comparable magnitude. Its value is generally obtained through trial and error by someone with a good understanding of the system's behaviour.

- $u(t)$ is the control input into the system.

The ultra-local model of equation 3.1 can be intuitively understood as an approximation of the plant dynamics that is valid for a very short duration of time (e.g. a few sample times). The estimated value of $\phi$ is considered to be constant during the aforementioned short duration, or piecewise constant during the duration of time. Figure 3.1 is an attempt at giving some visual intuition of the ultra-local model. Consider the true nonlinear response of the system represented as a time-varying hypersurface in the state-space. The hypersurface is approximated by the blue surface (also a function of time) but

is piecewise constant for the estimation window of the parameter $\phi$. Although this picture is not strictly correct, it captures the idea behind approximating a given plant using an ultra-local model.

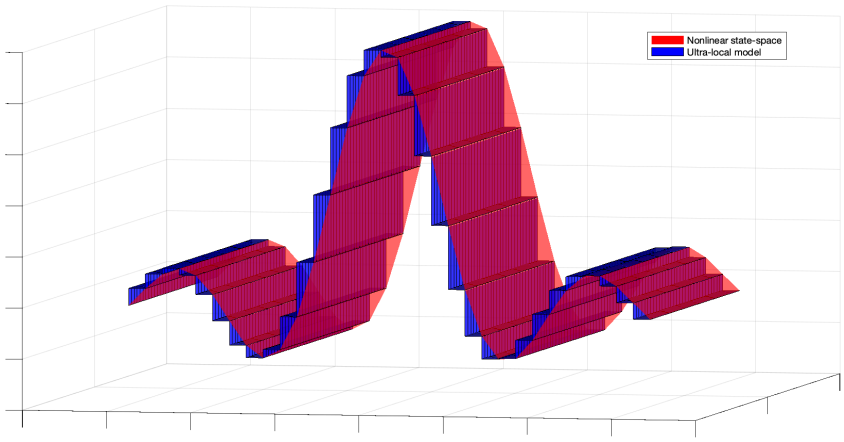



Figure 3.1: Visual intuition for the ultra-local model.

In order to control the position of the CL motor, an ultra-local model of order $\nu = 2$ was selected:

$$\ddot{y}(t) = \phi + \alpha u(t) \tag{3.2}$$

## 3.2 Control law and closed-loop dynamics

For the model developed in section 3.1, a closed loop control law is proposed here for the $\nu = 2$ case. This *intelligent PID* control law incorporates

the linearizing effect of the ultra-local model with the addition of Proportional-Integral-Derivative (PID) action and is given as follows:

$$u(t) = \frac{-\phi + \ddot{y}_d(t) - K_I \int e(t) - K_P e(t) - K_D \dot{e}(t)}{\alpha} \tag{3.3}$$

where $y_d$ is the desired output trajectory, $e = y - y_d$ is the tracking error and $K_P$, $K_I$, and $K_D$ are the PID gains. Closing the loop by combining equations 3.1 and 3.3 results in the following closed-loop dynamics:

$$\ddot{e}(t) + K_D \dot{e}(t) + K_P e(t) + K_I \int e(t)dt = 0 \tag{3.4}$$

It is worth noting that after closing the loop with the control law presented in equation 3.3, the parameter $\phi$, representing the unknown plant dynamics and disturbances, is absent from the above equation. Therefore, the closed-loop dynamics are governed by the linear ODE shown in equation 3.4 and can be tuned by appropriately picking the values for the PID gains.

## 3.3 Real-time estimation of the model parameter $\phi$

The ultra-local model in equation 3.2 can be rearranged to give the following

$$\phi = \ddot{y}(t) - \alpha u(t)$$

Here, in order to estimate the value of $\phi$, one needs to estimate the value of the second-order derivative of the output $y(t)$. In a discrete setting, since the current value of $u(t)$ is unavailable at time $t$, we use the value of

$u((k-1)t)$ in the above equation to avoid an algebraic loop in the digital implementation to get the following difference equation:

$$\phi = \ddot{y}[k] - \alpha u[k-1] \tag{3.5}$$

The algebraic estimation methods developed by Fliess and Ramírez in Fliess, Join, and Ramírez in [6] and [7] are the key innovations that solves this problem in real-time. The following sections describe the expressions for the estimation of the derivative of a given noisy signal.

Let $y(t)$ be a noisy signal whose second-order derivative we wish to estimate. We can write expressions for algebraic derivative estimators of $n^{th}$ order using an $n^{th}$ order approximation of the noisy signal [10]. Presented here are estimators of order 2 and 3 with the explicit derivation done for the case when the order is 2.

### 3.3.1 Estimator of order 2

We can write the second order approximation of $y(t)$ in the time domain and the Laplace domain ($s$-domain) using a Taylor expansion centered around zero (without loss of generality) as follows:

$$y(t) = y(0) + \frac{\dot{y}(0)}{1!}t + \frac{\ddot{y}(0)}{2!}t^2 = a_0 + a_1 t + a_2 t^2 \tag{3.6}$$

$$Y(s) = \frac{a_0}{s} + \frac{a_1}{s^2} + \frac{a_2}{s^3} \tag{3.7}$$

We can multiply the above $s$-domain expression by $s^2$ to isolate $a_1$ so that it may be removed from the calculation:

$$s^2 Y(s) = a_0 s + a_1 + \frac{a_3}{s} \tag{3.8}$$

Performing a $\frac{d2}{ds^2}$ operation to eliminate $a_0$ and $a_1$ and rearranging:

$$2Y(s) + 4s\frac{dY(s)}{ds} + s^2\frac{d^2Y(s)}{ds^2} = 2a_2\frac{1}{s^3}$$

Multiplying by $\frac{1}{s^3}$ to remove all derivatives (since derivatives are non-causal in a discrete sense) and writing the expression as a series of integrators:

$$\frac{2}{s^3}Y(s) + \frac{4}{s^2}\frac{dY(s)}{ds} + \frac{1}{s}\frac{d^2Y(s)}{ds^2} = 2a_2\frac{1}{s^6}$$

We can now obtain an expression for the second-order estimation of the second-order derivative of $y(t)$. To convert the above expression to the time domain, we need to use the Inverse Laplace formula and Cauchy's formula for repeated integration, which would give us the following expression:

$$\int_0^t (t-\tau)^2 y(\tau)d\tau - 4\int_0^t (t-\tau)\tau y(\tau)d\tau + \int_0^t \tau^2 y(\tau)d\tau = 2a_2\frac{t^5}{5!} \tag{3.9}$$

Rearranging the above expression gives us the derivative estimation:

$$a_2 = \ddot{y}(t) = \frac{60}{t^5}\int_0^t (t^2 - 6t\tau + 6\tau^2)y(\tau)d\tau \tag{3.10}$$

If one chooses to eliminate $a_1$ and $a_2$ in equation 3.8, one would be left with an expression for $a_0$ which is the value of the signal itself at the time in question. This estimation would then simply act as a real-time filter for the signal. This phenomenon also explains the high robustness of this method against noise in the system signals.

### 3.3.2 Estimator of order 3

Similar to section 3.3.1, if we use a third order Taylor expansion of the signal and follow the same steps as before, we get the following expression for the third-order estimation of the second-order derivative of $y(t)$:

$$a_2 = \ddot{y}(t) = \frac{120}{t^6} \int_0^t (4t^3 - 45t^2\tau 108t\tau^2 70\tau^3) y(\tau) d\tau \tag{3.11}$$

The discrete implementation of this estimator required to program it on a computer is developed in the following section.

## 3.4 Discrete real-time implementation of algebraic derivative estimator

The general form of a causal, real-time algebraic derivative estimator of a $j^{th}$ order derivative is developed in appendix A and is shown below:

$$a_j = y^{(j)}(t) = (-1)^j \int_0^T P(T,\tau) y(t-\tau) d\tau \tag{3.12}$$

where $T$ is the estimation window and $t$ is the time at which the derivative is evaluated. $P(T,\tau)$ is a polynomial of $t$ and $\tau$ as seen in the previous sections. This can be written in discrete form as a summation shown below:

$$a_j = y^{(j)}(t) \cong (-1)^j \sum_{k=1}^{N+1} \alpha_k P(T,\tau_k) y(t-\tau_k) \tag{3.13}$$

This sum can be computed in real-time using trapezoidal integration. To solve the expression from equation 3.13 in a real-time system, we can reduce the computational power needed by pre-computing the coefficients of

the discretized polynomial $P(T, \tau_k)$. We can also solve for $\alpha_k$, the coefficients that arise from the trapezoidal integration of the sum from equation 3.13. We introduce $P(\hat{T}, \tau_k)$, a vector that contains the coefficients resulting from the trapezoidal integration and discretization of $P(T, \tau)$:

$$P(\hat{T}, \tau_k) = \begin{bmatrix} \alpha_1 P(T, \tau_1) & \alpha_2 P(T, \tau_2) & \dots & \alpha_{N+1} P(T, \tau_{N+1}) \end{bmatrix} \tag{3.14}$$

where $\tau_k = (k-1)T_s$ and

$$\alpha_k = \begin{cases} \frac{T_s}{2}, & k = 1 \text{ or } k = N+1 \\ T_s, & k = 2, 3, \dots, N \end{cases}$$

We also introduce $\hat{y}(t - \tau_k)$, a vector containing the last $N+1$ samples of the signal $y(t)$. At each time $t = kT_s$, the newest sample is pushed to the top and the oldest sample is removed from the vector:

$$\hat{y}(t - \tau_k) = \begin{bmatrix} y(t) \\ y(t - T_s) \\ \vdots \\ y(t - NT_s) \end{bmatrix} \tag{3.15}$$

Using the notation introduced above, we can rewrite equation 3.13 to get an expression for the real-time estimation of the $j^{th}$ order derivative of $y(t)$:

$$a_j(t) = y^{(j)}(t) \approxeq (-1)^j \hat{P}_{\alpha_k}(T, \tau_k) \hat{y}(t - \tau_k) \tag{3.16}$$

# Chapter 4

# Controller Development

In this section, the process of designing the model-free controller for the Cruden driving simulator steering system is explained. The controller is developed within the ASM traffic Simulink model which is then used to compile and generate code to be run as an executable on the real-time SCALEXIO computer. As mentioned in section 1.1.1, the controller was implemented on the actual Cruden driving simulator but there was not enough time to tune it to achieve desired performance. As a result, there are no data showing how the tuned controller would perform on the actual system. Presented in this section is the controller implemented in a similar way in Simulink to control a motor model, as a proof-of-concept. In the following sections, the motor model is described and the results are presented for the controller's performance.

## 4.1 Test of feasibility

In the Cruden simulator, the self-aligning torque experienced by the wheels is sent through to the steering wheel as feedback to the driver. The Control Loading (CL) motor exerts this torque on the steering wheel which is felt by the driver. As previously mentioned in section 2.2.4, the self-aligning mo-

ment is demanded from the CL motor through the `To E2M CL` block inside the `IO` subsystem. This signal pathway was exploited to demand a user-defined torque from the CL motor to test the feasibility of this control approach.

## 4.2 Motor model

The controller developed in this chapter was first tested through simulation on a motor model before being implemented on the actual Cruden driving simulator system. This simulation was used to debug and refine the Simulink code. Although the model parameters for the simulated model are arbitrary, it is useful in testing the model-free control algorithm. The motor model used for this simulation test is presented in this section.

$$J\ddot{\theta} + b\dot{\theta} = K_t i \tag{4.1}$$

$$L_m \frac{di}{dt} + R_m i = V_{in} - K_e \dot{\theta} \tag{4.2}$$

where the symbols have the following meaning and values:

Table 4.1: Values of motor parameters

| **Parameter** | **Value** | **Units** |
|---|---|---|
| Motor resistance $R_m$ | 1.7 | $\Omega$ |
| Motor inductance $L_m$ | 0.5 | H |
| Motor inertia $J_m$ | 0.01 | kg·m$^2$ |
| Motor damping $b_m$ | 0.1 | N·m·s |
| Motor constant $K_t$ and $K_e$ | 0.0057 | N·m/A and V/rad/s |
| Motor position $\theta$ | - | rad |
| Motor velocity $\dot{\theta}$ | - | rad/s |
| Motor current $i$ | - | A |

In order to obtain position from the above equation, the velocity state is simply integrated, making it a third-order plant.

## 4.3 Control design

For the controller implementation on the Cruden driving simulator, the control loop is composed of the steering angle signal being read in from the hardware sensors, which is compared to the reference signal to get the error signal. This error then gets sent to the model-free controller along with the second-order time derivative of the reference signal and other signals. The control loop designed for this system is shown below:

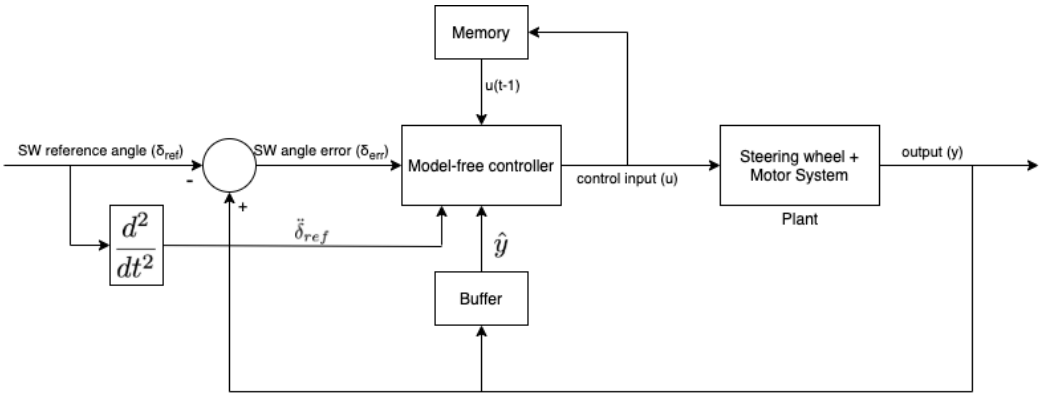
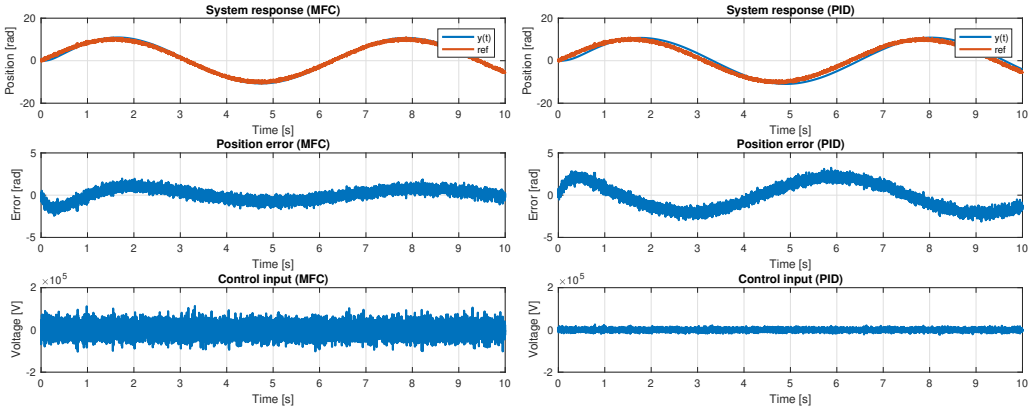



Figure 4.1: Control loop for the steering wheel position control problem.

The $P(\hat{T}, \tau_k)$ vector from equation 3.14 was precomputed offline by calculating the discretized polynomial shown in section 3.4. The Simulink model for the system is shown in appendix B. The system response was tested for the model-free controller and compared to a traditional PID controller. The primary reference signal used was $r(t) = 10\sin(t) + \sin(6t)$ which was polluted with some band-limited white noise. In order to create the $\hat{y}$ vector described in equation 3.15, a `Delay Line` block is used with the delay line size equal to $N + 1$. The output of this block is the inverted $\hat{y}$ vector, therefore a `Flip` block is used to put the vector in the correct orientation. The value of $u(t)$ for the previous time step is obtained by using a `Memory` block. All across the model, Simulink in-built `GoTo` blocks are used to collect the appropriate signals for visualization after the simulation.

## 4.4 Simulation results




(a) MFC (b) PID

Figure 4.2: Comparison of MFC vs. PID when the reference is a noisy signal.

The above figure compares the performance of the model-free controller against that of a PID controller tuned using the usual methods of tuning. It can be seen that the model-free controller performs better in the presence of a noisy signal. In this case, the noise power is 0.0001. Additionally, the PID controller must be tuned for a response on the slower side to be more robust against the noise, adding to its disadvantage. The model-free controller is also more robust against parameter changes when compared to the PID controller for a similar change.

Figure 4.3 shows the real-time algebraic estimation of $\ddot{y}(t)$ and the estimation error. Although the estimation error looks large from the plot below, in actuality the estimation quality is quite good and the $\ddot{y}(t)$ signal is calculated even in the presence of noise amplification from the original signal.

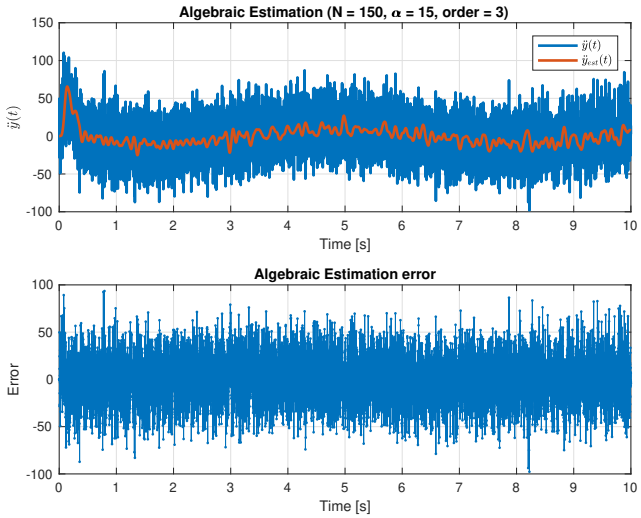


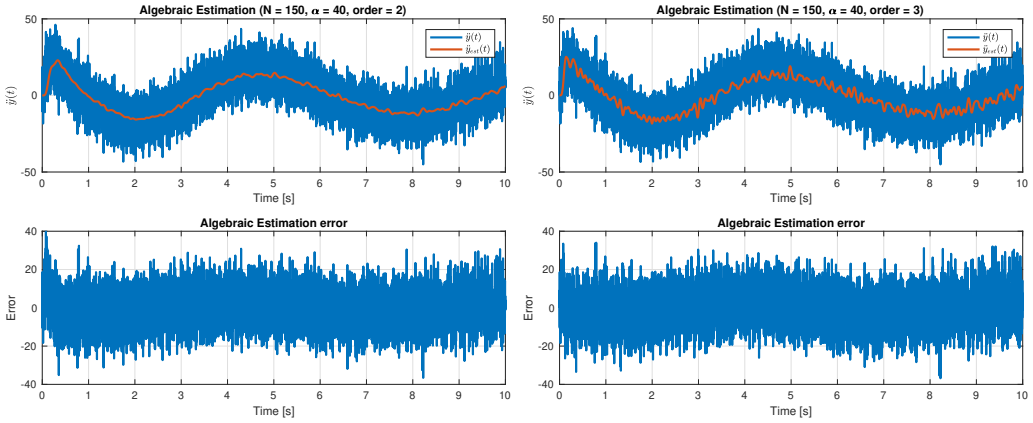

Figure 4.3: Algebraic derivative estimation for the case shown above.

### 4.4.1 Effect of estimation parameters

The performance of the model-free control method is quite sensitive to the parameters in the algebraic estimation step. The estimation window $N$ determines the quality of the algebraic derivative estimation which in turn affects the system response. A smaller value for $N$ means the estimator will run faster and the ultra-local model of the system will be more accurate. Bigger values of $N$ result in higher computational load and slower performance but offer better immunity against noise in the signal. However the estimation window must be large enough so that the numerical discretization error is overcome [10].

The order of the algebraic estimation also affects the quality of the estimation. Continuing the ideas described in section 3.3, the order of the algebraic estimator is the order of the Taylor approximation used to develop the estimation integral. The following figures show the effect of $N$ and estimator order on the system's response.

### Effect of estimation order

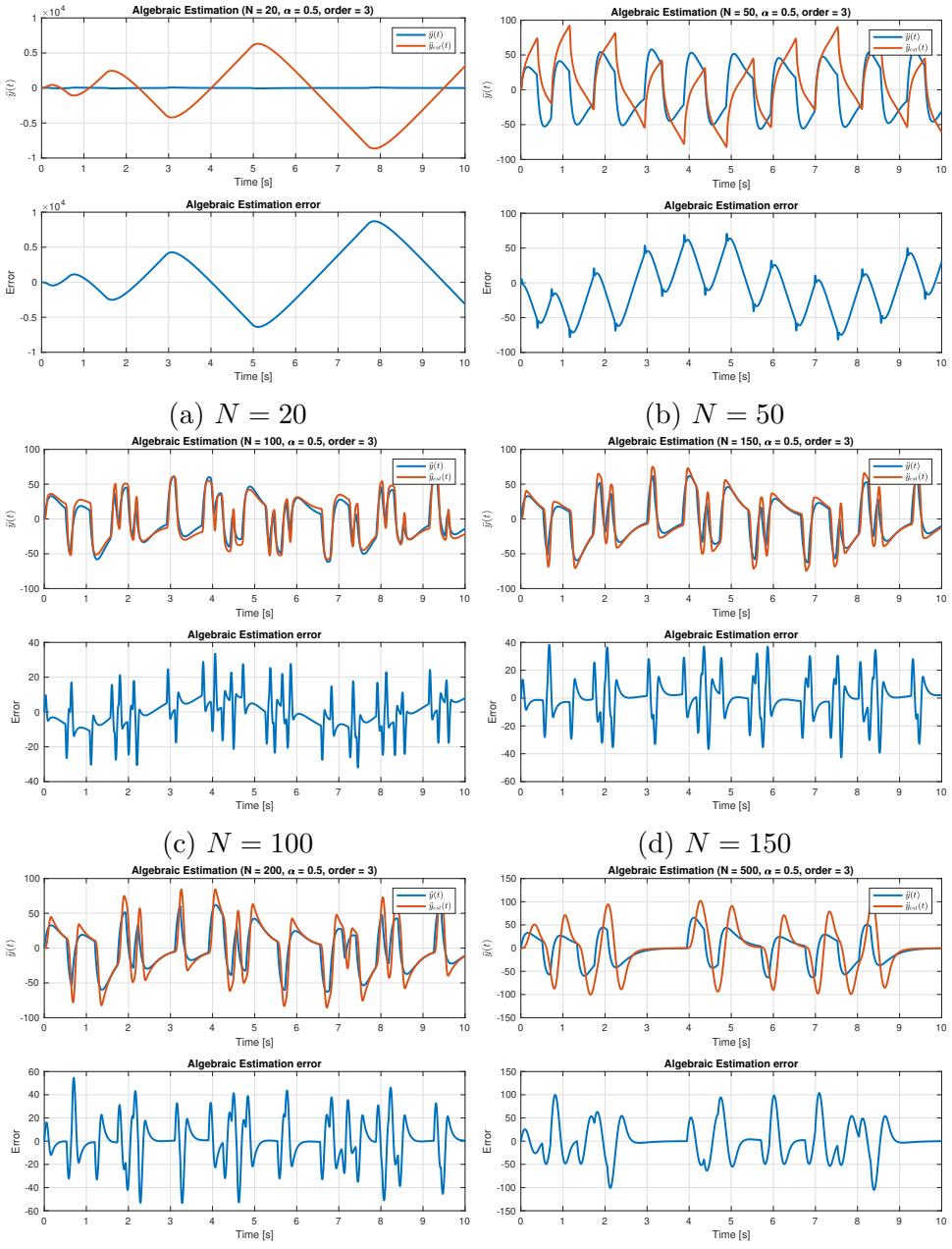



(a) Estimation order 2 (b) Estimation order 3

Figure 4.4: Comparison of a derivative estimator of order 2 vs. a derivative estimator of order 3.

In order to examine the effect of the order of the estimator on the quality of the estimation, band-limited white noise of power 0.0001 was added to the reference signal. The above figure shows how the order of the estimator affects the estimation of the derivative signal. The higher-order estimator captures more of the signal's movement, as is expected since this estimator is derived using a better approximation of the original noisy signal. On the other hand, the lower order estimation then appears to have a *smoothing* effect on the signal. The phase of the filter/estimator can be tuned by selecting an appropriate window, in this case $N = 150$. It is also noteworthy that a change in the value of estimation window $N$ has a similar effect for both the second-order and third-order estimators.

**Effect of $N$**




(a) $N = 20$ (b) $N = 50$

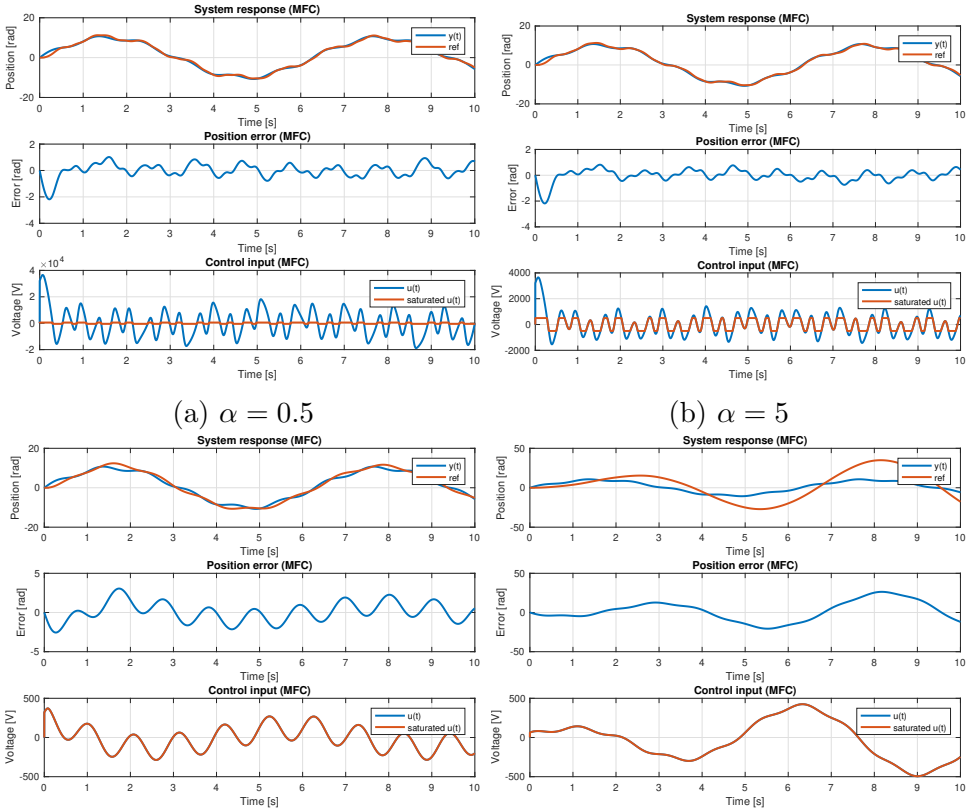
(c) $N = 100$ (d) $N = 150$

(e) $N = 200$ (f) $N = 500$

Figure 4.5: Comparing the effect of estimation window size on the quality of the algebraic estimation.

The figure above shows how the quality of the algebraic derivative estimation is affected by the estimation window parameter. As can be noted, there is a range of appropriate length of the estimation window that results in a good quality estimate. Values of $N$ that are too small or too big cause the estimation to drift and in turn, affect the control performance.

### 4.4.2 Effect of controller parameter $\alpha$




(a) $\alpha = 0.5$ (b) $\alpha = 5$

(c) $\alpha = 50$ (d) $\alpha = 250$

Figure 4.6: Comparing the effect of changing $\alpha$ on the system response and controller performance.

As per section 3.1, the $\alpha$ parameter is tuned such that $y^{(\nu)}$ and $u$ are of the same magnitude. As expected, changing the value of $\alpha$ scales the magnitude of the control signal. When the value of $\alpha$ is very large, as in the case of figure 4.6d, the control input is scaled to be proportionally small, therefore reducing the system performance in this case.

It must be noted, however, that as of the writing of this thesis, there is no well-defined heuristic or method for tuning the $\alpha$ parameter - that is still an area of active research.

# Chapter 5

# Conclusions and Future Work

The simulation results show that the model-free controller approach is feasible in developing automatic steering on the Cruden driving simulator hardware. The final step in implementing a functional system is tuning the controller to achieve the desired performance. To improve on the results presented in this work, the model-free controller could be tested for a non-linear motor model with noise. Also, the effect of simulation step size can be studied on the performance of the system. One limitation of the model-free controller is its poor performance for reference signals with infinite derivatives such as a step input. However, for any steering system, the reference signal will always be a smooth analytical signal and this issue will not arise for similar systems.

As far as the hardware implementation is concerned, an additional feedforward term could be added to the controller in the future to deal with the friction in the system and improve tracking performance.

The algebraic estimation methods used to determine the output derivatives worked very well. One further experiment would be to compare the effect of cascading lower order estimators and comparing their performance with a higher order estimator of the same total order.

# Appendices

# Appendix A

# Algebraic Derivative Estimation

## Deriving the general derivative estimator [5], [7], [10]

Let $\tilde{y}(t)$ be an analytical (infinitely differentiable) noisy signal in the time domain. This signal can be approximated by a truncated Taylor expansion

$$\tilde{y} \approxeq y(t) = \sum_{j=0}^{n} y^{(j)}(0)\frac{t^j}{j!} = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n \tag{A.1}$$

where $y^{(j)}(0)$ is the initial condition for the $j^{th}$ order derivatives of the signal, $n$ is the order of the approximation (and therefore the order of the estimator used), and the approximation is about $t = 0$ without loss of generality. Taking the Laplace transform of the above equation, we get the frequency domain signal as a series of integrators:

$$Y(S) = a_0 + \frac{a_1}{s^2} + \frac{a_2}{s^3} + \ldots + \frac{a_n}{s^{n+1}} \tag{A.2}$$

In the general case, the $n^{th}$ order estimator for the $j^{th}$ order derivative can be obtained by manipulating the above expression to isolate the appropriate term and converting back to the time domain to get the following

$$y^{(j)}(t) \approxeq y^{(j)}(0) = a_j = \frac{k}{p(t)} \int_0^t \tilde{P}(t,\tau) y(\tau) d\tau, \ (0 < t < \epsilon) \tag{A.3}$$

where $t$ is the time of estimation and $\epsilon$ is the interval during which this estimation is valid, $\tau$ is the variable of integration, $p(t)$ is a polynomial of $t$, $\tilde{P}(t,\tau)$ is a polynomial of both $t$ and $\tau$, and $k$ is a constant scalar. The above derivative estimator can be implemented in a fixed length window $T$ using the following expression:

$$y^{(j)}(0) = a_j = \frac{k}{p(T)} \int_0^T \tilde{P}(T,\tau) y(\tau) d\tau = \int_0^T P(T,\tau) y(\tau) d\tau \tag{A.4}$$

The expression in equation A.4 is not a causal estimator, because in order to estimate $y^{(j)}(\tau = 0)$, we need values of $y(\tau)$ for $\tau \in [0, T]$. We must transform this estimation such that it is a causal realization for it to be implemented in real-time. Consider another signal $\tilde{z}(t)$ approximated by the following Taylor expansion and $j^{th}$ order derivative estimators around $\tau = 0$

$$\tilde{z}(\tau) \approx z(\tau) = \sum_{j=0}^{n} a_j \frac{\tau^j}{j!} \implies a_j = \int_0^T P(T,\tau) z(\tau) d\tau$$

Make the following substitutions:

$$\tau \triangleq t - \theta \text{ and } y(\theta) \triangleq z(t - \theta)$$

Here, $\tau$ can now be seen as a time-shifted time variable while $z(\tau)$ is a time shifted signal. We can now write a Taylor expansion of $\tilde{z}(\tau)$ around $\theta = t$ as

$$y(\theta) \triangleq z(t - \theta) = \sum_{j=0}^{n} a_j \frac{(t-\theta)^j}{j!}$$

Now, consider the following expression

$$y(t) = z(t - \theta)|_{\theta = t} \text{ and}$$

$$y^{(j)}(t) = \left[\frac{d^j}{d\theta^j}\left(z(t-\theta)\right)\right]_{\theta=t} = \left[\frac{d^j}{d\theta^j}\left(\sum_{j=0}^{n} a_j \frac{(t-\theta)^j}{j!}\right)\right]_{\theta=t}$$

Upon working out the derivatives in the expression on the right hand side of the above equation, it can be shown that

$$y^{(j)}(t) = (-1)^j a_j$$

$$\text{where } a_j = \int_0^T P(T,\tau) z(\tau) d\tau = \int_0^T P(T,\tau) y(t-\tau)$$

Therefore, we can now write a causal derivative estimator calculated at time $t$ using an estimation window of length $T$

$$a_j = y^{(j)}(t) = (-1)^j \int_0^T P(T,\tau) y(t-\tau) d\tau \tag{A.5}$$

The integral in equation A.5 can be numerically solved, which will give its approximate value (within arbitrary numerical precision, provided the availability of sufficient computing power), using the trapezoidal integration rule. We can make $T = T_s N$, where $T$ is the window of estimation, $(N+1)$ is the number of samples used for estimation, and $T_s$ is the sampling time. The expression from equation A.5 expressed as a discrete sum is as follows:

$$a_j = y^{(j)}(t) \approxeq (-1)^j \sum_{k=1}^{N+1} \alpha_k P(T,\tau_k) y(t-\tau_k) \tag{A.6}$$

# Appendix B

# Simulink block diagrams

$$u(t) = \frac{-\phi + \ddot{y}_d(t) - K_I \int e(t) - K_P e(t) - K_D \dot{e}(t)}{\alpha}$$

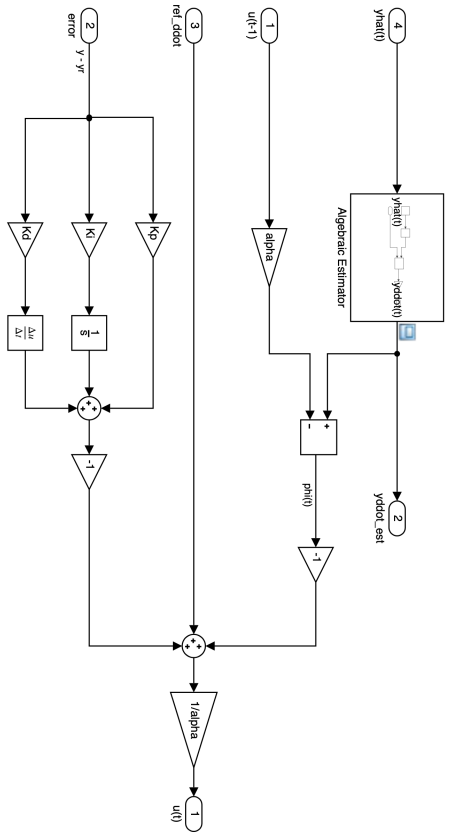



Figure B.1: Simulink implementation of the model-free controller algorithm.

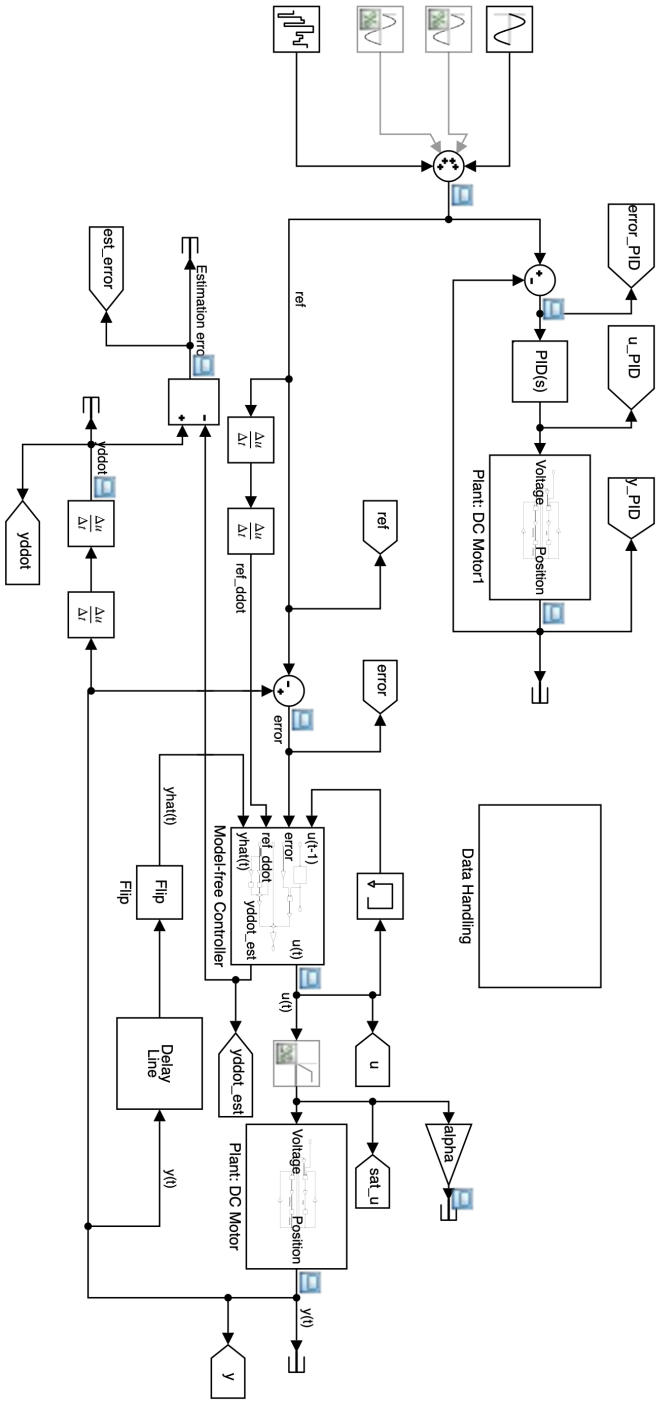



Figure B.2: Simulink block diagram of the model-free controller simulation.

# Bibliography


[1] Cruden B.V., Pedro de Medinalaan 25, 1086 XP, Amsterdam, The Netherlands. *Automotive Simulator Manual*, v11 edition.

[2] dSPACE. Asm vehicle dyamics (brochure), 2013.

[3] dSPACE, dSPACE GmbH, Rathenaustaße 26, 33102 Paderborn, Germany. *ASM User Guide*, release 2018 a edition, May 2018.

[4] dSPACE, dSPACE GmbH, Rathenaustaße 26, 33102 Paderborn, Germany. *ASM Environment Reference*, release 2019 a edition, May 2019.

[5] Michel Fliess and Cédric Join. Model-free control. *International Journal of Control*, May 2013.

[6] Michel Fliess, Cédric Join, and Herbertt Sira-Ramírez. Non-linear estimation is easy. *Int. J. Modelling Identification and Control*, Special Issue on Non-Linear Observers 4(1):12–27, October 2008.

[7] Michel Fliess and Herbertt Sira-Ramírez. An algebraic framework for linear identification. *ESAIM: Control, Optimisation and Calculus of Variations*, 9:151–168, January 2003.

[8] Marno Hopmans, Ruud van Gaal, Nico Kruithof, Edwin de Vries, Jelle van Doornik, and Robbert van Hassel. *ePhyseNet Guide*. Cruden B.V.,

Pedro de Medinalaan 25, 1086 XP, Amsterdam, The Netherlands, v11.11 edition, January 2019.

[9] Mamadou Mboup, Cédric Join, and Michel Fliess. Numerical differentiation with annihilators in noise environment. *Numerical Algorithms*, (50):439–467, 2009.

[10] Matheus Schwalb Moraes. Algebraic derivative estimation applied to nonlinear control of magnetic levitation. Master's thesis, Escola Politécnica, Universidade de São Paulo, 2016.

[11] R. Morales, J.A. Somolinos, and H. Sira-Ramírez. Control of a dc motor using algebraic derivative estimation with real time experiments. *Measurement: Journal of the International Measurement Confederation*, 47:401–417, January 2014.

[12] E2M Technologies. eforce control loading systems (pss). Website.

[13] Alf J. van der Poorten. Some problems of recurrent interest. Technical Report 81-0037, School of Mathematics and Physics, Macquarie University, North Ryde, Australia 2113, August 1981.

[14] Zejiang Wang. *Cruden operation guides*. Mobility Systems Laboratory at UT Austin, October 2019.