POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering

Master's Degree Thesis

Modeling Approaches for Vehicular ECUs Networks for Synthetic Data Generation.

Supervisors

Prof. ALESSANDRO SAVINO

Prof. STEFANO DI CARLO

Dr. FRANCO OBERTI

Candidate

SADEK MISTO KIRDI

April 2024

Abstract

This master's thesis is dedicated to an in-depth examination of Battery Electric Vehicles (BEVs), with a pronounced emphasis on constructing a detailed systematic model of BEVs and seamlessly integrating the Controller Area Network (CAN) platform. Utilizing Simscape within the Simulink environment as the foundational framework, the primary objective is to simulate cybersecurity attacks on vehicular systems. This approach allows the generation of attack scenarios common to BEVs, providing invaluable insights into the resultant behaviors and vulnerabilities exposed by these cybersecurity breaches.

Central to this thesis is the development and application of a BEV model in Simulink, designed specifically to facilitate the emulation of CAN injection attacks. This innovative methodology enables the study to not only generate data but also to directly observe the effects of various cyber threats on the BEV's operational integrity and safety. By conducting a series of simulated attacks, the research offers a unique perspective on the potential impacts these security breaches may have on BEVs, thereby underscoring the critical need for advanced protective measures in the automotive domain.

Further exploration within the thesis encompasses a comprehensive review of the existing literature on CAN injection attacks, alongside a detailed investigation into the deployment and analysis of such attacks on the developed BEV model. This investigation aims to assess the effectiveness of different cyberattacks and their consequent effects on the BEV system, thus highlighting the importance of implementing robust cybersecurity strategies.

In conclusion, the thesis reiterates the essential findings and stresses the imperative for enhanced cybersecurity protocols in BEVs. It advocates for future research directions, including the implementation of sophisticated intrusion detection systems and the adoption of secure communication frameworks, to mitigate the risks associated with CAN injection attacks. This study significantly contributes to the ongoing discourse on safeguarding BEVs against cyber threats, advocating for proactive and advanced measures to maintain the vehicles' reliability and integrity in an interconnected automotive landscape.

Acknowledgements

I would like to express my deepest gratitude to my family for their unwavering support throughout this journey. To my parents, Ibrahim and Nawal, your endless encouragement, patience, and understanding have been the cornerstone of my academic pursuits. Your sacrifices and belief in me have given me the strength to persevere, and for that, I am eternally grateful.

I am also immensely thankful to my brother, Mohamad, for his constant encouragement, and guidance and for being a source of inspiration. Your belief in my abilities has been a constant source of motivation, and I am grateful to have you by my side.

I extend my sincere appreciation to my professors and supervisors, Professor Alessandro Savino, Professor Stefano Di Carlo, and Doctor Franco Oberti for offering me this opportunity and guiding me through this journey. Your guidance, expertise, and support have been invaluable throughout the research process. Your insightful feedback and constructive criticism have helped shape this thesis into its final form. I am deeply grateful for the time and effort you have dedicated to mentoring me.

I would also like to thank my friends, colleagues, and everyone who was a part in this journey for their contributions and support during the past years.

Lastly, to all those who have supported me in ways both seen and unseen, thank you. Your belief in me has made all the difference.

Sadek Misto Kirdi.

Table of Contents

List of Tables

List of Figures

Acronyms

BEVs	Battery Electric Vehicles
BEV	Battery Electric Vehicle
ICE	Internal Combustion Engine
USA	United States of America
AC	Alternating Current
DC	Direct Current
CAN	Control Area Network
ISO	International Standards Organization
ECUs	Electronic Control Units
ECU	Electronic Control Unit
RPM	Revolutions Per Minute
ABS	Anti-lock Braking System
LIN	Local Interconnect Network
OSI	Open Systems Interconnection
SOF	Start of Frame
ID	Identifier
RTR	Remote Transmission Request
DLC	Data Length Code
CRC	Cyclic Redundancy Check
ACK	Acknowledgment Field
EOF	End of Frame

IFS	Inter-frame Space
SRR	Substitute Remote Request
IDE	Identifier Extension
FD	Flexible Data Rate
XL	eXtra Large
CiA	CAN in Automation
BRS	Bit Rate Switching
VNT	Vehicle Network Toolbox
PMSM	Permanent Magnet Synchronous Motor
IM	Induction Motor
PWM	Pulse Width Modulation
PI	Proportional Integral
FOC	Field Oriented Control
DOS	Denial of Service
AMDC	Artemis Motorway Driving cycle
WHVC	World Harmonized Vehicle Cycle
EUDC	Extra Urban driving cycle

Chapter 1

Introduction to BEVs

1.1 Introduction

Electric vehicles have been around since the 19th century, predating gasoline-powered automobiles. They were quite popular in the late 1800s and early 1900s. However, by the 1920s, their prevalence declined dramatically due to competition from cheaper, gasoline-powered vehicles that could take advantage of abundant, affordable oil. Though electric cars have a long history, spanning over a hundred years, the 20th-century rise of gas-powered models temporarily eroded the electric vehicle market.
In the automobile industry, the real transition from Internal Combustion Engine (ICE) vehicles to BEVs started to pick a significant pace between the early 2000s and mid-2010s.

From then on, several reasons contributed to the current trend toward BEVs:

1. Concerns about the environment: As people become more conscious of how emissions from fossil fuels affect the climate, interest in electric cars as greener options have grown.

2. Technological developments: Lithium-ion battery technology has improved energy density and driving range, making electric vehicles more useful for daily use.

3. Government incentives and policies: To promote the use of electric vehicles, numerous governments worldwide have put in place tax breaks, subsidies, and pollution rules that favour cars with zero emissions, as seen in Figure 1.1.

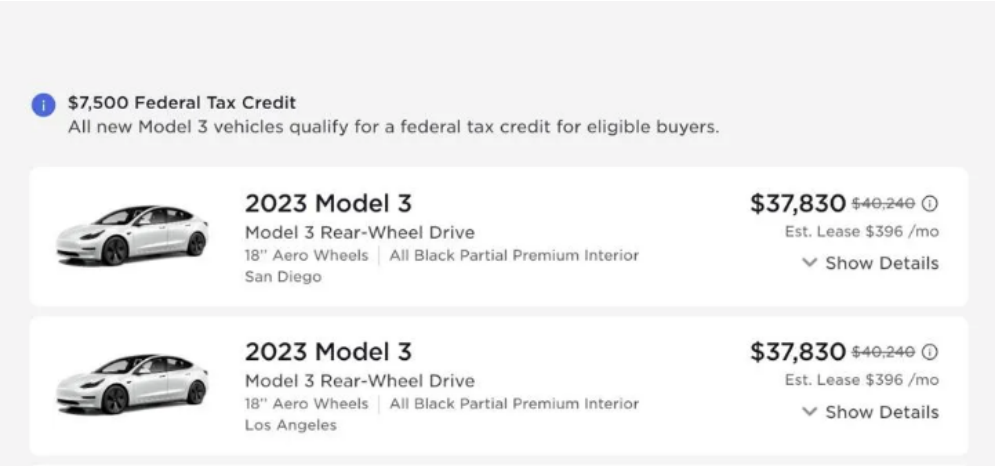


Figure 1.1: Tax Relief for Tesla vehicles in the USA.[1]

4. Industry innovation and investment: Automakers began investing heavily in electric vehicle technology, developing new models and infrastructure to support widespread adoption.

With the introduction of high-performance, long-range electric automobiles like the Tesla Roadster and later the Model S, businesses like Tesla Motors were instrumental in reviving interest in electric vehicles around 2010. The transition from conventional ICE vehicles was further accelerated by other big automakers who progressively followed suit and introduced their electric vehicle variants. Since then, the market has embraced BEVs due to rising investments in research and development and the creation of infrastructure (such as charging networks) and improved performance and environmental advantages of electric vehicles, as seen in Figure 1.2.

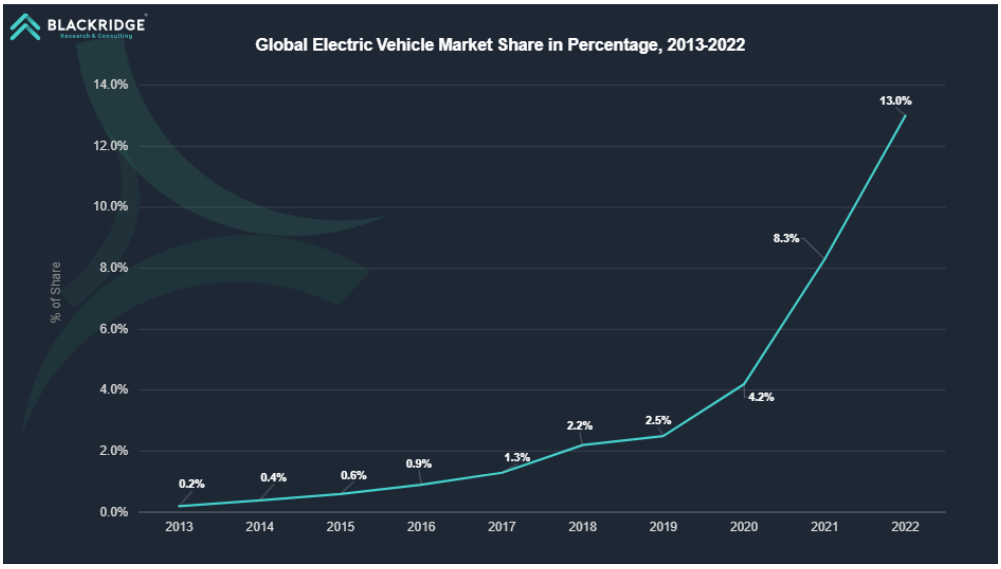


Figure 1.2: Global BEV market share growth between 2013 and 2022.[2]

1.2 Overview of Battery Electric Vehicles

A BEV is a kind of electric car that runs entirely on electricity and is kept in rechargeable batteries to power an electric motor. BEVs emit no pollutants from their tailpipe and lack an internal combustion engine. To refuel, they need to be connected to an outside power source, like an outlet or charging station.

Key features of BEVs include:

- **Electric Motor:** BEVs have an electric motor that runs on the energy stored in the batteries. Electric motors, including synchronous, permanent magnet, and Alternating Current (AC) induction motors, are employed in BEVs. The motor's power output and efficiency greatly influence the vehicle's performance.
- **Battery Pack:** An essential part of the vehicle's electric motor power storage system is the battery pack. The battery pack is made up of several separate battery cells that are connected either in parallel or in series. These cells could be made of nickel-metal hybrid, lithium-ion, or another rechargeable battery. The battery pack's capacity and energy density dictate the car's distance between charges. The performance, economy, and range of BEVs can all be enhanced by battery advancements.
- **Charging:** For battery recharging, BEVs require an external power supply. Charging at public charging stations or at home using a regular electrical socket is possible. The charge is done at various levels:
  - **Level 1 Charging:** Uses a standard household outlet. It is the slowest charging method but is widely available.
  - **Level 2 Charging:** Requires a dedicated charging unit. It's faster than Level 1 charging, is commonly found at public charging stations, and can be installed at homes with charging stations.
  - **Level 3 DC Fast Charging:** Provides rapid charging by delivering high-voltage DC power directly to the battery. This charging method is found at public fast-charging stations and significantly reduces charging times.
- **Zero Emissions:** Since they operate solely on electricity, BEVs produce zero tailpipe emissions, making them environmentally friendly.

On the other hand, Hybrid vehicles differ from BEVs in several ways:

Combination of Power Sources: Hybrids utilize an internal combustion engine and an electric motor powered by a smaller battery pack. They can operate using a gasoline engine, an electric motor, or a combination.

Lower Electric-Only Range: Hybrids usually have a limited electric-only range compared to BEVs since their electric motors are smaller and designed to assist the internal combustion engine rather than solely power the vehicle.

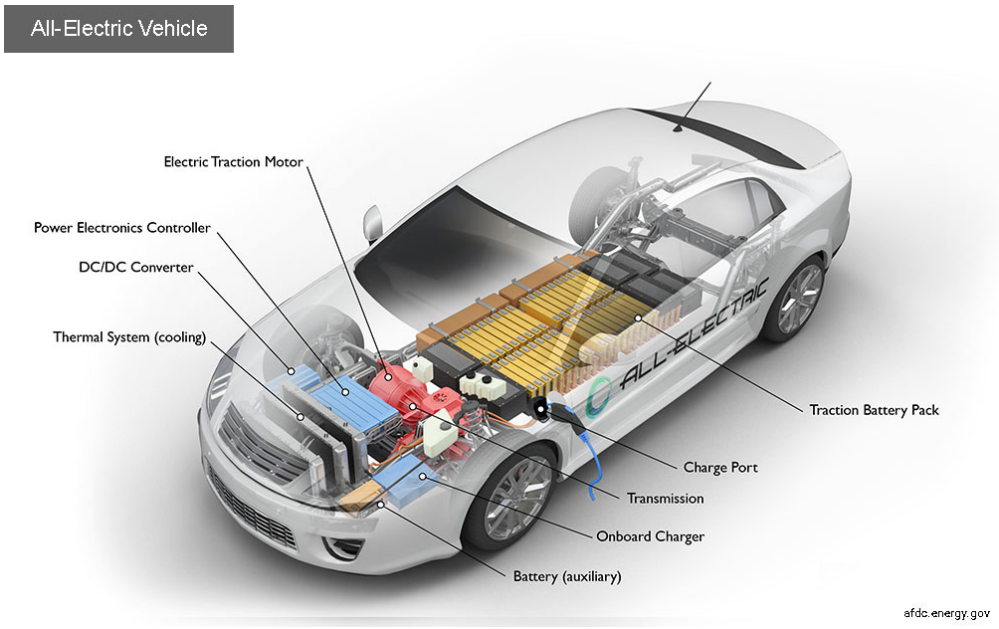


Figure 1.3: BEV Main Components.[3]

1.3 Thesis Outline

After the general introduction about BEV in the first chapter, in the second chapter, we examine the theoretical framework surrounding the CAN network, encompassing its structure, communication protocols, and working principles. Subsequently, in chapter three, we devise a simulation model using Simscape to emulate a BEV.In chapter four, we discuss incorporating the CAN network into the simulation above model. The emphasis will be on conducting various simulated attacks on the CAN network within the BEV model to assess its susceptibility to cyber threats and the results will be presented and analyzed. Finally, the last chapter will outline avenues for future research, exploring potential enhancements and strategies to fortify the security of CAN networks in automotive systems.

Chapter 2

CAN: Functionality and Key Aspects

2.1 Introduction

The Controller Area Network (CAN) is a communication protocol widely used in automotive and embedded systems to enable robust and efficient data transmission between various components within a vehicle. Initially developed by Bosch in the mid-1980s, CAN has become a standard in the automotive industry due to its reliability, real-time capabilities, and cost-effectiveness.

Automakers used point-to-point wiring techniques to link electronics in cars before the development of CAN. The number of devices in cars expanded, and with it did the length of wires required to link them. Both the vehicle's price and weight increased as a result. It also complicated the wiring system and made it challenging to locate errors[4, 5]. Bosh developed the CAN in an attempt to solve this issue. Bosh released CAN Specification 2.0 in 1991 to standardize the CAN protocol. The International Standards Organization (ISO) recognized the CAN protocol in 1993 and published ISO 11898-1 as the international standard. Further enhancing the CAN specification [6] is ISO 11898-2.

2.2 How CAN works

In contrast to conventional networks like Ethernet or USB, CAN does not transfer big data blocks between nodes A and B while managed by a central bus master. CAN is a peer-to-peer network. This means there is no master to control the activities of the devices on the network. Any nodes can start the transmission of messages whenever the bus is free. The transmitted CAN messages do not have any destination address but are visible to all nodes on the network. The receiving nodes use the message arbitration identifiers in the messages to decide whether to accept the message, but they are visible to all nodes on the network. The communication architecture of the CAN is built on an event-triggered, message-oriented structure that facilitates effective data transfer between Electronic Control Units (ECUs) in automobiles. The message-oriented and event-triggered communication modes are the two main communication modes that underpin the design of CAN.

1. **Message-Oriented Communication:**

   - Frames: Data is sent in frames via CAN's message-based communication protocol. These frames have extra control bits for synchronization, error checks, and an identifier and data payload.
   - Priority of Message: A unique identifier is assigned to every CAN message, establishing its priority throughout the network. Higher priority messages are indicated by lower identifier values, which permits crucial material to take precedence over less important information.
   - Broadcasting: Every node (ECU) connected to the CAN bus receives messages via broadcast. Every node receives a message and, using its specified filters assesses the identification to see if the message pertains to its purpose.

2. **Event-Triggered Communication:**

   - Asynchronous Transmission: CAN functions based on an event-based mechanism. CAN nodes communicate by transmitting messages when certain events, like changes in sensor readings or the occurrence of specified system events, happen, instead of adhering to a rigid, preset timetable like in Local Interconnect Network (LIN) for example.
   - Non-Polling Nature: CAN bus nodes do not constantly query one another for information. Rather, they only communicate when it's essential or when something happens that makes it imperative to do so.

A CAN network ensures data integrity across all system nodes by broadcasting several brief messages, such as temperature or Revolutions Per Minute (RPM), to the whole network. Data usability is referred to as data consistency. The multiple ECUs that comprise the vehicle's CAN network exchange data, ranging from the motor to the Anti-lock Braking System (ABS) controller. The LIN bus standard is used for non-critical subsystems such as air-conditioning, where data transmission speed and reliability are less critical.

2.3 CAN Layers

2.3.1 CAN Physical Layer

Following the layer-based Open Systems Interconnection (OSI) architecture, the CAN communications protocol, ISO-11898: 2003, specifies the information flow between devices on a network. The model's physical layer defines communication between devices connected by the physical medium. The data-link layer and physical layer in Figure 2.1 are the lowest two tiers of the seven-layer OSI/ISO model according to the ISO 11898 design.

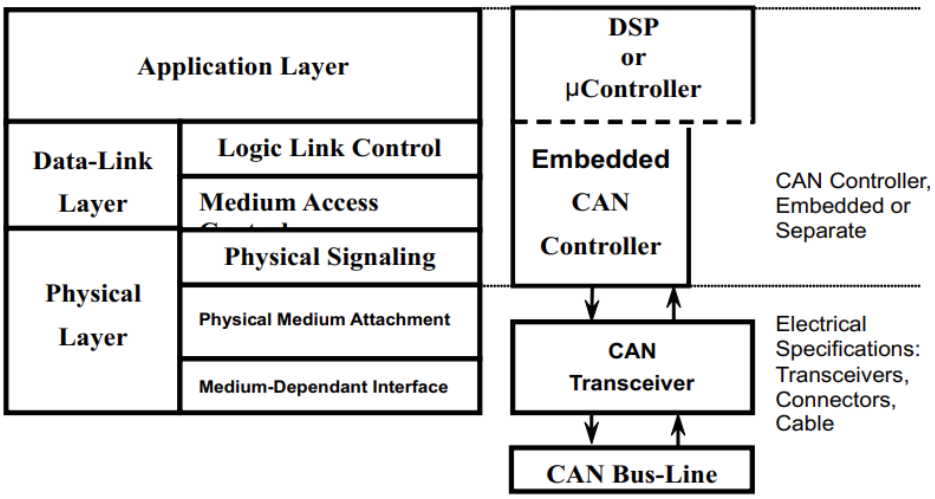


Figure 2.1: The Layered ISO 11898 Standard Architecture.[7]

The above layers of the OSI layers architecture, such as the Network layer, are useless because any routing protocol is needed in a closed network where only ECUs are addressed. The CANprotocol uses differential signalling - a two-wire (twisted-pair) communication line, which includes:

- CAN high
- CAN low

Using just one pair of wires to connect each Electronic Control Unit (ECU) in the network, these two wires transfer all data between them. However, as more ECUs were added to the system, making it more complex, engineers had to split them into different CAN bus "branches". These "branches" will communicate with each other via the "gateway", as illustrated in Figure [2.2]. Despite being split, each CAN bus branch still follows the general idea: All ECUs in their system are connected using only one pair of wires. The passive bus architecture of the Physical layer present in Figure 2.3 comprises copper twisted pair wires with a nominal impedance of 120Ω. This bus employs differential wired-AND signals, which involve transmitting two complementary signals to subtract the signals to remove noise. Two signals, CAN low (CAN_L) and CAN high (CAN_H), are either not driven and dragged to a recessive state with $CAN_H \leq CAN_L$ or driven to a dominant state with $CAN_H \geq CAN_L$. The wired-AND approach, which gives nodes with lower ID numbers priority on the bus, is supported by a 0 data bit that encodes a dominant state and a 1 data bit that encodes a recessive state.[9]

The voltage levels used in CAN bus communication depend on the physical layer specification. For high-speed CAN (ISO 11898-2), the voltage levels typically range from 0V to 5V.

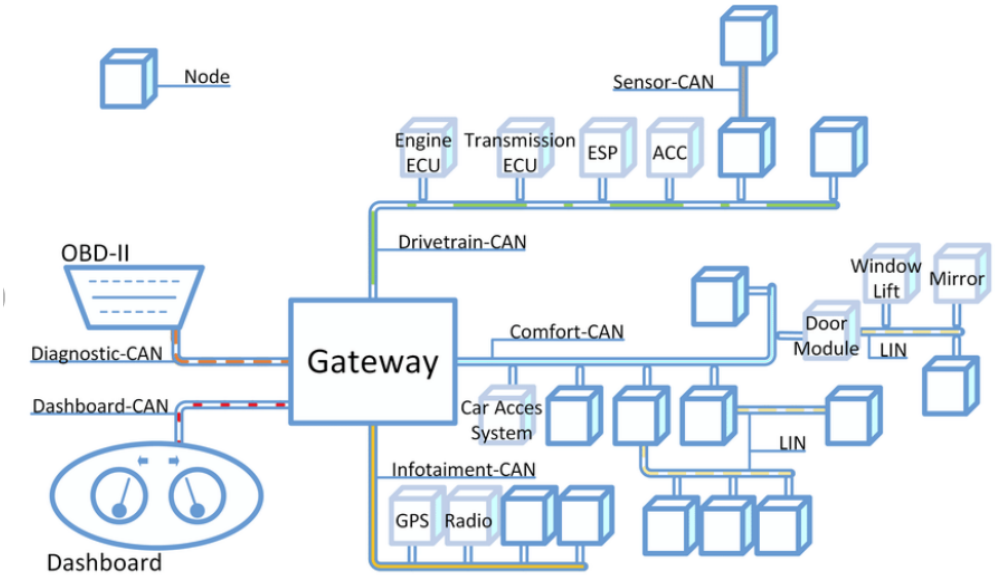


Figure 2.2: ECUs connected through Gateway.[8]

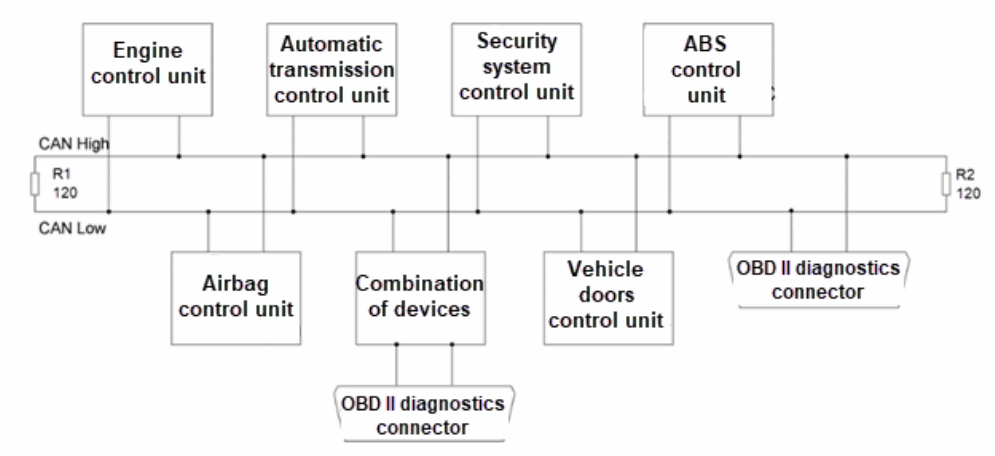


Figure 2.3: ECUs connected to the physical layer.[10]

2.3.2 CAN Data Link Layer

From the Physical layer to the Data link layer, Data is encapsulated into frames, and arbitration determines which frames are to be sent first. There are two standards that the CAN protocol can adhere to regarding the data packet structure. With the standard 11-bit identification, the ISO 11898:2003 typically offers data speeds ranging from 125 kbps to 1 Mbps (Figure 2.4-a) or the "extended" 29-bit identifier (Figure 2.4-b) which was later added to it.

Figure 2.4: CAN Frame.[11]

As seen in figure 2.4-a), the frame comprises several components:

- Start of Frame (SOF): Marks the beginning of a CAN frame, and it is a single dominant bit to reserve the bus for transmission and synchronization.
- Identifier (ID): Identifies the message and determines its priority. Standard CAN uses 11-bit identifiers (CAN 2.0A), while extended CAN uses 29-bit identifiers (CAN 2.0B). Lower ID values indicate higher priority.
- Remote Transmission Request (RTR): Is the Remote Transmission Request bit an extra-priority bit for Data Frames. For a Data frame RTR has a dominant bit while for a Remote Frame, RTR has a recessive bit.
- IDE is the Identifier Extension bit, and is set to "0" if the frame is in standard form.
- r0 is a recessive bit at the transmitter.
- Data Length Code (DLC) is made of 4 bits to indicate the number of bytes (0÷8) of the Data. With r0 and IDE, it forms the Control Field.
- Data Field: Contains the actual payload/data to be transmitted (up to 8 bytes in standard CAN).
- Cyclic Redundancy Check (CRC): Provides error detection capabilities to ensure data integrity. It is made up of 16 bits(15 bits for parity check and 1 for delimiter).
- Acknowledgment Field (ACK): Indicates acknowledgment of the message by receiving nodes. Set as a recessive bit by the sender and then changed to a dominant bit by the receiver.
- End of Frame (EOF): Marks the end of the CAN frame with 7 bits and it must be recessive, and it turns off bit stuffing, which is a technique to ensure enough transitions to maintain synchronization by inserting a bit of opposite polarity after five consecutive bits of the same polarity.

- Inter-frame Space (IFS) is the Inter-frame Space with 7 bits containing the time the controller requires to move a correctly received frame to its proper position in the network [7].

The Extended CAN protocol data frame is seen in Figure 2.4-b). Because the Identifier may take on up to 2^{29} values, this new protocol has the benefit of increasing the maximum number of nodes in the network. It conveys the same standard message but with the following additions:

- Substitute Remote Request (SRR): This type of request replaces the RTR with a Standard frame in its place. If the 11-bit identification is the same, there is a recessive bit that gives the Standard frame priority.
- The Identifier Extension (IDE) is a recessive bit for the Extended CAN data frame that comes just after the SRR.
- In the case of Extended format, r1 is an extra reserve bit that is recessive.

In CAN protocol, various frames facilitate communication between the ECUs on the CAN bus. These frames serve different purposes and contain specific fields to send information. There are four main frames in the CAN protocol:

1. **Data Frame:**
   - Most often utilized frame type.
   - Structure: Consists of an Identifier (ID), Data field (payload), Control bits (for data length and error checking), and CRC for error detection.
   - Use: Actual data is carried in data frames for node-to-node transmission. Within the CAN network, they are used to transmit information, sensor readings, control commands, and other relevant data.
2. **Remote Frame:**
   - Used For communication between requests and responses.
   - Structure: Lacks a data payload but has an Identifier (ID) and Control bits, much like a data frame with an empty payload.
   - Use: A node sends a remote frame to another node to request data. After receiving a remote frame with a matched identifier, the receiving node replies with a data frame with the desired information.
3. **Error Frame:**
   - To indicate communication issues that were found (Communication Errors)
   - Structure: Contains an Error Flag and Error Control field and the fixed format for error signalling.

- Use: Sent by any node that notices a mistake (such as a bit stuffing error, CRC error, or other error conditions). Error frames notify other nodes about problems occurring during communication.

4. **Overload Frame:**

   - Notifies the CAN bus of an overload situation.
   - Structure: Has two flags: Active Error and Overload.
   - Use: Overload frames are transmitted to notify nodes to postpone transmission to lower bus load when the CAN bus is overloaded due to excessive message flow.

2.4 Arbirtration in CAN

Arbitration is the process used in CAN protocol to decide which message has priority access to the bus when several nodes try to transmit simultaneously. Identification numbers and CAN message priority levels are crucial to the arbitration procedure.

Role of Identifiers:

1. **Unique Message Identification:**

   - The identifiers, or unique values supplied to each message, allow CAN messages to be uniquely identified.
   - While extended identifiers are 29 bits long, standard identifiers are only 11 bits long.
   - The identifier aids nodes in differentiating between various bus messages.

2. **Priority Determined by Identifier:**

   - In the context of the CAN, a higher priority is implied by lower numerical identifier values.
   - When there is bus access, messages with lower identifier values are prioritized.
   - The arbitration procedure utilizes the identifier values to establish message priority when several nodes try to send messages simultaneously.

2.4.1 How is the arbitration process performed?

A. Bit-wise Comparison: A node sends its identification and data onto the bus to start a message transmission. Every other bus node keeps an eye on the sent identifying bits simultaneously. Nodes continuously verify the sent bits against the bits that uniquely identify them. A bitwise comparison is made between the transmitted and matching bit on the bus as each bit is sent into the bus.
B. Non-Destructive Arbitration: Because CAN's arbitration procedure is non-destructive, messages with a lower priority won't affect those with a higher priority. A transmitting node will immediately stop transmission if it detects during arbitration a difference between the bit being broadcast and the bit on the bus (i.e., another node is sending a dominant bit that indicates a lower identifier value).

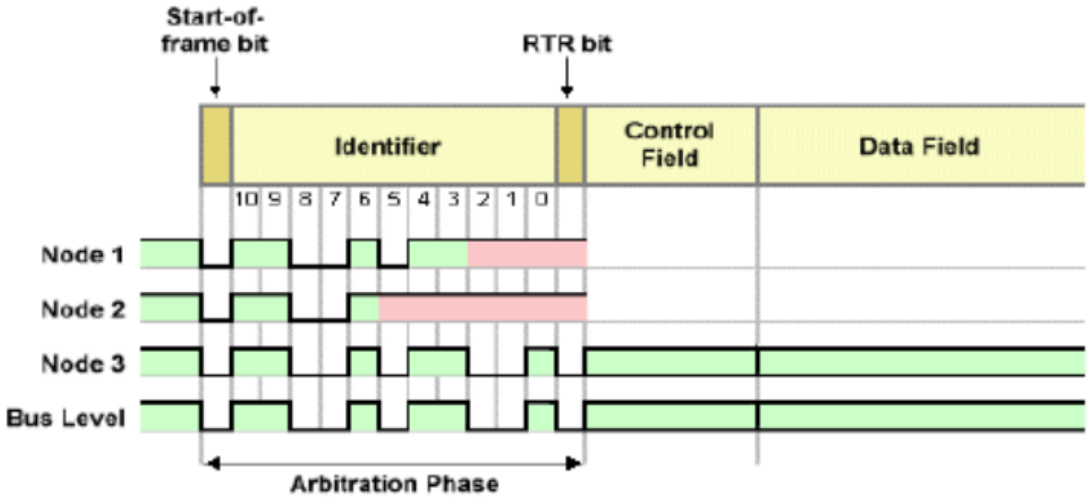


Figure 2.5: Example of Arbitration.[12]

In the following example in figure 2.5, all three nodes detect the bus as idle and decide to transmit their messages simultaneously. Nodes 1,2 and 3 transmit their identifiers bit by bit onto the bus. The bits from the 10^{th} bit up to the 3^{rd} bit are the same on all three nodes. On the 2^{nd} bit, Node 3 has a dominant bit (0) while Nodes 1 and 2 have a recessive bit (1). Per the CAN arbitration rules, dominant bits take precedence over recessive bits, so node 3 wins the arbitration and starts transmitting on the bus. Nodes 1 and 2 both detect that their transmitted bits differ from the bus level, indicating that a higher priority message is being sent on the bus. They both stop their transmission and become receivers on the bus. After node 3 Successfully transmits its message, the bus becomes idle again, allowing subsequent nodes to reattempt transmission.

2.5 CAN FD and CAN XL

Classical CAN has two main issues: Bit rate, which can reach a maximum of 1Mb/s, and its payload length, which can go up to 8B. To overcome those two main issues, Bosh proposed the CAN Flexible Data Rate (FD), which became part of ISO 11898 in 2015. CAN FD can reach bit rates up to 8Mbit/s with appropriate transceivers and cabling and a payload length of up to 64B. CAN also proposed CAN eXtra Large (XL) in CAN in Automation (CiA), which has been active since 2019. CAN XL can reach a bit rate of up to 10Mb/s and a payload of up to 2048B.

2.5.1 Why are CAN FD and XL faster?

In classical CAN, all bits are sent at the same rate, while both FD and XL are based on the **"Bit Rate Switching(BRS)"** to achieve a higher bit rate.

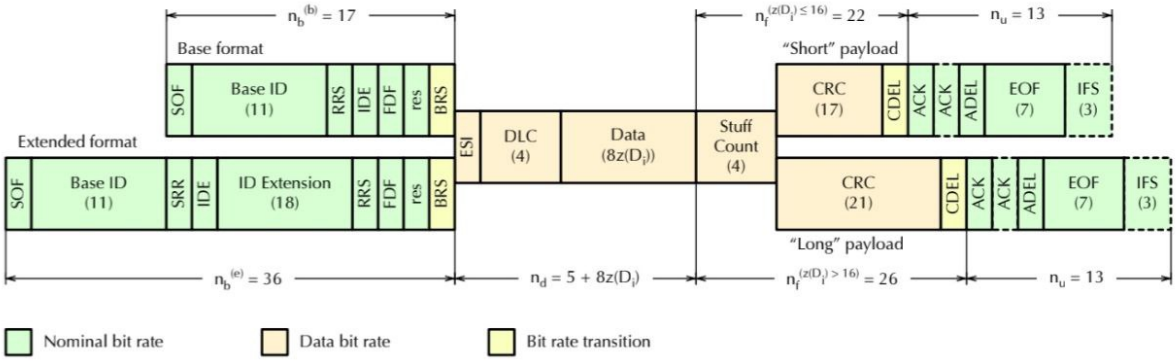


Figure 2.6: CAN FD Frame.[13]

As seen in figure 2.6, during arbitration, CAN FD works at the same nominal bit rate(up to 1Mbit/s) as CAN. However, during the Data transmission phase, CAN FD can switch its Data Bit Rate and achieve higher rates, thus transmitting the data at a higher rate. This is favourable when trying to transmit large data payloads as it reduces the time required for data transmission.

Chapter 3

Simscape Model Description

3.1 Overview

The present chapter provides a thorough overview of the Simscape model created to simulate CAN communications and examine possible vulnerabilities associated with CAN injection attacks in the context of BEVs. Simscape is primarily used to enhance the Simulink environment for modelling "multidomain" simulations, or those in which multiple domains (mechanical, electric, hydraulic, pneumatic, etc.) may be represented inside a single simulation model, and their interactions can be evaluated. Because of the multiple interactions between the mechanical and electrical domains in our particular situation, using Simscape is beneficial and efficient. Indeed, Simscape enables the creation of physical component models based on physical linkages that work in tandem with block diagrams. This model's primary goals are simulating a BEV system with different driver cycle sources, simulating the behaviour of CAN communication protocols, and evaluating the network's vulnerability to malicious injection attacks. This chapter will explore the architecture aspects, parameterization, simulation setup, and validation techniques. Additionally, integrating attack scenarios to evaluate the CAN network's vulnerabilities in a controlled virtual environment will be covered. The simulation model not only facilitates the emulation of CAN message transmissions but also enables the assessment of the impact and detection of simulated attacks, thus providing valuable insights into fortifying the security measures of CAN-based systems in BEVs. This model was created with an eye toward potential developments in CAN network-specific cybersecurity research in the future. Its simulation framework and architecture are designed to be scalable and adaptable to increasingly complex assault and defence scenarios. This fundamental model is expected to serve as a cornerstone for other research projects that attempt to fully simulate, comprehend, and strengthen CAN network security against constantly changing cyber threats.

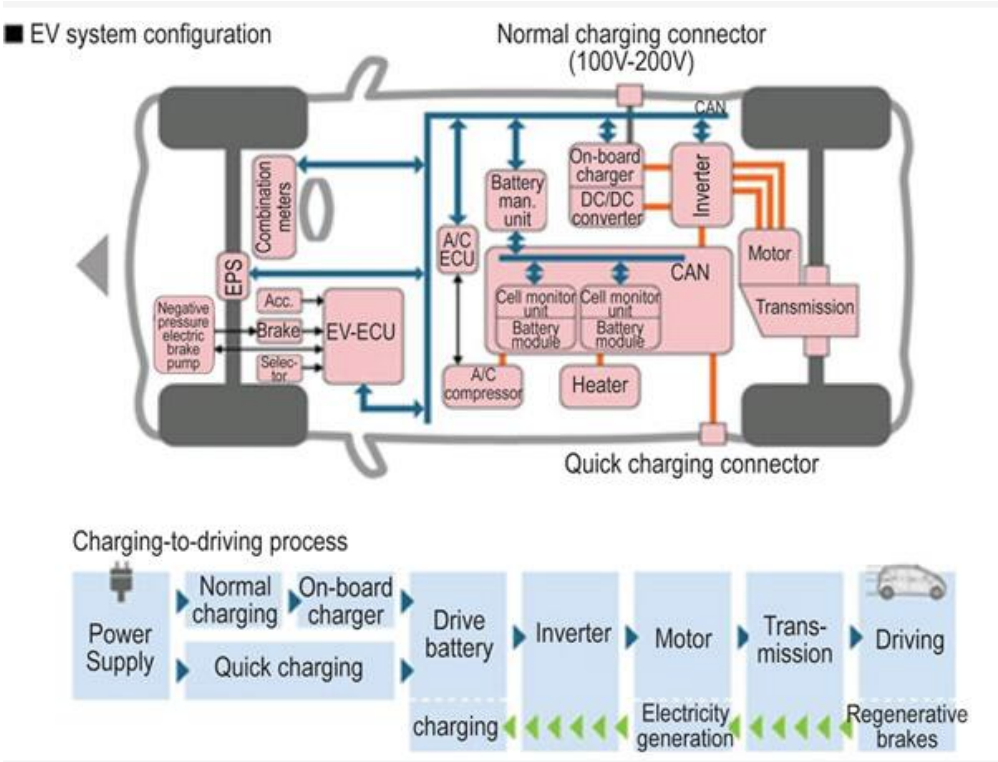


Figure 3.1: CAN Bus connection inside a vehicle [14].

3.2 Methodology

The BEV model was constructed using a systematic approach that prioritized the integration of crucial components, including the motor, chassis, battery, and control algorithm. These components were created and connected to replicate the core operations of an electric vehicle system. Starting from the chassis, the vehicle's structural backbone, the model included its physical attributes, distribution of weight, and other mechanical features critical to the vehicle's dynamics. A simplified battery representation was used to supply the motor and highlight the state of charge dynamics while avoiding going into the intricate components and complicated management systems typical of a real-world battery configuration. Furthermore, the motor—is essential for vehicle propulsion, emphasizing the model with an emphasis on its performance attributes and interactions with other components of the vehicle's powertrain. The control algorithm was created to manage critical functions in the simulated environment by optimizing and controlling the vehicle system's performance. This chapter integrates and analyses a simulated CAN inside the pre-existing BEV model architecture.

3.3 Model Architecture

The main components of an EV are:

- Battery

- Motor Controller
- Motor
- Vehicle Body(including Transmission, wheels)

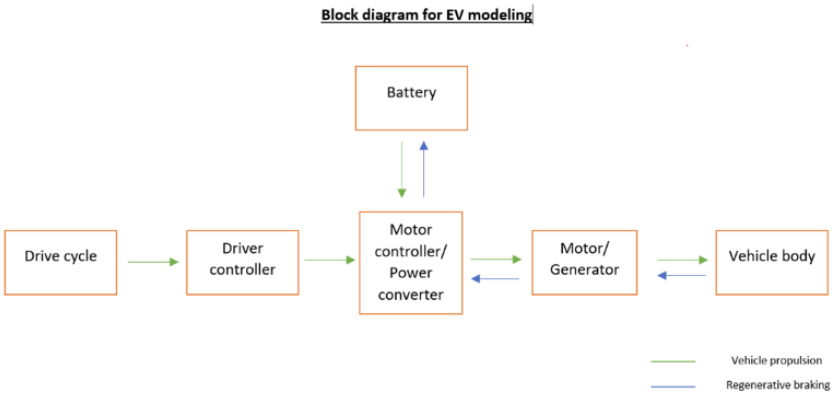


Figure 3.2: Block Diagram.

Typically, a BEV is modeled in this manner. The motor receives electrical power from a battery and transforms it into mechanical power to move the vehicle. The system must always have a reference when conducting a simulation so that it is aware of its goals. Since the system wouldn't know when to brake, accelerate, or stop the car without a reference drive cycle, it would be impossible to assess the overall system's performance because we would never know how well everything is operating.

3.3.1 Vehicle Body

The vehicle body subsystem has a main component, the "Vehicle Body block", representing a two-axle vehicle body in longitudinal motion. It doesn't consider lateral forces and represents a two-axle vehicle body in longitudinal motion. The block accounts for body mass, aerodynamic drag, road incline, and weight distribution between axles due to acceleration and road profile. The vehicle can have the same or a different number of wheels on each axle. Optionally include pitch and suspension dynamics or additional variable mass and inertia [15].

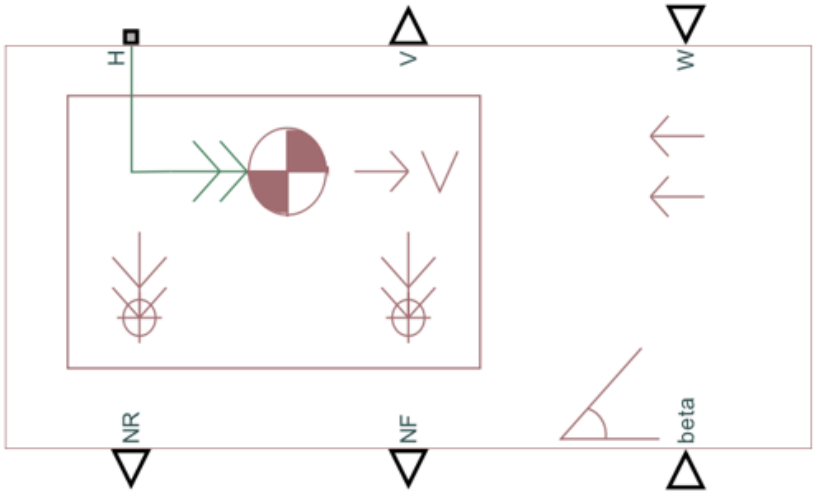


Figure 3.3: Vehicle Body block in Matlab [15].

The different ports of the block indicate:

- V: output port, which indicates the vehicle speed. Triangle-associated means that we are dealing with a physical signal, so as already said, a PS-Simulink converter will be necessary to link this port with a Scope.
- NR: Physical signal output port indicates normal loads acting on rear wheels.
- NF: Physical signal output port indicates normal loads acting on Front wheels.
- H: represents the physical connection responsible for the longitudinal motion of the vehicle. In our model, it will be the rear axle, so the rear wheels must be connected to this port.
- W: input port, represents wind velocity in m/s. In our model it is not considered, so it will be set to 0.
- β: input physical signal which models the slope of the road. In our model, it is also not considered and will be set to 0.

Inside the block, it is necessary to insert vehicle body parameters such as the Mass in Kg and the number of wheels per axle, whereas in our model, we are considering 2 per axle. Then, we have the horizontal distance from the centre of gravity (CG) from the front and rear axles and the vertical distance from the ground. Gravitational acceleration should also be provided. Drag is an optional input, and we considered it for our model. We used the Tesla Model 3 parameters to parameterize the model, which are available online.

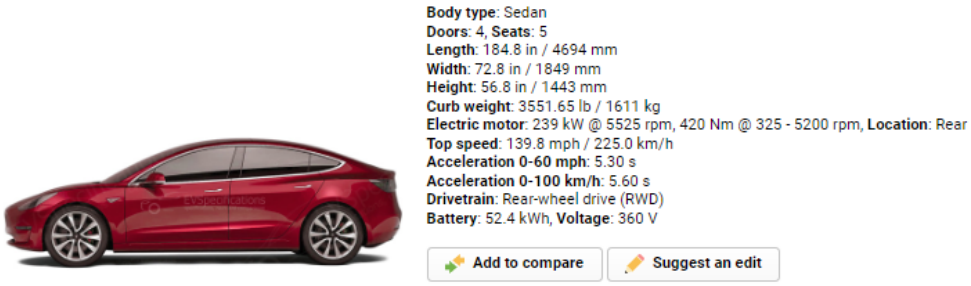


Figure 3.4: Tesla Model 3 Specifications [16].

Tire Specifications (Factory)

Tire Size	Location	Size
18"	Front/Rear	P235/45R18
19"	Front/Rear	P235/40R19
20"	Front/Rear	P235/35R20

Figure 3.5: Tesla Model 3 Tire Specifications [16].

The drag coefficient of the Model 3 is 0.23 and the frontal area is 2.22 m^2. From the above Data, we can derive the parameters needed to insert inside the vehicle body block, which are summarized in the following table:

Mass	1611 kg
Frontal Area	2.22 m^2
Drag Coefficient	0.23
Tire radius	0.3345(calculated from tire specs)
Coefficient of rolling resistance	0.01

Table 3.1: Main Parameters Inserted in the Vehicle Body Model.

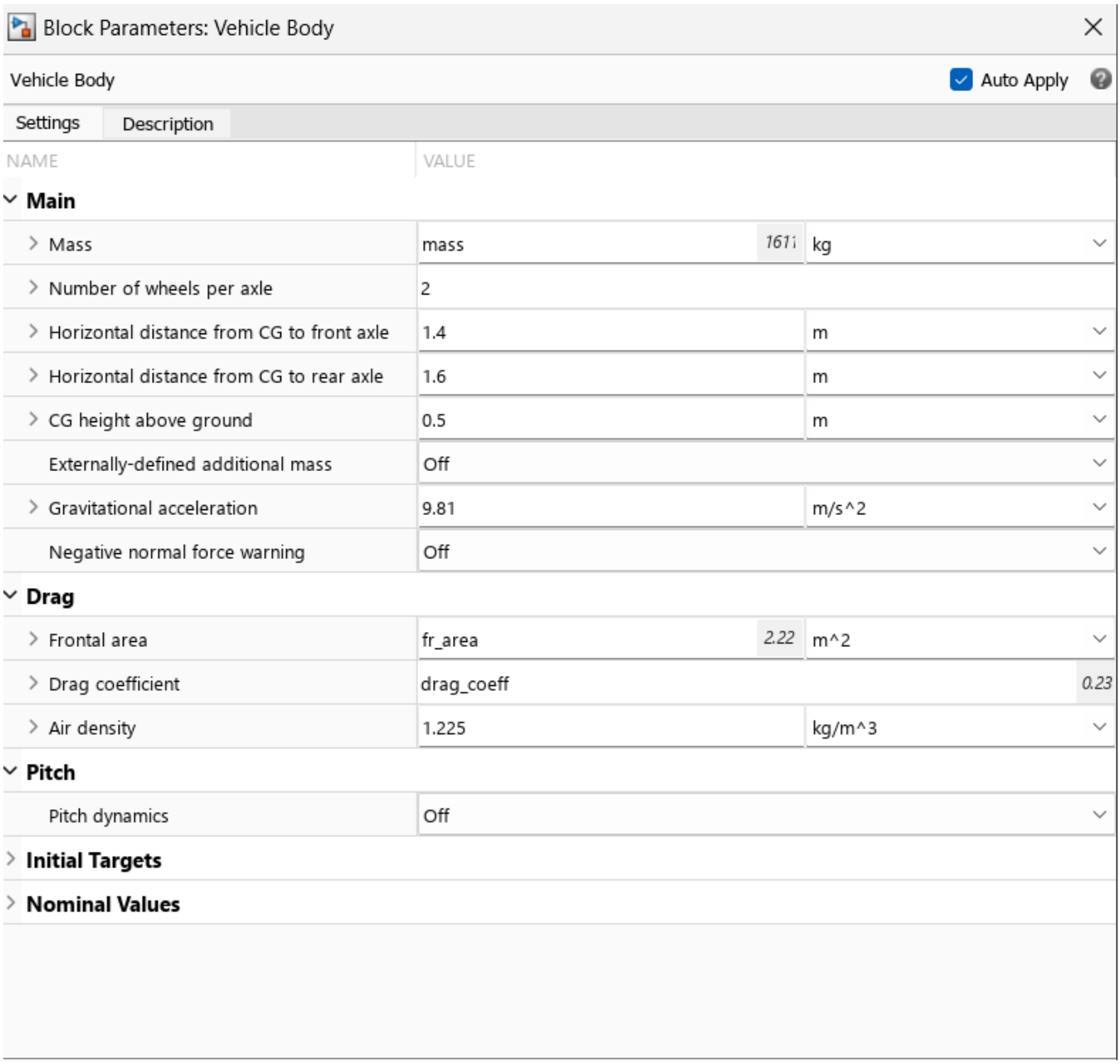


Figure 3.6: Input values for the vehicle body [15].

3.3.2 Tires

After creating the vehicle body, we should connect to it the four wheels, two on the front and two on the rear.

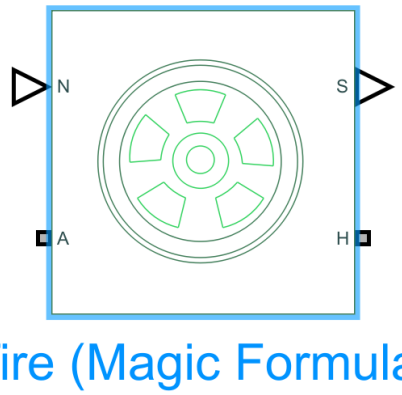


Figure 3.7: Tire Simscape Block [17].

Tyre in Figure 3.7 Represents a highway tyre's longitudinal behaviour characterized by the Magic Formula. Optionally, the effects of tyre inertia, compliance, and rolling resistance can be included. The effects of scaling coefficients can optionally be included when parameterized by Load-dependent Magic Formula coefficients or the effective rolling radius model is Load and velocity-dependent Magic Formula. It has four ports:

- A: Is the wheel axle's mechanical rotational conserving port. The two rear wheels will be connected between them through this one, and we will connect the motor axle to them. Also, the front wheels will be connected between them through the A port.

- H: Is the wheel hub's mechanical translational conserving port through which the thrust developed by the tire is applied to the vehicle. All four wheels should be connected between them through this port.

- N: Is a physical signal input port that applies the normal force acting on the tyre. The force is considered positive if it acts downwards.

- S: is a physical signal output port that reports the tyre slip.

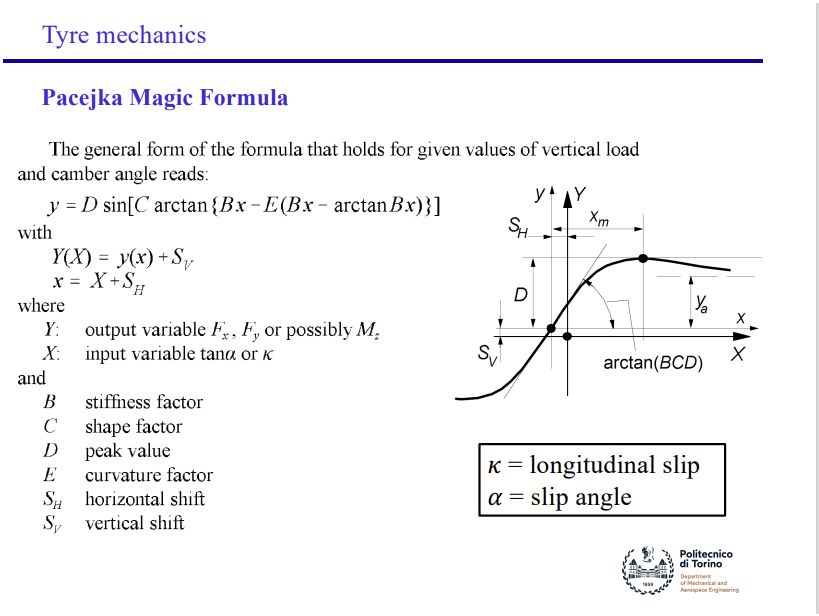


Figure 3.8: Tire Magic Formula [18].

Inside the block of tyres in Figure 3.7, we should insert the parameters needed, such as the vertical load and the coefficients for the magical formula of the tyre. The Tire (Magic Formula) block represents a tire with longitudinal behaviour given by the Magic Formula. Figure 3.8 presents an empirical equation based on four fitting coefficients.

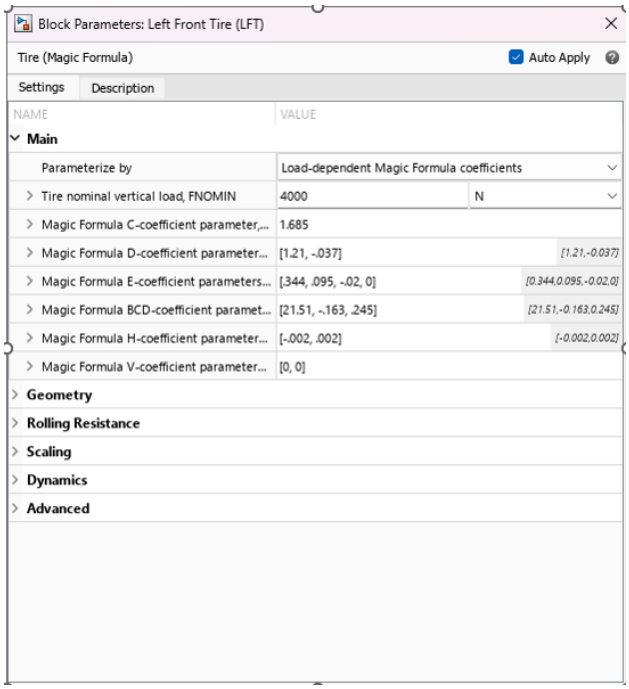


Figure 3.9: Tire Magic Formula parametrization.

Now, after defining the vehicle body and the tyres' model, we can connect all of them as shown in Figure 3.10 : Port 1 represents the connection to the transmission axle, and

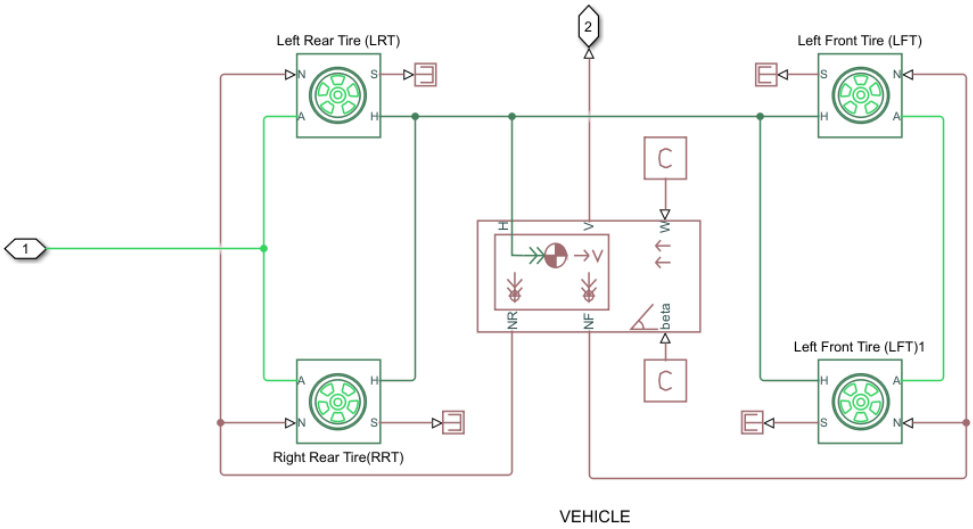


Figure 3.10: Vehicle body and tyres all connected.

connection port 2 represents the velocity output port in m/s.

3.3.3 Transmission

The transmission subsystem for this BEV is modeled as a gear pair, which reduces the speed of the electric motor. It is an automatic single-speed reduction gear, and it has a value=9.

Transmission
Information about the transmission of the model.

Type of transmission Information about the type of transmission used by the vehicle. Generally, EVs are equipped with one-speed automatic units.	Automatic single-speed reduction gear
Drivetrain Information about the type of drivetrain (the system that delivers power to the driving wheels).	Rear-wheel drive (RWD)
Axle ratio The axle ratio a.k.a. final drive ratio of the electric motor. It shows the number of revolutions of the driveshaft needed to spin the axle one complete turn.	9

Figure 3.11: Transmission Gear [16].

In Simscape, we can find the block as a "Gear Box" or "Simple Gear", where both have one input and one output indicating the input and the output shaft, respectively.

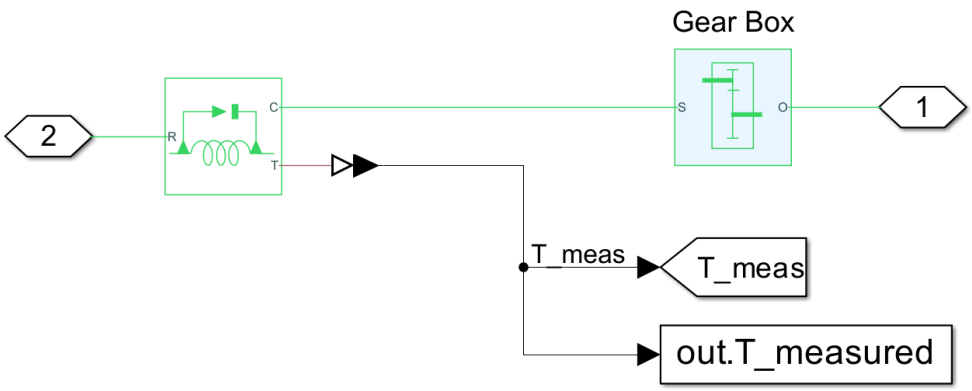


Figure 3.12: Gear Box Subsystem.

In Figure 3.12, we can see the input port (2) shaft that will be the motor shaft while from the port (1) the output shaft. The block before the gearbox is an Ideal Torque Sensor block, which is "a device that converts a variable passing through the sensor into a control signal proportional to the torque with a specified coefficient of proportionality. The sensor is ideal since it does not account for inertia, friction, delays, energy consumption, etc. Connections R and C are mechanical rotational conserving ports connecting the sensor to the line whose torque is being monitored. Connection T is a physical signal port that outputs the measurement result. The sensor's positive direction is from port R to port C" [19].

3.3.4 Electric Motor and Controller

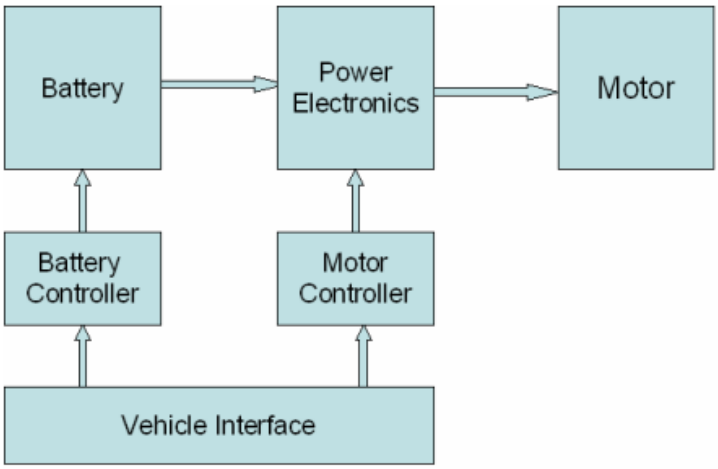


Figure 3.13: Electric Vehicle Drive Train [20].

Figure 3.13 showcases the overall structure of an Electric Vehicle Drive Train. The drive train draws energy from the battery while in motion. Moreover, the drive train is capable of charging the battery by using the motor as a generator during regeneration, which happens

when the vehicle brakes. Generally, the battery is composed of lithium-ion cells and provides power with over 300 volts and a high current. Key battery metrics are monitored by a battery controller, which also manages the battery pack. The power electronics unit converts the DC battery voltage into a three-phase AC voltage with the correct frequency and voltage to enable the motor to operate at the desired speed and torque. An AC Induction Motor (IM) or a Permanent Magnet Synchronous Motor (PMSM) is typically used as the AC motor, which can provide either braking or acceleration torque for both rotational directions. When the brakes are applied, the motor reverses the directions of torque and current, known as regeneration mode. This mode not only helps recharge the battery but also provides torque for vehicle braking. To invert the DC voltage, the motor controller generates an AC voltage at the appropriate frequency and voltage. Normally, a 10-20 KHz pulse width modulated AC voltage is used for the motor, and the voltage and frequency are adjusted to provide the appropriate motor speed and magnetic field values [20]. The Vehicle Interface provides an interface with the controls and sensors at the vehicle level and communicates with the Battery Controller, Motor Controller, and both. A CAN communication system is used for communication among the individual units.

3.4 The Way to the Final Powertrain Model

3.4.1 First Powertrain Model

In the first trial in this thesis, a DC permanent magnet motor was used to model the power train and get more comfortable with Simscape in the simulation model presented below. This model uses Simscape's "DC motor" block, and for the control, it uses a Pulse Width Modulation (PWM) controller using an H-bridge driver.

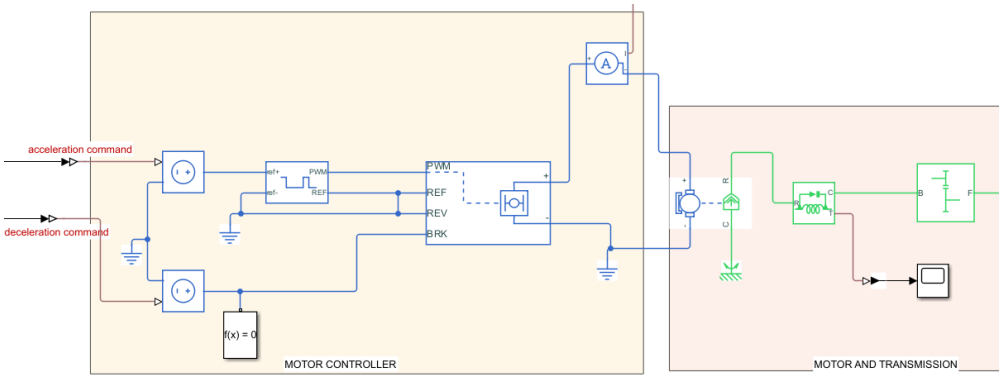


Figure 3.14: Powertrain with DC motor.

Electric Motor is modelled as a DC motor, whose positive and negative terminals are linked to the corresponding ones of an H-Bridge Simscape block. The H-bridge receives input from a PWM controller Simscape block for the acceleration phases and receives directly from the driver braking input. PWM receives input from the throttle pedal coming from the driver, which is then converted into a current signal through an ideal current source block.

To obtain the acceleration and the deceleration commands, the "longitudinal Driver" block represents a Proportional Integral (PI) controller, and its proportional and integral gains are to be tuned. If the accelerator signal is given, the motor controller converts that signal into the appropriate voltage needed to supply to the motor (refer to H-bridge configuration above), which likewise happens with the brake signal. In addition, when the brake signal is given, this is a power converter with regenerative braking, which means that it takes current back from the motor (which is acting like a generator), controls it appropriately to not destroy the battery with a sudden in surge of current, and transfers it to the battery to charge it.

Although this powertrain model provides good results in tracking the reference driving cycle provided, it was not considered in this thesis since this type of motor is not appropriate for BEV applications due to weight and efficiency considerations. We decided to model the motor as a PMSM motor with Field Oriented Control (FOC) control to account for a more realistic model. The choice to utilize a PMSM for simulating a BEV's drivetrain arises from several beneficial features that these motors provide, such as high efficiency, as Interio PMSM motors exhibit high energy efficiency levels, significantly reducing energy losses during operation. This efficiency translates into longer driving ranges for BEVs on a single charge, enhancing electric vehicles' overall performance and practicality. PMSM consists of a three-phase stator and a permanent magnet rotor. There is no need for slip rings and brushes since the permanent magnet in the rotor provides the magnetic field.

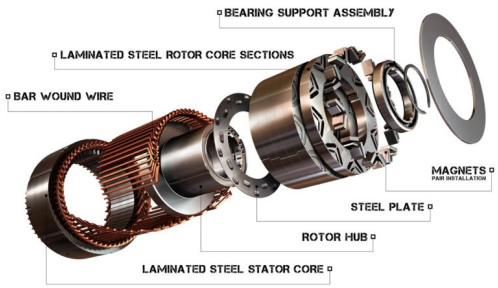


Figure 3.15: Construction of inset magnet PM motors[21].

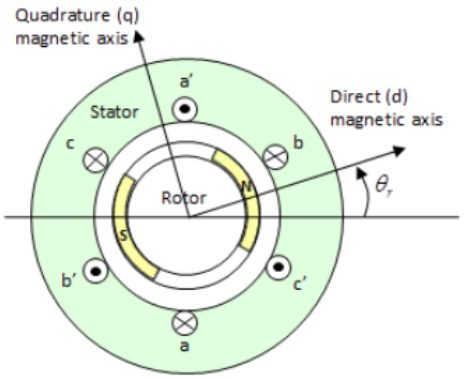


Figure 3.16: Motor construction with a single pole-pair on the rotor[22].

3.4.2 Second Powertrain Model

Mathematical Background of PMSM

Accurate mathematical models are essential to designing and controlling PMSMs effectively. The mathematical representation of a PMSM typically involves equations that describe the relationships between currents, voltages, magnetic flux, and torque. Different modelling techniques, such as modelling in the ABC frame and d-q frame, are employed to simplify the analysis of the motor's behaviour in a rotating reference frame. The Clark and Park transformations enable the separation of the direct-axis (d-axis) and quadrature-axis (q-axis) components, facilitating control strategies that decouple torque and flux control.

PMSM Modeling in *abc* frame

In the ABC frame, the rotor is modelled simply as the magnetic equivalent salient rotor. Since there is no excitation circuit in the roto of PMSM (no rotor winding), we only consider the stator winding equations. The 3-phase voltage equations (3.1,3.2,3.3) are reported below [23]:

$$v_a = R_s i_a + \frac{d\lambda_a}{dt} \tag{3.1}$$

$$v_b = R_s i_b + \frac{d\lambda_b}{dt} \tag{3.2}$$

$$v_c = R_s i_c + \frac{d\lambda_c}{dt} \tag{3.3}$$

The flux linkages, due to stator currents, and PM flux on the rotor for each phase are given as all the equations are referred to the stator side:

$$\lambda_a = L_{aa} i_a + L_{ab} i_b + L_{ac} ic + \psi_{aPM} \tag{3.4}$$

$$\lambda_b = L_{ba} i_a + L_{bb} i_b + L_{bc} i_c + \psi_{bPM} \tag{3.5}$$

$$\lambda_c = L_{ca} i_c + L_{cb} i_b + L_{cc} i_c + \psi_{cPM} \tag{3.6}$$

Where L_{aa}, L_{bb}, L_{cc} are the stator self-inductances which consist of leakage inductance and the magnetizing inductance.
The flux caused by PM varies with the angular position of the rotor for each phase. The self-inductance of a PMSM may differ depending on the angular position of the rotor because

the effective air gap may vary with the position of the rotor. The self-inductance varies sinusoidally with the rotor. The self-inductances for each phase are reported below:

$$L_{aa} = L_l + L_A + L_B \cos(2\theta_r) \tag{3.7}$$

$$L_{bb} = L_l + L_A + L_B \cos\left(2\left(\theta - \frac{2\pi}{3}\right)\right) \tag{3.8}$$

$$L_{cc} = L_l + L_A + L_B \cos\left(2\left(\theta + \frac{2\pi}{3}\right)\right) \tag{3.9}$$

Where L_l, L_A, L_B represents the leakage inductance, the average value of the magnetizing inductance, and the variation in the value of the magnetizing inductance. The mutual inductances between the stator windings also vary sinusoidally concerning the angle θ_r.

With these inductances, the stator inductance $\mathbf{L_s}$ is given by:

$$\begin{bmatrix} L_l + L_A + L_B\cos(2\theta_r) & -\frac{1}{2}L_A - L_B\cos\left(2\left(\theta - \frac{\pi}{3}\right)\right) & -\frac{1}{2}L_A - L_B\cos\left(2\left(\theta + \frac{\pi}{3}\right)\right) \\ -\frac{1}{2}L_A - L_B\cos\left(2\left(\theta - \frac{\pi}{3}\right)\right) & L_l + L_A + L_B\cos\left(2\left(\theta - \frac{2\pi}{3}\right)\right) & -\frac{1}{2}L_A - L_B\cos(2\theta_r) \\ -\frac{1}{2}L_A - L_B\cos\left(2\left(\theta + \frac{\pi}{3}\right)\right) & -\frac{1}{2}L_A - L_B\cos(2\theta_r) & L_l + L_A + L_B\cos\left(2\left(\theta + \frac{2\pi}{3}\right)\right) \end{bmatrix} \tag{3.10}$$

The flux in the form of a matrix is expressed as:

$$\boldsymbol{\lambda_{abc}} = \boldsymbol{L_s i_{abc}} + \boldsymbol{\psi_{PM}} \tag{3.11}$$

Reference Frame Transformations

Their voltage and current equations usually describe the behavior of three-phase machines. The coefficients of the differential equations describing their behaviour are time-dependent. The mathematical modelling of such a system is usually very complex since the flux linkages, induced voltages and currents change continuously when the circuit is in relative motion. For such complex analyses of electrical machines, mathematical transformations are often used to decouple variables and solve equations with time-varying quantities by relating all variables to a rotating reference frame. The two transformations are:

- **Clarke Transformation** -This transformation converts three-phase quantities-abc into two-phase quadrature quantities-$\alpha\beta$, or stationary reference frame.

$$\begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}\begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} \tag{3.12}$$

- **Park Transformation** -This transformation converts vectors from two-phase orthogonal stationary reference frame-$\alpha\beta$ into orthogonal rotating reference frame- dq. The transformation is performed considering the rotor flux angle θ_r. The definition of the space vector by three-phase components is as follows:

$$f_{abc} = \frac{3}{2}(f_a + e^{j\frac{2\pi}{3}} f_b + e^{j\frac{4\pi}{3}} f_c) \tag{3.13}$$

PMSM modelling in dq Frame:

The three-phase voltage equations of the stator were given as follows:

$$\boldsymbol{v_{abc}} = \boldsymbol{R_s i_{abc}} + \frac{d\boldsymbol{\lambda_{abc}}}{dt} \tag{3.14}$$

The transformation matrix K for transforming the abc-frame into the dq-frame, and the inverse matrix K^{-1}, for the reverse transformation are defined:

$$K(\theta_r) = \frac{2}{3}\begin{bmatrix} \cos\theta_r & \cos\left(2\left(\theta_r - \frac{2\pi}{3}\right)\right) & \cos\left(2\left(\theta_r + \frac{2\pi}{3}\right)\right) \\ -\sin\theta_r & -\sin\left(2\left(\theta_r - \frac{2\pi}{3}\right)\right) & -\sin\left(2\left(\theta_r + \frac{2\pi}{3}\right)\right) \end{bmatrix} \tag{3.15}$$

$$K^{-1}(\theta_r) = \frac{2}{3}\begin{bmatrix} \cos\theta_r & -\sin\theta_r \\ \cos\left(2\left(\theta_r - \frac{2\pi}{3}\right)\right) & -\sin\left(2\left(\theta_r - \frac{2\pi}{3}\right)\right) \\ \cos\left(2\left(\theta_r + \frac{2\pi}{3}\right)\right) & -\sin\left(2\left(\theta_r + \frac{2\pi}{3}\right)\right) \end{bmatrix} \tag{3.16}$$

Multiplying on both sides the Eq-3.14 with the K matrix, we have [23]:

$$v_{dq} = R_s i_{dq} + \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix}\lambda_{dq} + \frac{d\lambda_{dq}}{dt} \tag{3.17}$$

Since this is the complex space vector of the voltage, it can be decomposed into dq components as:

$$v_d = R_s i_d - \omega\lambda_q + \frac{d\lambda_d}{dt} \tag{3.18}$$

$$v_q = R_s i_q + \omega\lambda_d + \frac{d\lambda_q}{dt} \tag{3.19}$$

Due to the rotation frame, we have the speed component in the voltage equations, and as seen, the dq fluxes are coupled. The same steps can be followed to transform the flux linkage equations from the ABC frame to the dq frame, obtaining the following flux equation[23]:

$$\boldsymbol{\lambda_{dq}} = KL_sK^{-1}i_{dq} + \psi_{dq-PM} \tag{3.20}$$

With some manipulation of the above equation and considering that the PM is aligned with the d-axis, we can obtain the inductances Ld and Lq as:

$$L_d = L_l + \frac{3}{2}(L_A - L_B);\ L_q = L_l + \frac{3}{2}(L_A + L_B) \tag{3.21}$$

The above transformations are valid when the d-axis is oriented with the rotor flux. The electromagnetic torque equation is:

$$T_e = \frac{3}{2} n_p \left(\psi_{PM} i_q + (L_d - L_q)\, i_d i_q \right) \tag{3.22}$$

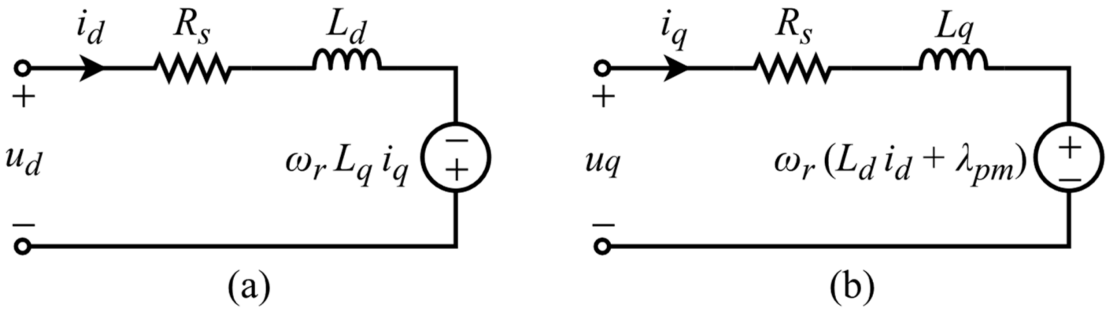


Figure 3.17: Equivalent $d-q$ axis circuits of PMSM [24].

Now that we have described the mathematical model of the PMSM, we can move on to see how it was implemented in the simscape second powertrain model:

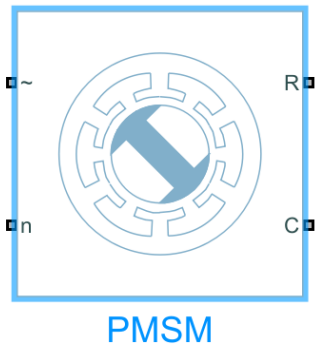


Figure 3.18: PMSM block in Simscape Library[22].

The above block present in Figure 3.18 is the PMSM block in Simscape, and it has four ports:

- ∼: Represent the 3-phase current abc.
- n: Electrical conserving port associated with the neutral phase.
- R: Mechanical rotational conserving port associated with the motor rotor.
- C: Mechanical rotational conserving port associated with the motor case.

As an input, it receives the 3-phase current ABC and the neutral phase, while from the output port, we can connect the rotating shaft to the "R" port and the conserving mechanical part of the motor case to the "C" port.

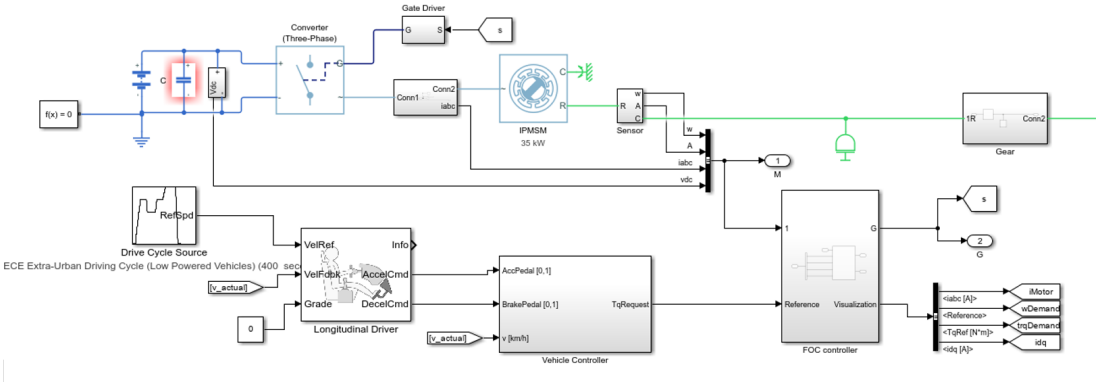


Figure 3.19: Second Powertrain Model.

Figure 3.19, the entire power train model, starting from the motor block connected to the 3-phase current abc. The gate driver controls the gates of the converter depending on the value of s, which comes from the FOC controller. The battery is connected to a capacitor to the converter to remove the voltage ripples. In the vehicle controller block, we have the control algorithm that translates the acceleration and deceleration commands to a Torque request that is then fed to the FOC controller by turn. The FOC controller in Figure 3.20 also inputs the 3 phase currents ABC, the rotor speed, the rotor angle, and the DC link voltage. As an output, it provides the command that controls the six pulse waveforms that determine switching behaviour in the attached power converter and a set of signals such as idq, torque limit, and torque demand to the motor.

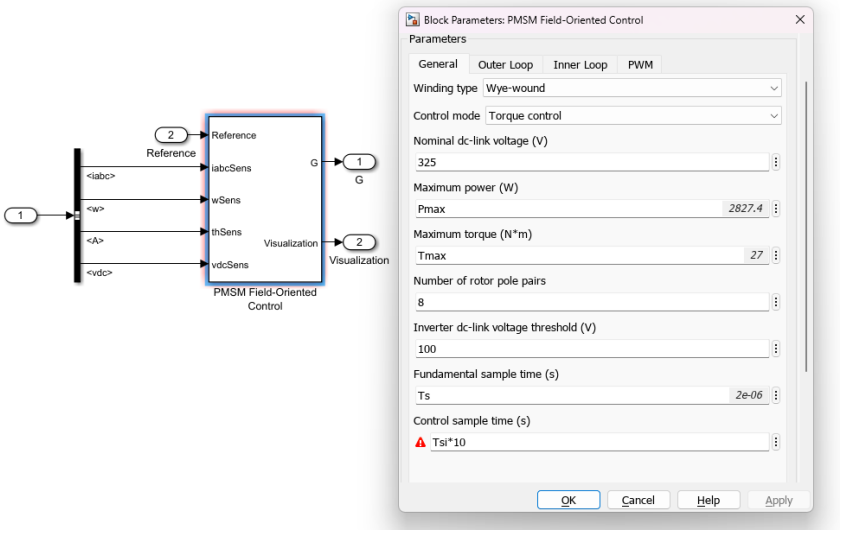


Figure 3.20: FOC in Simscape

Due to the complexity of this powertrain model, it had a high computational cost, and the simulations were very slow. They wouldn't let us simulate the CAN communications and injections well, which is the primary purpose of this thesis, and for that, a simpler powertrain model was needed.

3.4.3 Final Powertrain Model

The last model to model the electric motor was the "Motor and Drive" block, which models the motor and the drive at the system level. The Motor and Drive (System Level) block represents a generic motor and drives with closed-loop torque control. This block is used to model a brushless motor (such as a PMSM) and drive, or, more generally, traction and actuation systems implemented using a wide range of motor types [25].

It also allows both simplified and tabulated definitions of electrical losses. The default, simplified behaviour is to model the losses as the sum of the following four terms:

- A series resistance between the DC power supply and the motor drive.
- Fixed losses independent of torque and speed, P0. Use this to account for fixed converter losses.
- A torque-dependent electrical loss $k_t\tau^2$, where τ is the torque and k_t is a constant. This represents ohmic losses in the copper windings.
- A speed-dependent electrical loss $k_w\omega^2$, where ω is the speed and k_w is a constant. This represents iron losses due to eddy currents.

In simscape we can find the block by searching "motor and drive" [25].

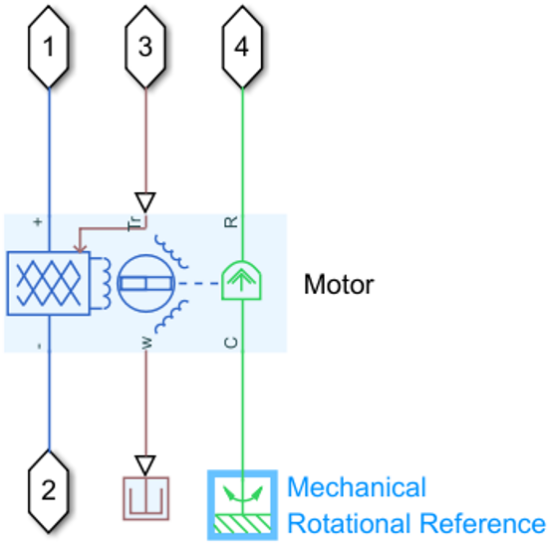


Figure 3.21: Motor and Drive in Simscape

The block present in Figure 3.21 has 6 ports:

- Tr: Reference Torque Demand
- W: Mechanical Speed Output
- +: Positive electrical DC supply
- -: Negative electrical DC supply
- C: Mechanical rotational conserving port associated with the motor case.
- R: Mechanical rotational conserving port associated with the motor rotor.

To the Tr, we connect the Torque Reference that we would like to obtain at the motor. To the + and the – pols, we connect the positive and negative poles of the battery. A mechanical reference port is connected to port C, while port R is the output port from which we connect the shaft to the transmission gear and then to the vehicle Body. Although this motor model looks simple, it provides us with accurate results and a fast simulation, as shown in the next chapter.

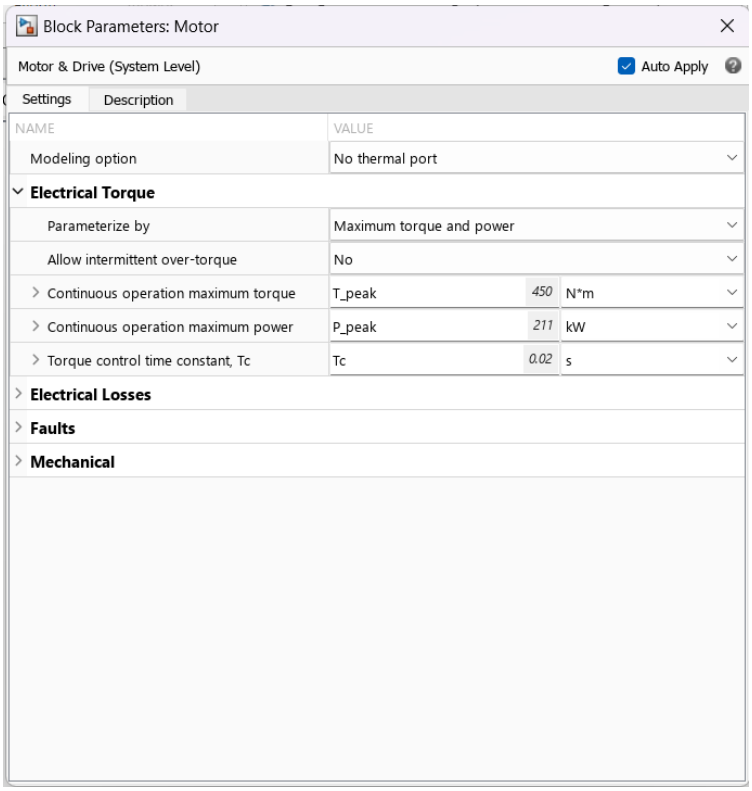


Figure 3.22: Motor and Drive Parameter Settings

3.5 Torque Request Control

To obtain the Torque Request, we should first be able to get the acceleration and the braking commands, which then will be translated into a torque request that will be sent as an input to the electric motor; here, we use the block "Longitudinal Driver" present in figure 3.23.

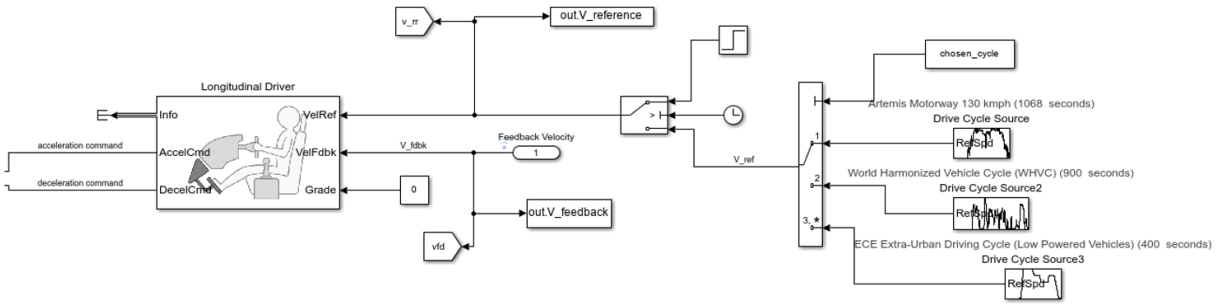


Figure 3.23: Longitudinal Driver and driving Cycles Subsystem

A longitudinal speed-tracking controller is implemented using the Longitudinal Driver block. Based on reference and feedback velocities, the block outputs normalized instructions for braking and accelerating that range from 0 to 1. The block can be used to model a driver's dynamic response or generate the commands necessary to track a longitudinal drive cycle. The controller can be modeled as a PI, Scheduled PI, and Predictive Controller. For our application, we will choose a PI controller [26]. As we can see, the block takes three inputs and three outputs. The three inputs are the reference velocity, the feedback velocity, and the road grade angle. The PI controller receives an input signal from a pre-established driving cycle in the velocity reference port "VelRef". In contrast, the "VelFDBK" port receives the actual feedback velocity of our vehicle and, according to our forward-based model, gives a throttle and brake command signal as outputs. Our application's last port, "Grade", is set to 0. As outputs, as mentioned above, the longitudinal driver block provides the acceleration and deceleration commands normalized between 0 and 1.
The most essential part of this block is the parametrization phase, a key factor for simulation stability. The Driver block requires many PI controller parameters. These latter are very affecting factors, and a careful tuning phase has been necessary to implement both the proportional and the integral gains to achieve the desired system response.

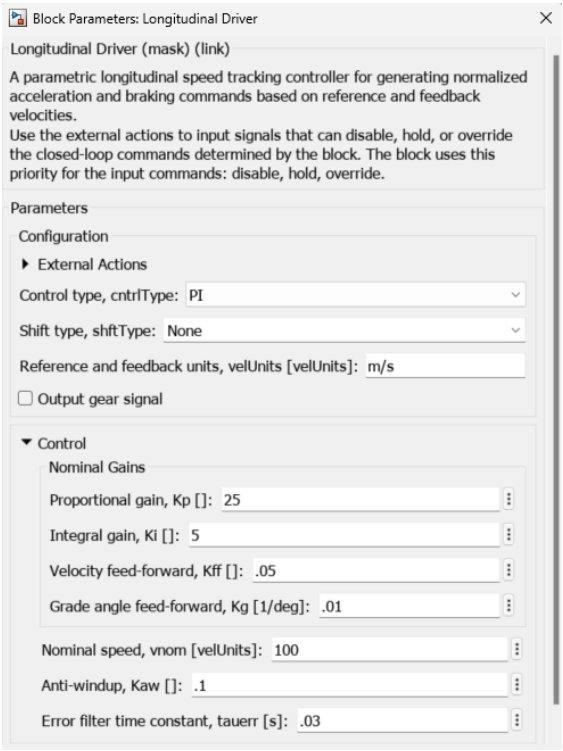


Figure 3.24: Longitudinal Driver block Parameters.

The proportional gain is represented by the Proportional Gain Kp. Based on the current error signal, this number scales the proportional component of the controller's output, which is set to a high number and may be interpreted as the driver's "aggression factor." The integral Gain is represented by the Ki parameter in a PI controller block. Based on the integrated error signal over time, this number scales the integral component of the controller's output. In Figure 3.23, we can see the different driving cycles attached to the input, and depending on which cycle the user wants to activate, it will be activated respectively. After defining how the acceleration and deceleration commands are obtained, we can move on and explain how both the acceleration and deceleration commands are translated to a Torque command that is then provided to the motor block. The motor must be able to give traction power to the vehicle. Still, we should also consider reversing the torque to negative values to recharge the battery during the braking manoeuvres.

To translate the acceleration and deceleration commands to a torque command, the normalized values of the acceleration and the deceleration are multiplied into a lookup table. Where this lookup table is a data structure that contains pre-defined values, often representing a percentage of the peak torque available from the vehicle's powertrain or actuator system, we will obtain a positive torque for acceleration. In contrast, for braking, we will obtain a negative torque. "The lookup table performs an n-dimensional interpolated table lookup, including index searches; in our case, it's a 1-dimensional interpolation. Breakpoint sets relate the input values to positions in the table. The first dimension corresponds to the top (or left) input port" [27].

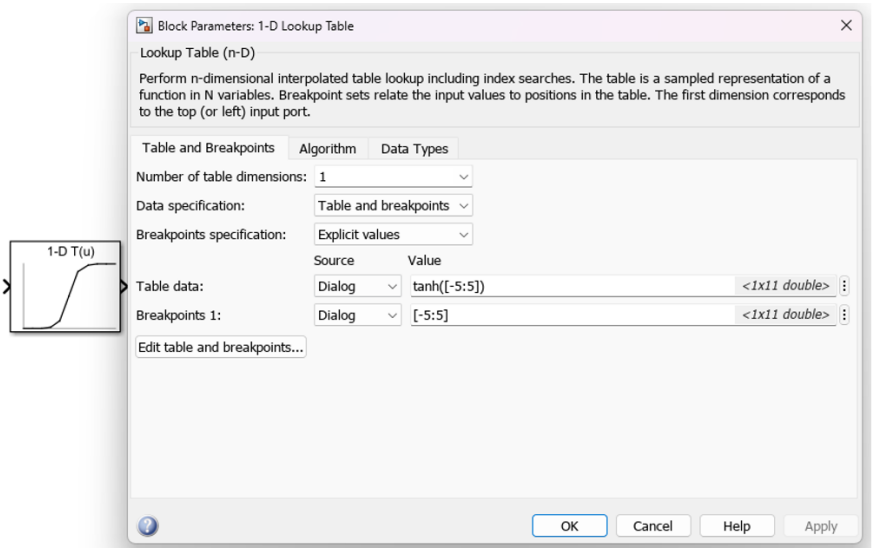


Figure 3.25: Look Up Table.

The table data represents the percentage of the maximum torque available, and it's provided as a vector. In contrast, the breakpoints represent the positions or input values along the axis of the lookup table. These breakpoints indicate where specific output values (from the table data) are applied or interpolated for corresponding input values. In our case, they represent how much the accelerating or decelerating pedal is pressed. The torque from the acceleration and deceleration tables is summed to obtain the final torque request, as seen in Figure 3.26 below.

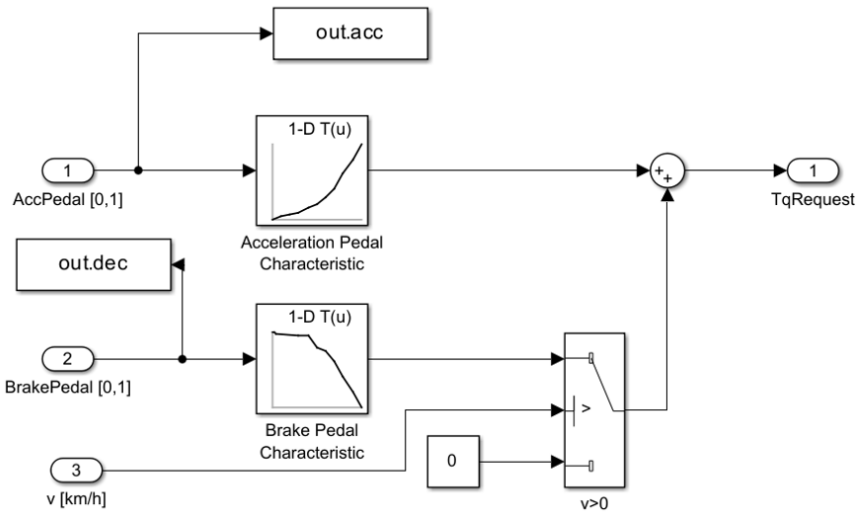


Figure 3.26: Torque Request.

3.6 Battery

The Tesla Model 3 uses a lithium-ion battery, known for its high energy density and efficiency. This battery contains 2976 cells arranged in 4 modules with a total nominal voltage of 360V. From the data of model 3 that we took, the battery pack had a nominal voltage equal to 360 volts, while its total energy capacity is equal to 52.4 kWh. From the following data, we can calculate the rated charge of the battery using the formula: Rated Capacity (Ah) = Total Energy (kWh) / Nominal Voltage (V).
For the Tesla Model 3: Rated Capacity (Ah) = $\frac{52{,}400\text{Wh}}{360\,\text{V}}$ = 145.5Ah.

To model the battery in a simscape, a "battery" block was used. This block has two ports, the positive port and the negative port. It takes several parameters as inputs and can be modelled to simulate the charging and discharging behaviour of the battery and then to measure the state of charge or simply by considering that it has an infinite battery charge capacity. In our model, the charging and discharging of the battery were considered so that we could measure the state of the charge.

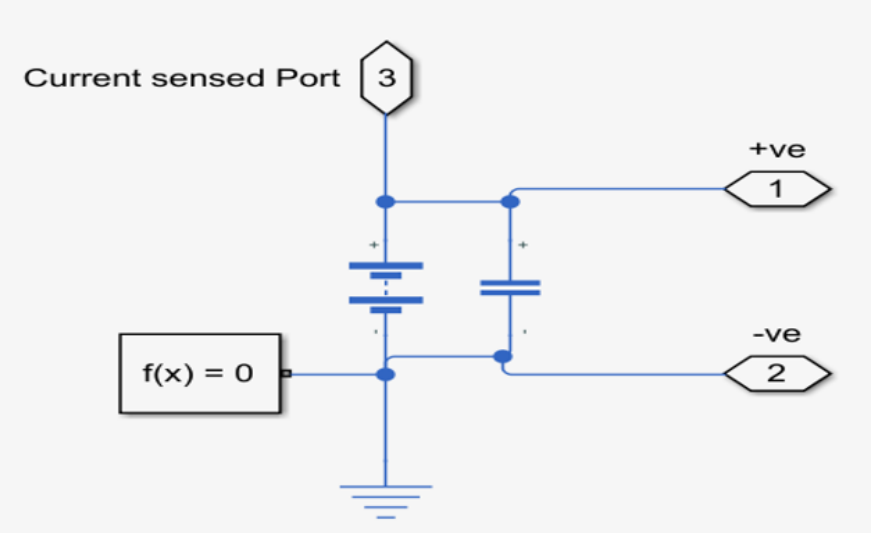


Figure 3.27: Battery in Simscape.

As seen in figure 3.27, the battery is directly connected in parallel to a capacitor to filter out the voltage ripples. The polls of the battery are connected to the polls of the motor, and to close the circuit, the downbeat poll is also connected to an electrical reference. The f(x)=0 block must be connected anywhere in the model so the simulated model can run. The current sensed port is used to sense the current of the battery that will be used to estimate the battery's state of charge. Parameters such as nominal voltage, internal resistance, Cell Capacity, Voltage V1, and Charge AH1 were all inserted as inputs to the battery block, as seen in the figure 3.28 above.

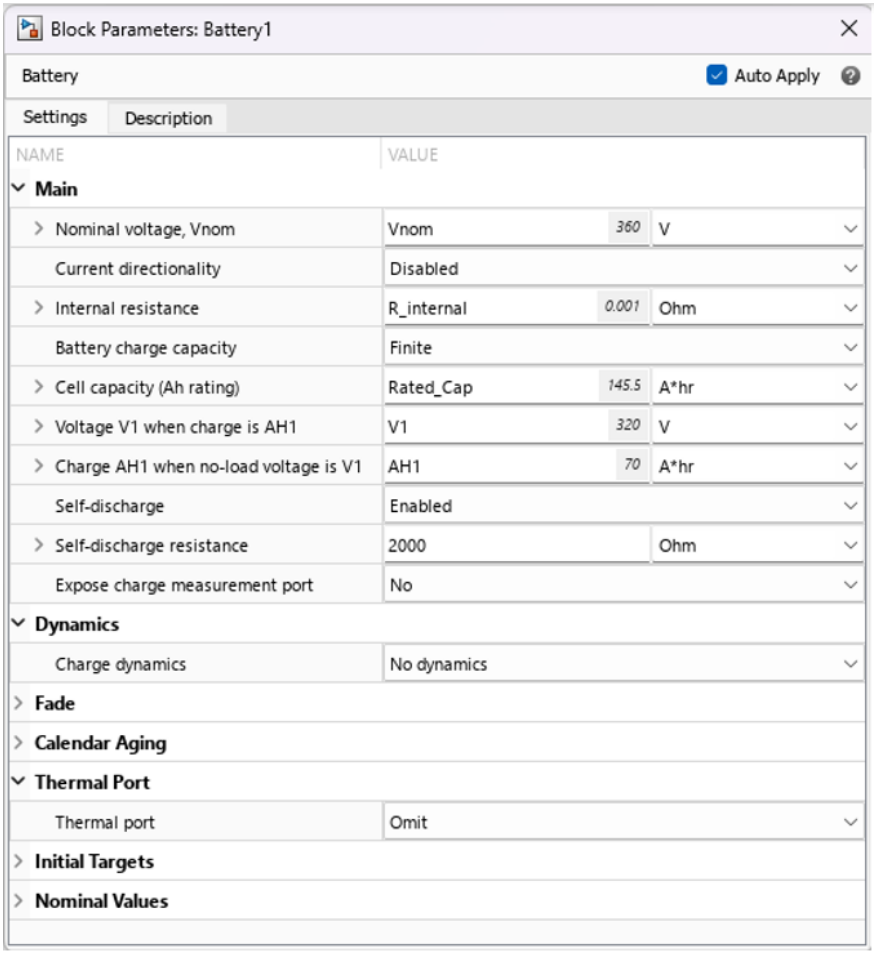


Figure 3.28: Battery Parameter settings in Simscape.

3.6.1 State Of Charge Estimation

A subsystem of various blocks was created to measure the battery's state of charge. The first block is the 'Rate Transition' block, which handles data transfer between different rates and tasks.

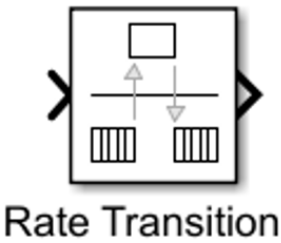


Figure 3.29: Rate Transition Block.

Next, a 'Gain' block is used to multiply or divide the signal from the rate transition block. At this point, the battery's ampere rating multiplied by 3600 (hourly seconds) will be divided by the gain block to divide the current signal. The gain block settings display this.

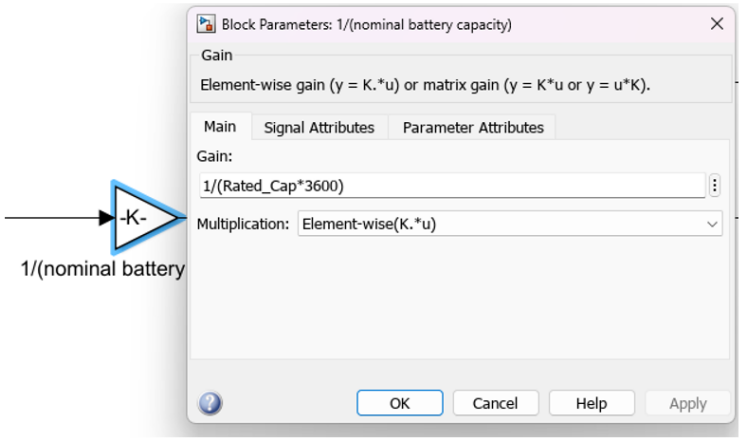


Figure 3.30: Gain Block.

Next, the current signal with respect to time is discretely integrated using a "Discrete - Time Integrator" block. The discrete time integrator block is shown in Figure 3.31.

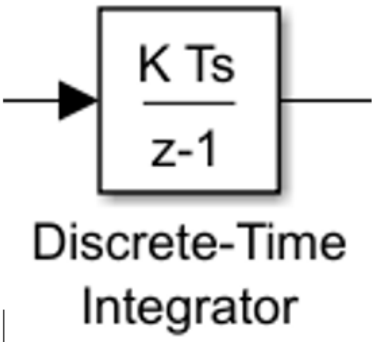


Figure 3.31: Discrete Time Integrator Block.

To display the battery's initial state of charge, a constant block is employed. When running a simulation, I changed the constant block to contain a value of 1, which stands for 100, assuming that the battery is always completely charged. Next, a 'sum' block performs addition and subtraction operations. The signal from the 'discrete time integrator' block will be subtracted from the constant block representing 100 percent charge. These components are connected as shown below. The PS-Simulink converter block is then used to combine these blocks into a subsystem known as "SOC Estimate" and connect it to the battery circuit.

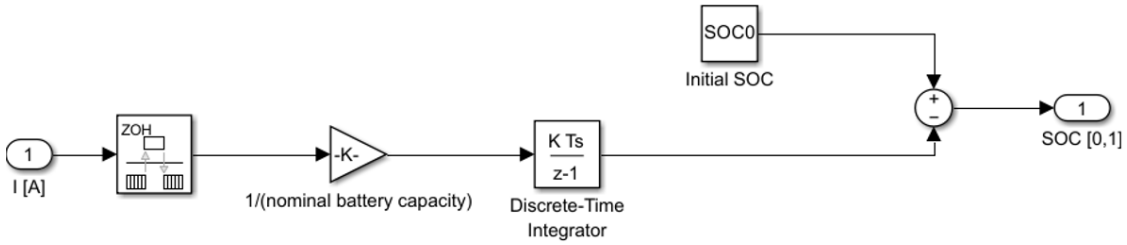


Figure 3.32: Subsystem for the SOC.

To simulate the discharging of the battery, an Idc group block that contains data on the discharging of the current of a similar battery from a MATLAB battery model [28] was used to make the model as realistic as possible. This discharging current was then subtracted from the current coming from the battery, and then the final current was fed into the subsystem for the SOC estimation. The final system relative to the SOC estimation of the battery is shown in the figure 3.33 below.

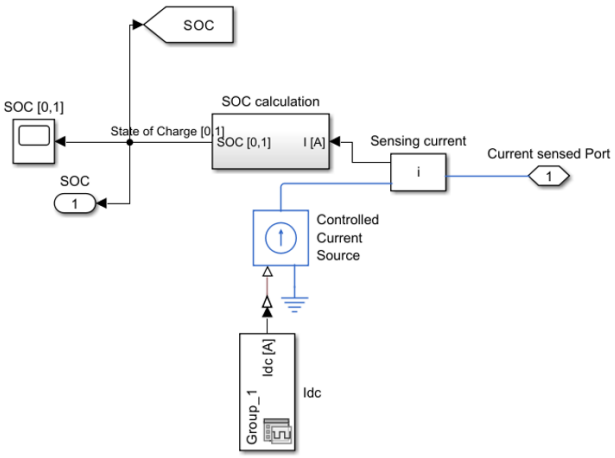
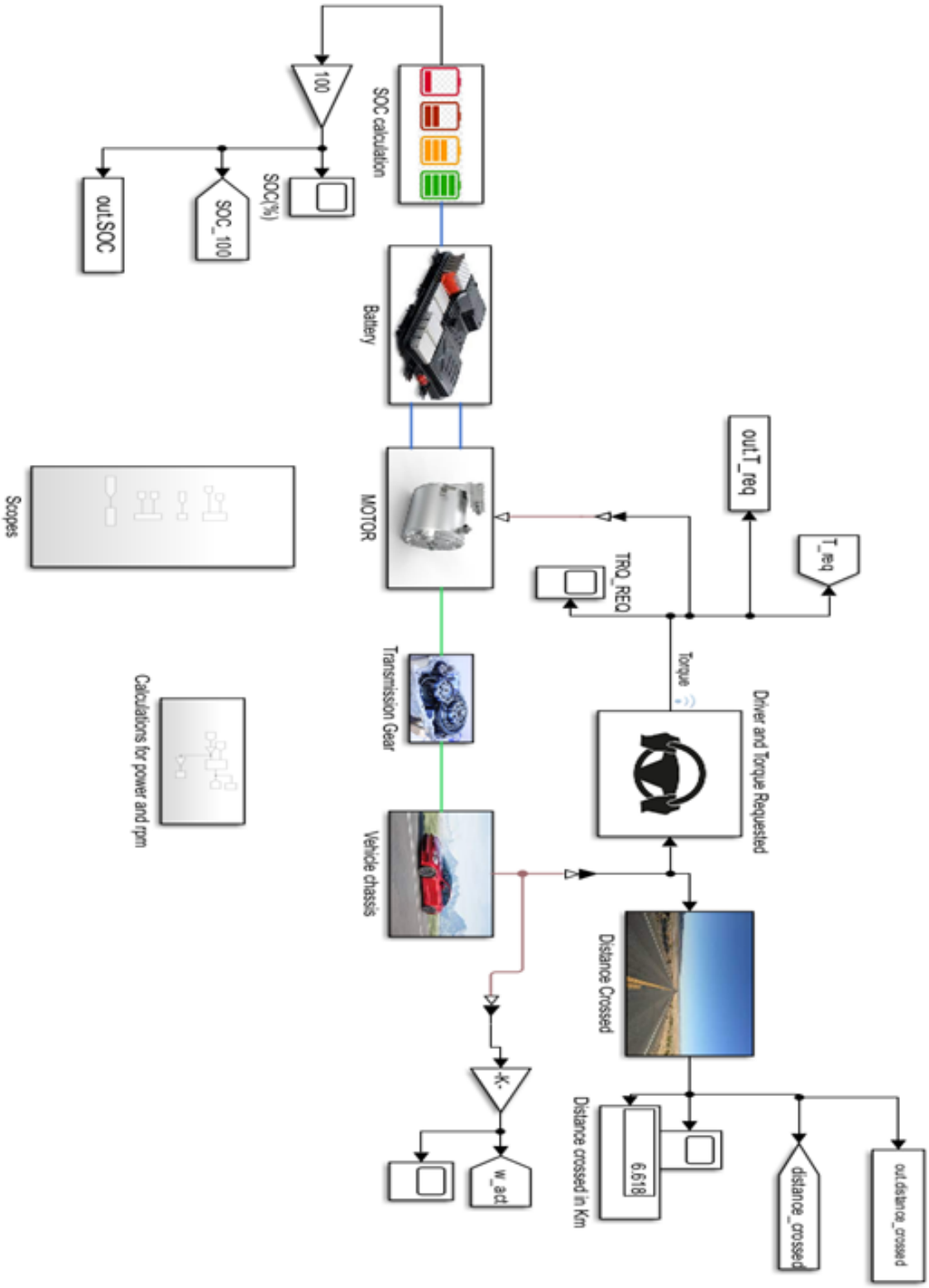


Figure 3.33: SOC Estimation.

The output SOC estimation will be valued between 0 and 1 and then multiplied by a gain of 100 to transform it into a percentage. Now that we have described all the components used in the model, we can show the full compact model before the CAN network is added.

3.7 Full Model


100
SOC calculation
outSOC
SOC_100
SOC(%)
Battery
MOTOR
outT_req
T_req
TRQ_REQ
Scopes
Torque
Driver and Torque Requested
Transmission Gear
Calculations for power and rpm
Vehicle chassis
Distance Crossed
Distance crossed in Km
6.618
distance_crossed
out.distance_crossed
w_act

Chapter 4

Integrating the CAN Into the Model and Simulation Results

After describing how the BEV system model was built, it is time to integrate the CAN network into the model. The initial step is to build the network in the SimScape model by utilizing the vector CANdb++ Editor for the CAN network.

4.1 Creation of CAN Database

CANdb++ Editor is a vector software tool for designing, editing, and managing databases in the CAN communication protocol. It is primarily utilized in automotive and industrial applications where CAN is the preferred communication standard between various ECUs within a Network. Users can design and modify the database, specifying the message frames, signal layouts, and other network-related information required for proper communication between ECUs. It supports the creation of comprehensive databases that facilitate smooth and standardized communication across the network. The CANdb++ Editor also allows users to import and export database files in various formats, enabling compatibility with different CAN network development tools, simulation environments, and hardware devices used for testing and validation. When CANdb++ Editor is first launched, and by clicking

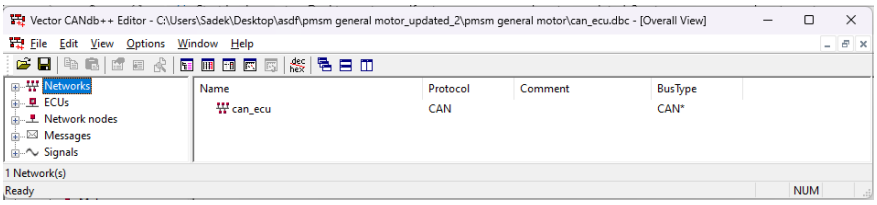


Figure 4.1: CANdb++ Editor dbc file main page

"File", you choose to create a new database (as a dbc file or other files) or open an existing database. The figure above shows the main page when a new database is chosen to be created.

- In the “Network”, we can find the network we created and the chosen protocol. In our case, the Protocol will be “CAN”.
- In the “ECUs”, we can find the electronic control units connected to the Network.
- The ECUs are connected to the networks using the “Network nodes”, where the network nodes act as the interfaces between the ECUs and the network. Several network nodes can be assigned to a control unit, enabling the modelling of a gateway used as a connecting element between two or more CAN buses.
- In the “Messages”, we define the messages we want to transmit over the CAN bus. The message is defined by parameters such as the name, identifier, DLC, cycle time...
- "Signals” represents the smallest unit of information; the signal is then related to a message where it occupies the DATA bytes of the message. One or more signals can be transported in the message depending on the signal length and number of bytes. The signal’s position within the message is specified by indicating the starting bit.

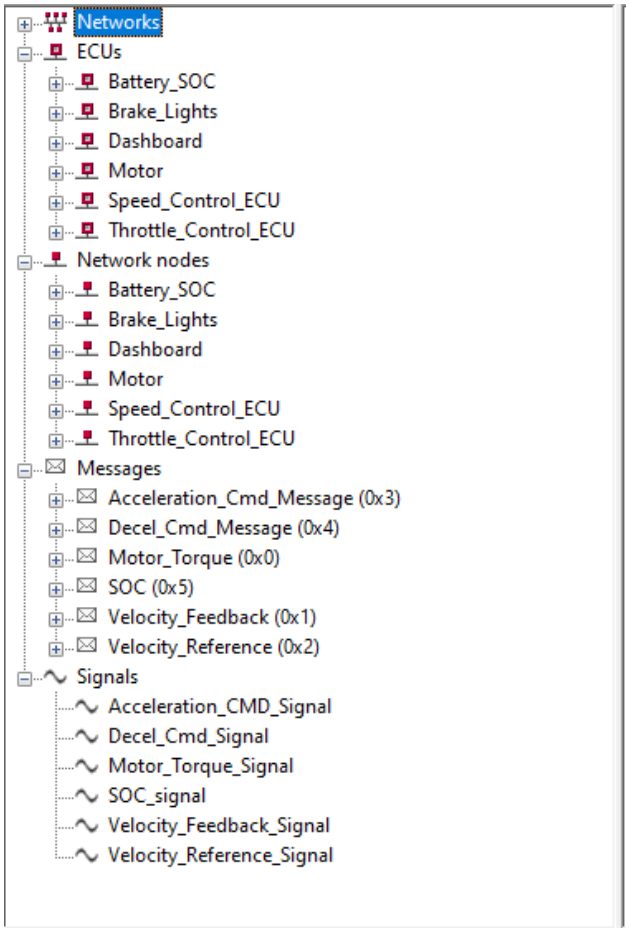


Figure 4.2: CAN network implemented in the model.

4.2 Integrating CAN into the model.

Once the DBC file is ready, the next step is to create the channel, and that is done in the MATLAB script, where we use the vehicle network toolbox to define a virtual channel. Several Blocks are needed in Simulink to integrate the CAN into the model. The first block is the "CAN Configuration" block in which we choose the CAN device, the channel, and its bus speed, as shown in the figure below [29].

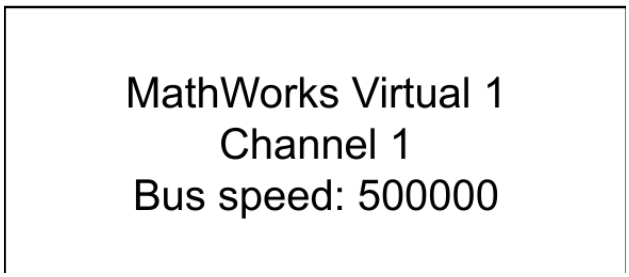


Figure 4.3: CAN configuration block.

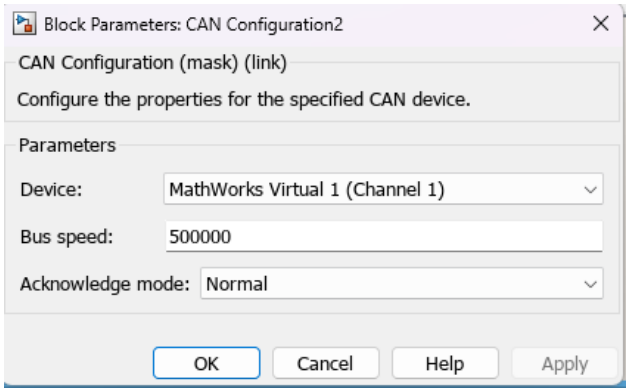


Figure 4.4: CAN configuration parameters.

In the whole block, we need to add two configuration blocks, as one is for the transmission and the other is for receiving the messages. The second block is the "CAN Pack" block, which packs the signal coming from the model and inputs it into the message that contains that specific signal. The block only accepts discrete values, so to transform continuous signals to discrete signals, we add the "Zero-Order-Hold" block and define a sample time of 0.01 [30].

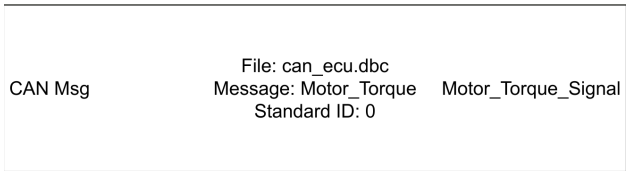


Figure 4.5: CAN pack.

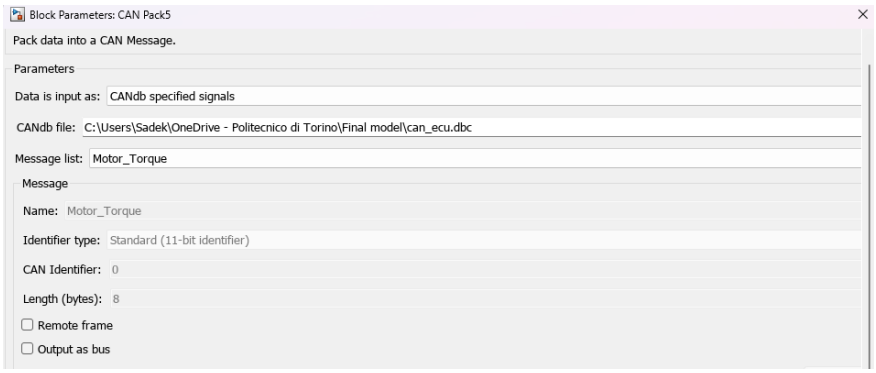


Figure 4.6: CAN pack parameters configuration [30].

Inside the "CAN Pack" block, we should define the CANdb file and the message in which the signal will be included. To transmit the message and the signal on the CAN network, we use the block "CAN Transmit", in which we only define the CAN device as a parameter, as shown in the figure below [31].

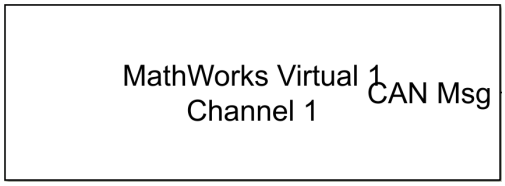


Figure 4.7: CANTrasmit Block [31].

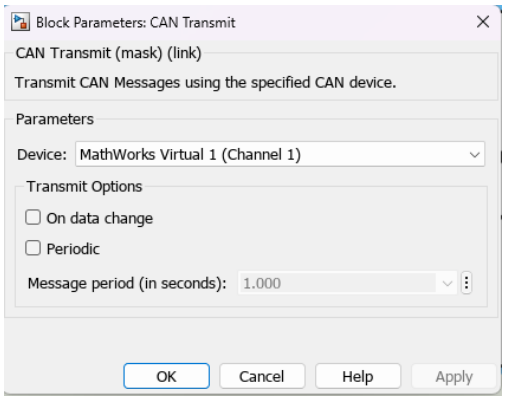


Figure 4.8: CAN transmit block parameters configuration.

Now that we have successfully transmitted the message, we need to receive it from the CAN network and to do so, we use the "CAN Receive" block in the figure below. The CAN Receive block receives messages from the CAN network and delivers them to the Simulink model. It outputs one or all messages at each timestep, depending on the block parameters [32].

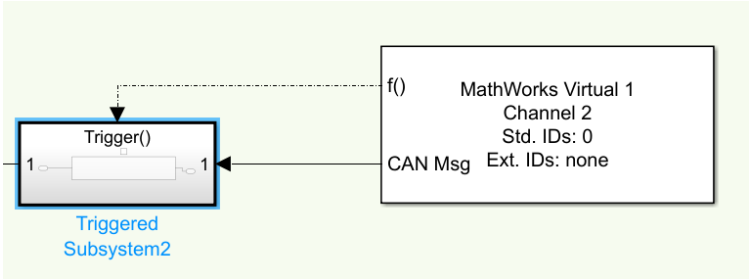


Figure 4.9: CAN Receive and triggered block [32].

It has 2 outport ports [32] :

- CAN Msg: The CAN Msg output port contains one or more packed CAN messages received at that particular timestep, output as a signal bus or CAN_MESSAGE.
- f (): The f () output port triggers a Function-Call subsystem. If the block receives a new message, it triggers a Function from this port. You can then connect to a function-call subsystem to unpack and process a message.

As parameter inputs for the receive block, we should define the device, ID filter if needed to receive only specific messages, and the sample time as shown in the figure below.

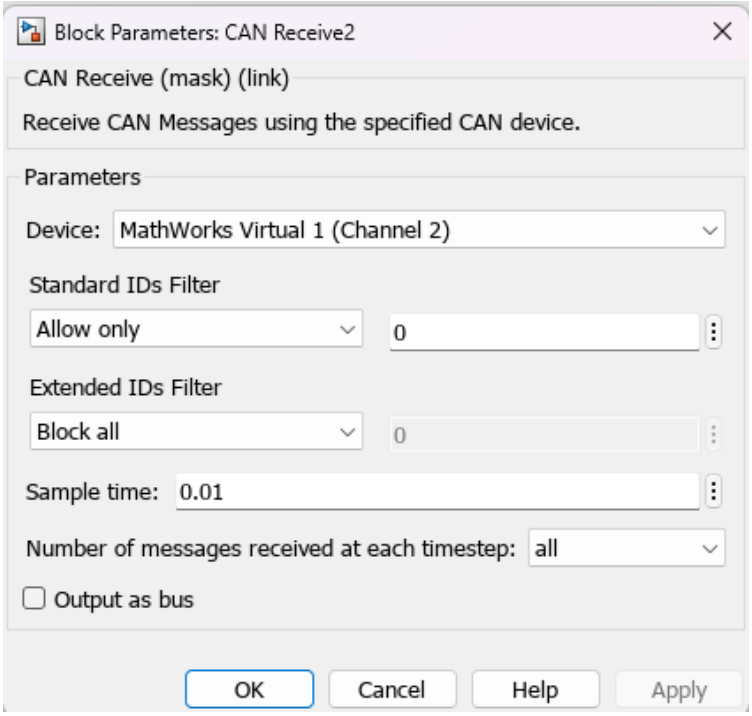


Figure 4.10: CAN Receive block parameters configuration.

Now, inside the triggered subsystem, we define the "CAN Unpack" block needed to unpack Data from a CAN message into signal data. As input, it takes the CAN message

that comes from the "CAN receive" block, and it's triggered each time a message is received thanks to the trigger function f(). As output, we get the unpacked signal [33].

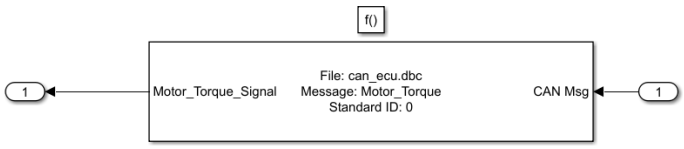


Figure 4.11: CAN Unpack block [33].

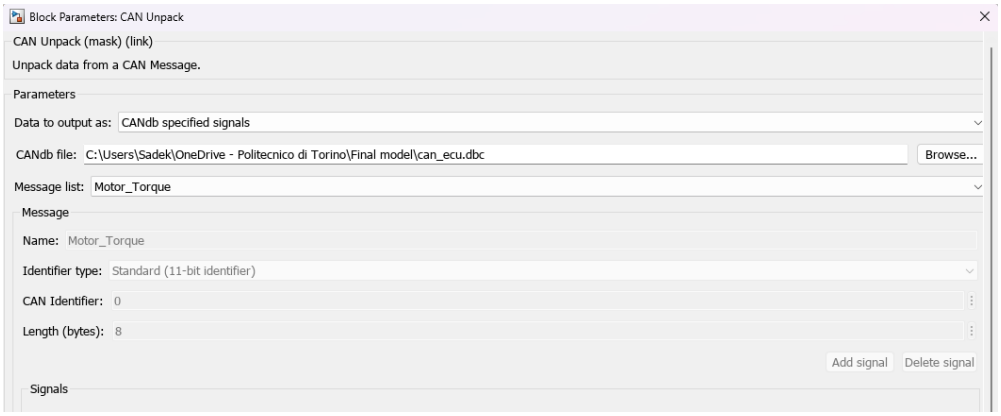


Figure 4.12: CAN Unpack block parameters configuration.

Now that we have defined all the necessary blocks to integrate the CAN network into the model, we can show the complete blocks in action for some messages, such as the torque, speed, and State of charge.

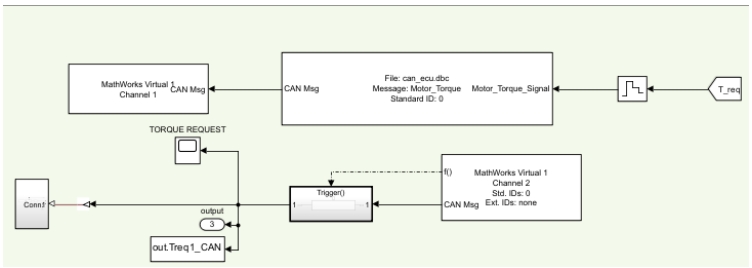


Figure 4.13: Torque Request through CAN.

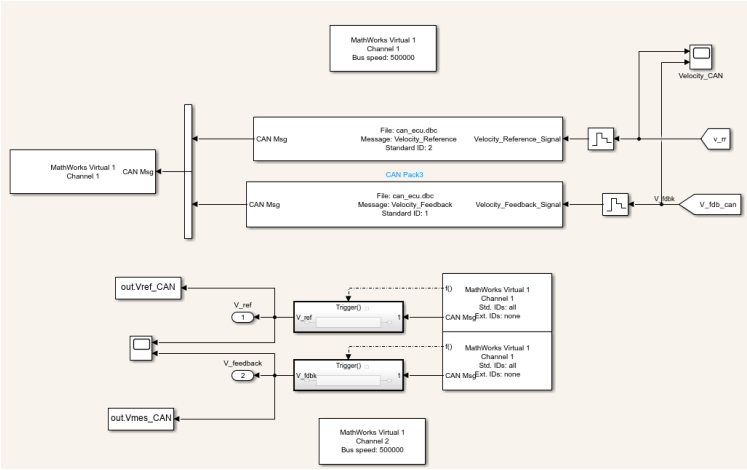


Figure 4.14: Velocity through CAN

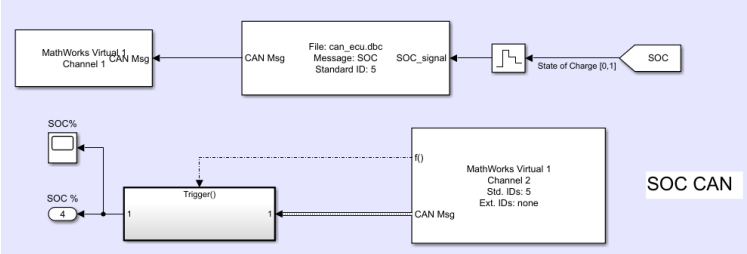


Figure 4.15: SOC through CAN

4.3 Full Model with the CAN integrated

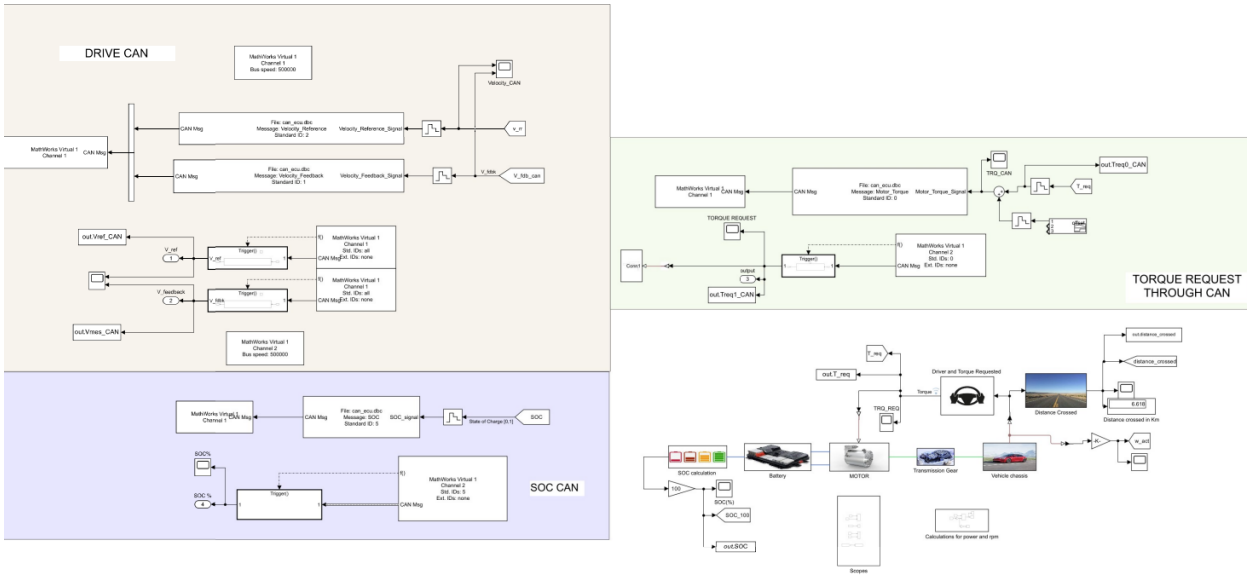


Figure 4.16: Full Model with CAN integrated

4.4 CAN Attack

4.4.1 Literature

CAN attacks refer to various security threats and vulnerabilities associated with using the CAN protocol in automotive and industrial control systems. The inherent design of CAN, which prioritizes efficiency and simplicity, can sometimes lead to vulnerabilities that attackers may exploit. Different types of attacks on CAN are present that can be summarized below as follows :
Message Manipulation: Message manipulation involves altering the content of CAN messages to deceive or disrupt the normal operation of connected devices. An attacker could inject a forged message into the CAN bus, tricking a vehicle's engine control unit (ECU) into adjusting fuel injection parameters, leading to inefficient fuel combustion or potential engine damage.
Denial of Service (DOS): Denial of Service attacks aim to overwhelm the CAN bus with a high volume of messages, disrupting normal communication and causing a temporary or permanent loss of service. Flooding the CAN bus with many messages can overload the bandwidth, preventing critical messages, such as those related to braking or acceleration, from reaching their destinations on time.
Eavesdropping: Eavesdropping involves unauthorized interception of CAN messages to gather sensitive information without altering the transmitted data. An attacker could tap into the CAN bus to listen to messages related to unlocking car doors, capturing signals that could later be used to gain unauthorized access.
Spoofing: Spoofing attacks involve impersonating a legitimate device on the CAN bus by sending messages with fake identifiers. An attacker might send messages with the identifier of a trusted sensor, providing false data to the ECU. For instance, sending incorrect speed information leads to incorrect vehicle speed readings.
Replay Attacks: In replay attacks, attackers capture and later retransmit valid CAN messages to achieve unauthorized actions. An attacker captures a legitimate unlock signal from a key fob to the car's security system. The attacker later replays this signal to unlock the vehicle without the original key fob.

It's crucial to note that the automotive industry is aware of these security concerns, and efforts are made to enhance the security of CAN networks. This includes developing and adopting secure communication protocols, implementing encryption techniques, and using intrusion detection systems to identify and mitigate potential attacks. The first known attack on a CAN bus ever recorded took place in 2007, where Hoppe and Dittman were able to manipulate the windows lift [34]. Since then, different attack scenarios have been implemented. In 2010, Koscher et al. [35] demonstrated that attackers can infiltrate virtually any ECU, they CAN leverage this ability to widen their attack on other critical ECUs over a set of experiments. In their attack, they could control the engagement and disengagement of the brakes while driving, making it difficult for the driver to stop. They managed to access the bus by connecting to the OBD port, embedding the malware inside the vehicles' components,, and then disconnected from it. Some critics say that the physical access requirement of the

CAN attacks make them infeasible, but remote attacks took place over the years.
Modern Cars are equipped with wireless interfaces such as telematics, remote keyless entry, Bluetooth, etc. These interfaces may have communication with the CAN network via a gateway ECU with a firewall, So if attackers can overpass the firewall, then they can connect to the CAN network and manipulate it as demonstrated by different researchers during the years [36],[37],[38]. In 2014, to attract eyes and research to the security issue of the CAN network, researchers Charlie Miller and Chris Valasek performed a remote attack against an unaltered 2014 Jeep Cherokee and similar vehicles. This experiment raised concerns about the security of the CAN bus, where on the same day they published the video of their attack, Fiat Chrysler Automotive announced a recall of 1.4 million affected vehicles so a security update could be installed.
At the time, it was the first physical product recall caused by a cybersecurity vulnerability. To perform their attack, they identified vulnerabilities in the Uconnect infotainment system, which at that time was the in-car entertainment and communication system used by FCA vehicles. Then, they exploited these vulnerabilities to access the vehicle's CAN bus by remotely accessing it through a cellular connection. Once they got inside the infotainment system, they were able to send malicious commands to the CAN bus, which controls different functions in critical systems such as brakes, steering, and engine control. Their video demonstrated that they can turn off brakes, control the steering, and even turn off the engine while the car is in motion all from a remote. The incident prompted increased attention to cybersecurity in the automotive industry, leading to improved security measures for vehicle software and communication systems [38].

Figure 4.17: As part of their hack, Charlie Miller and Chris Valasek broadcast their cartoon images on the entertainment centre display of a Jeep Cherokee driven by Wired's Andy Greenberg (YouTube screenshot)[39].

4.4.2 CAN injection in our model

Simulating an Attack on the CAN network of a BEV Simulink model can give us an idea of how dangerous it can be, especially if the attack involves critical ECUs like the ones responsible for the brakes, motor, and at the same time it is cost and time efficient. In our model, we assume that attackers could access the CAN network to send malicious messages and signals. To simulate the CAN injection attack, we used the signal builder block in Simulink in such a way we could build different signals to simulate different attacking behaviours and see their effect on the vehicle. The attack was done on the motor CAN as a critical component.

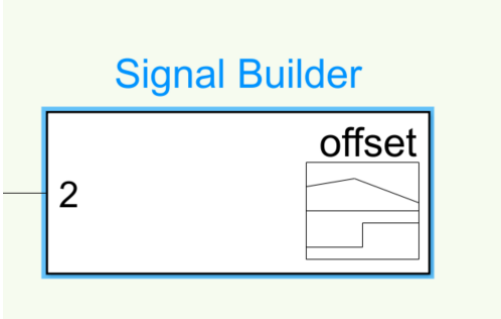


Figure 4.18: Signal Builder in Simulink to create the offset for the attack.

Inside the signal builder, we can create different signals to simulate different scenarios.

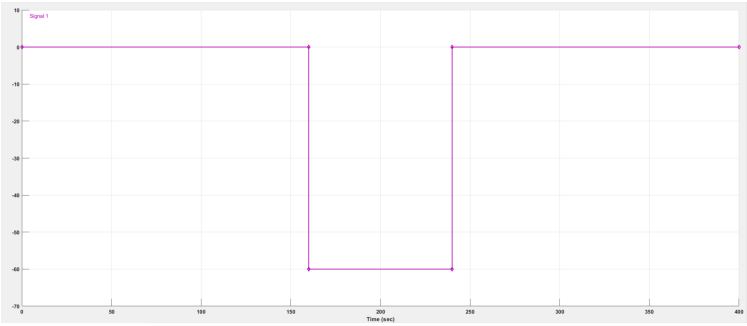


Figure 4.19: Signal created to simulate a braking torque.

For example, as shown in the figure, one of the scenarios we simulated was to send a braking torque signal to the ECU of the motor and force the motor to brake when it shouldn't. The frequency and the amplitude of the attack signal can be adjusted to control the severity of the attack. After creating the signal, we connect the signal builder to the CAN model, where the signal coming from the torque requested and the attack signal offset coming from the signal builder block are then summed up using the Sum block that adds the attack signal to the requested torque signal. This effectively injects the attack signal into the normal operation of the motor control system. As shown in the figure below, the summed signal is then packed as one signal inside the CAN message of the motor torque,. The resulting CAN message is then sent to the motor controller, which interprets it as

the requested torque signal. The attack signal effectively overrides the driver's input and causes the motor to operate at a different speed and torque than intended„ as shown in the simulations.

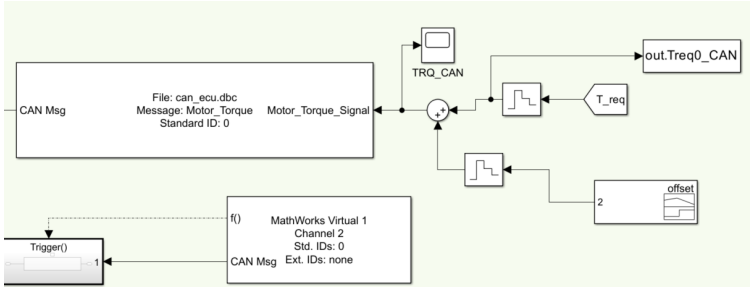


Figure 4.20: CAN signal attack integration into the model.

4.5 Simulation Results and analysis

To make the model user-friendly and for the tests to run automatically, we implemented a code inside MATLAB to prompt the user. The prompt asks the user to choose the desired driving cycle to simulate, and then the simulation is performed automatically. As shown in the figure below, the driving cycles available are the Artemis Motorway Driving cycle (AMDC), World Harmonized Vehicle Cycle (WHVC), and Extra Urban driving cycle (EUDC) (low-powered vehicles). Each driving cycle has a different set of speed trajectory and acceleration data. The chosen driving cycle will simulate the vehicle's performance under real-world driving conditions.

```
Enter the number of the desired driver cycle-choose:
 1 for Artemis Motorway Driving cycle
 2 for WHVC
 3 for ECE urban driving cycle
 4 for cruise control
```

Figure 4.21: Prompt to ask the user to choose the driving cycle.

The simulation is performed automatically once the user enters the desired driving cycle number. The MATLAB code will read the data for the chosen driving cycle and use it to simulate the vehicle's performance. The results of the simulation will be displayed in the MATLAB workspace.
To verify that the model works well and tracks the references, it is essential to simulate the model under normal operating conditions without attack. This allows us to assess the model's ability to produce accurate results and maintain stability in the absence of malicious interference, we first simulate three driving cycles without the attack to see how the model behaves.

4.6 Artemis Motorway Driving Cycle Results

Before showing the different results obtained for this driving cycle, it is worth showing the desired velocity trajectory for this cycle that will act as a reference for our model. We will also reference the desired velocity trajectory for the rest of the cycles.

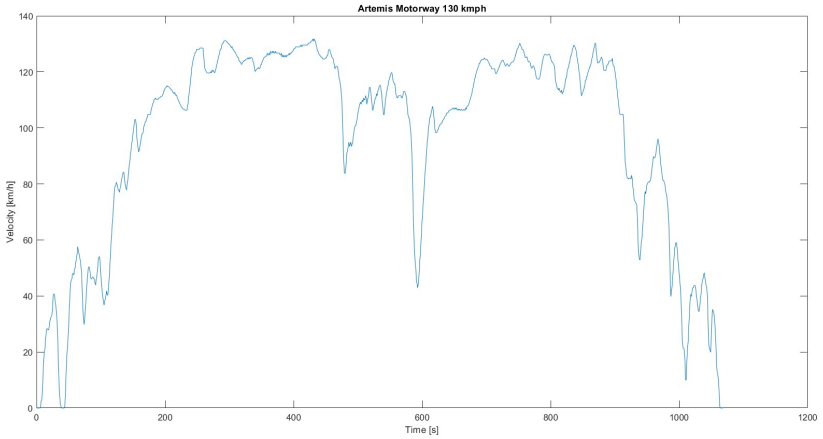


Figure 4.22: Artemis Motorway Reference Driving Cycle.

As we can see from the title of this graph, this cycle has an overall duration of 1068 seconds, and it tests the car's capability at a relatively high speed, starting from 0 and reaching a maximum speed of 130 kmph.

4.6.1 Torque vs Time.

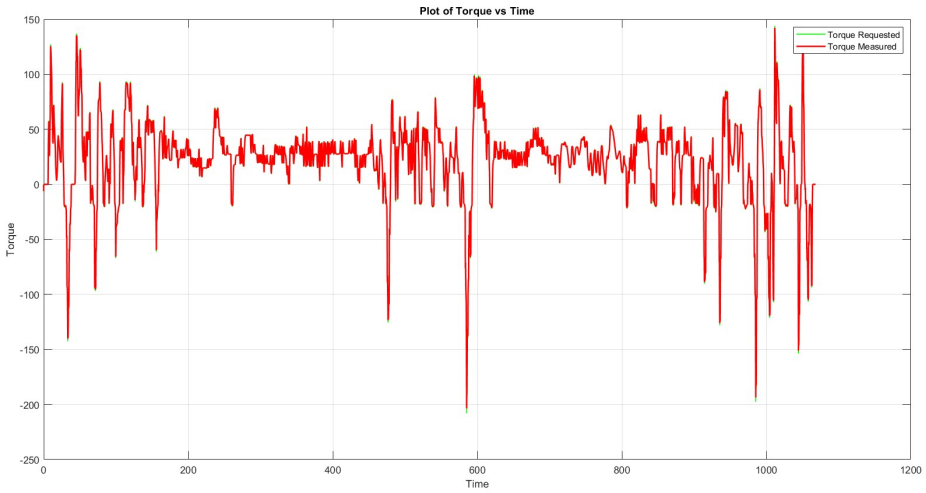


Figure 4.23: Plot of Torque vs Time.

In Figure 4.23, we can see the obtained results regarding the torque behaviour, where the green line represents the torque request, and the red line represents the torque obtained. As you can see, the torque requested and obtained are closely matched most of the time. This is a good thing because it means the motor control system is working well and accurately tracking the references.
To better visualize the tracking of the model, we can look at the velocity profile requested and obtained by the model.

4.6.2 Velocity vs Time

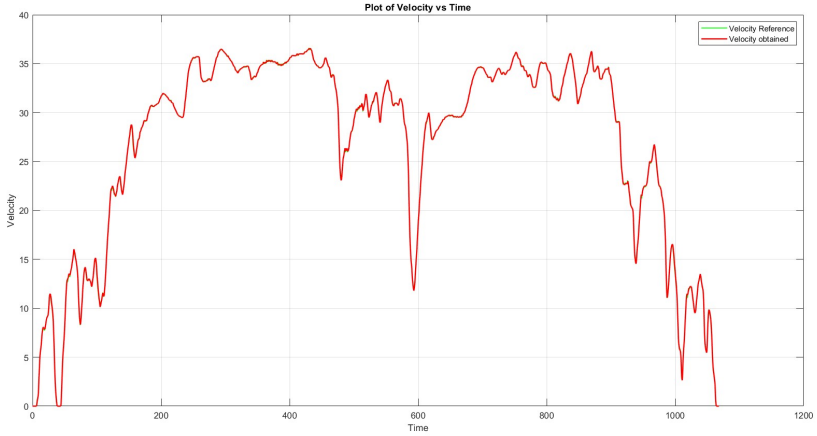


Figure 4.24: Plot of Velocity vs Time.

Also, in Figure 4.24, the red line stands for the actual velocity, and the green line for the reference velocity. It's clear that the model is tracking the reference velocity profile very well from this point of view, but to be sure, we should take a closer look at different points in the graph. In the close-up present in Figure 4.25, we can see a difference and some delay between the two lines, and that is due to the sudden change in the reference velocity at edges and the short time for the model to act up and follow it. Still, the model tracks well even with some delay and a few overshoots/undershoots.

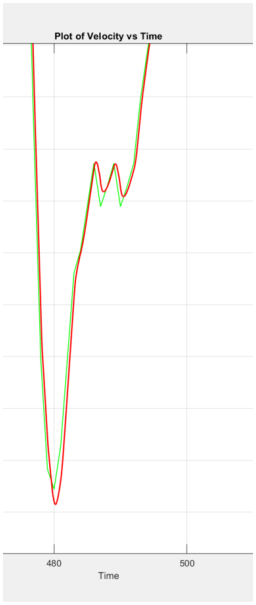


Figure 4.25: Close up to the velocity profile

4.6.3 Acceleration and Deceleration vs Time

Figure 4.26: Normalized Acceleration and Deceleration vs Time.

Figure 4.26 shows how the throttle changes between the acceleration and deceleration pedals. Both the acceleration and deceleration commands were normalized, as explained in Chapter 3. We can now also check if the CAN network is working as it should by monitoring the same signals and receiving the messages over the CAN network.
To monitor the signals and the messages received, we can do this in 2 ways: by using the CAN Explorer APP in the vehicle network toolbox or by implementing a code that sends the messages and the signals into MATLAB's workspace. Both ways are possible to be

used in the model, but we will explain how to visualize and receive signals in the CAN Explorer APP as it's such an interesting application and provides us with good visualization of the signals/messages and shows how the messages are shown in MATLAB's Workspace.

CAN Explorer APP: The CAN Explorer app in Vehicle Network Toolbox™ is a graphical user interface (GUI) tool that simplifies the visualization and analysis of CAN bus data. The app allows a variety of CAN interface devices to be connected to it, such as the ones of Vector and the virtual devices of MATLAB. For our model, we used a virtual device. Once connected, we can view real-time CAN bus traffic and decode messages into their signals with their respective values. Besides choosing the device, we should also provide the CAN database file(.dbc) to the CAN Explorer App.

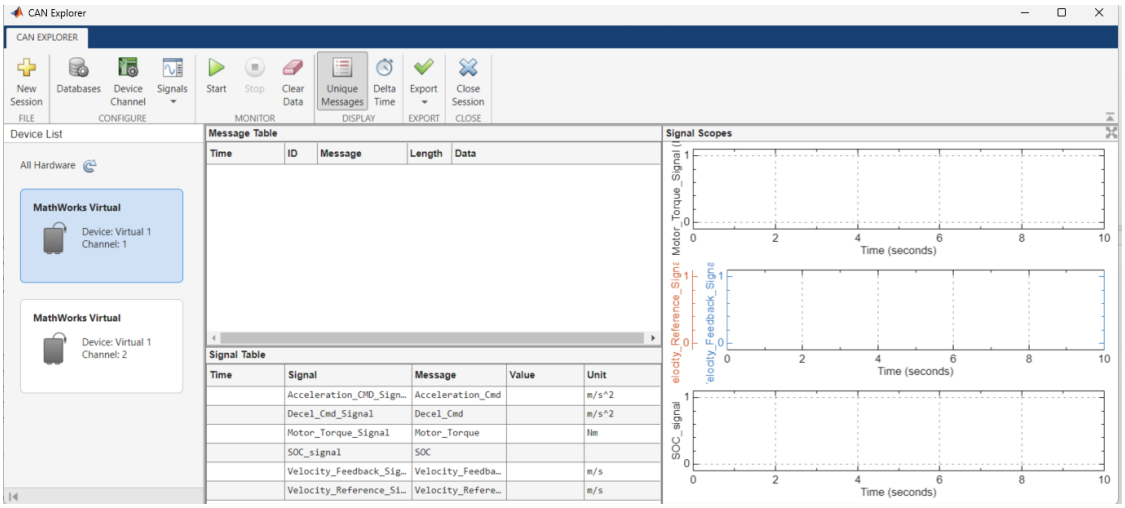


Figure 4.27: CAN Explorer App Interface.

As we can see in figure 4.27 above, on the left, we have the device we chose; in the middle white space, we will have a table for the messages that are being transmitted, and in the mid-bottom, a table of the signals we chose to monitor. On the far right, we can visualize the signals concerning time.
Once we start the application, we will see the messages being transmitted, the signal values, and their respective plot. At the end of the simulation, we can also choose to export the data acquired as shown in Figure 4.28

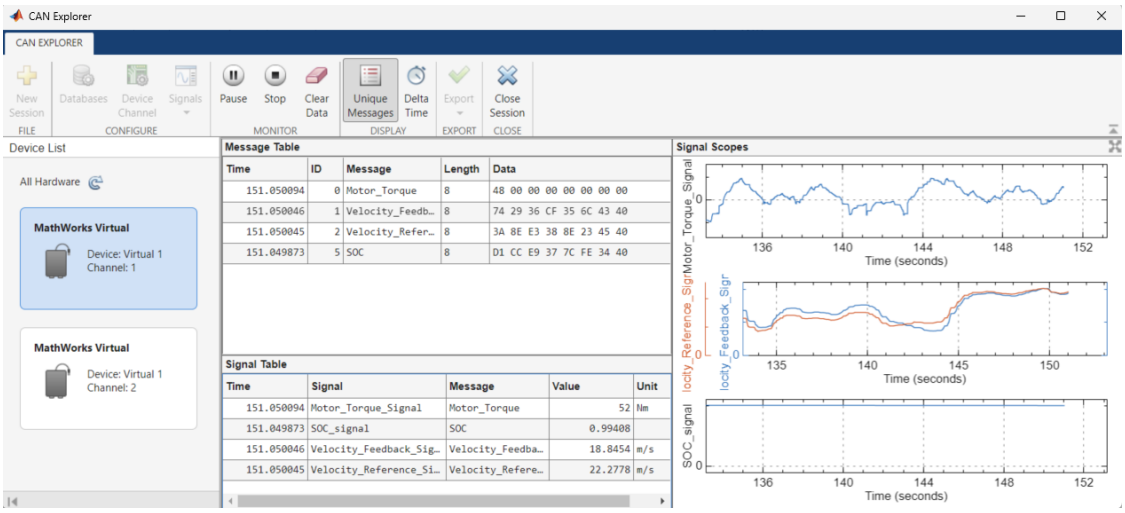


Figure 4.28: CAN Explorer App in action

In Figure 4.28, the messages are being received, the signal is being read, and its value with its respective unit is also shown. CAN Explorer is suitable for reading and receiving messages live. However, it provides poor graphs, and for that, we decided to check the graphs that come from the CAN network in the model, as shown in Figures 4.29 and 4.30.

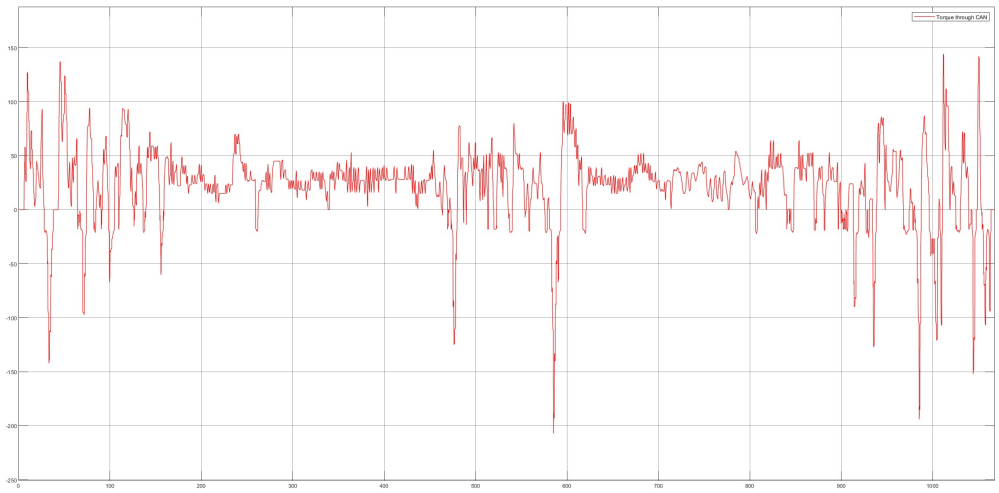


Figure 4.29: Torque Signal Through CAN.

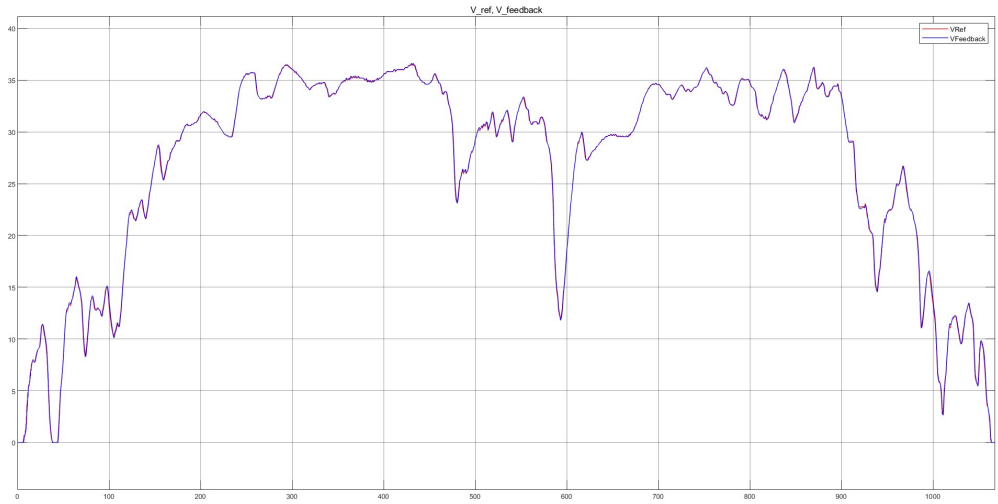


Figure 4.30: Velocity Signal Through CAN.

The tracking here of the reference ensures that the integration of the CAN network into the model is correct, as the model is tracking the reference as expected.

```
1
You chose Artemis Motorway driving cycle

msgs =

  427204×8 timetable

    Time        ID    Extended    Name                 Data                                  Length    Signals        Error    Remote
    _________    __    ________    _________________    ___________________________________    ______    ___________    _____    ______

    126.42 sec    5       false    {'SOC'              }    {[           0 0 0 0 0 0 53 64]}        8    {1×1 struct}    false    false
    126.62 sec    2       false    {'Velocity_Reference'}    {[           0 0 0 0 0 0 52 64]}        8    {1×1 struct}    false    false
    126.62 sec    1       false    {'Velocity_Feedback' }    {[           0 0 0 0 0 0 52 64]}        8    {1×1 struct}    false    false
    126.64 sec    0       false    {'Motor_Torque'     }    {[            20 0 0 0 0 0 0 0]}        8    {1×1 struct}    false    false
    127.42 sec    5       false    {'SOC'              }    {[           0 0 0 0 0 0 53 64]}        8    {1×1 struct}    false    false

        :         :         :              :                                    :                    :          :           :        :

    258.41 sec    0       false    {'Motor_Torque'     }    {[            20 0 0 0 0 0 0 0]}        8    {1×1 struct}    false    false
    258.41 sec    5       false    {'SOC'              }    {[    254 33 21 8 13 249 52 64]}        8    {1×1 struct}    false    false
    258.41 sec    2       false    {'Velocity_Reference'}    {[           0 0 0 0 0 0 52 64]}        8    {1×1 struct}    false    false
    258.41 sec    1       false    {'Velocity_Feedback' }    {[40 245 255 255 255 255 51 64]}        8    {1×1 struct}    false    false
    258.41 sec    0       false    {'Motor_Torque'     }    {[            20 0 0 0 0 0 0 0]}        8    {1×1 struct}    false    false

  Display all 427204 rows.
```

Figure 4.31: Timetable of the CAN Messages in Matlab Workspace.

In Figure, 4.31, we can see that the user chose the “Artemis Motorway Driving Cycle”, and then the messages are printed in the workspace and saved as a timetable inside a variable called “msgs” that can also be accessed from MATLAB.

We will directly show the results for the other two driving cycles without commenting on them, as they also track the reference cycles well.

4.7 World harmonized vehicle driving cycle Simulation Results.

Also, for this driving cycle, it is worth showing the desired velocity trajectory that will act as a reference for our model.

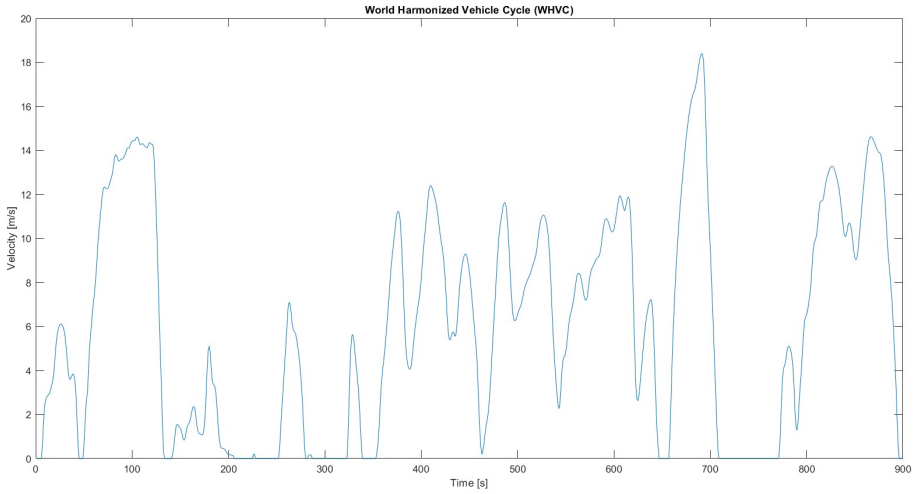


Figure 4.32: WHVC Reference Cycle.

4.7.1 Torque vs Time.

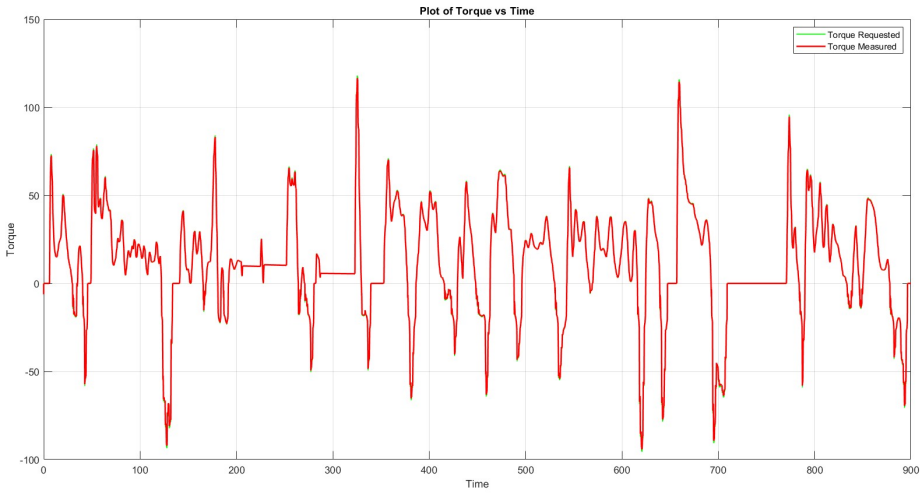


Figure 4.33: Plot of Torque vs Time.

4.7.2 Velocity vs Time.

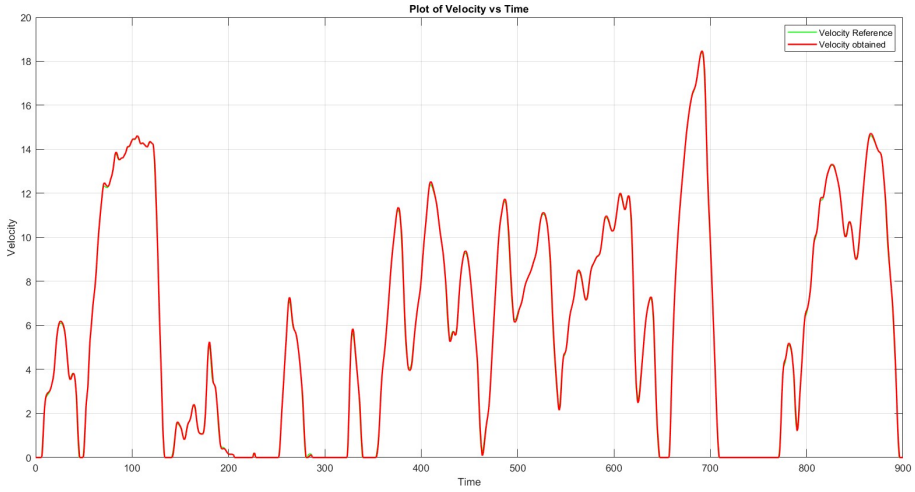


Figure 4.34: Plot of Velocity vs Time.

4.7.3 Acceleration and Deceleration vs Time.

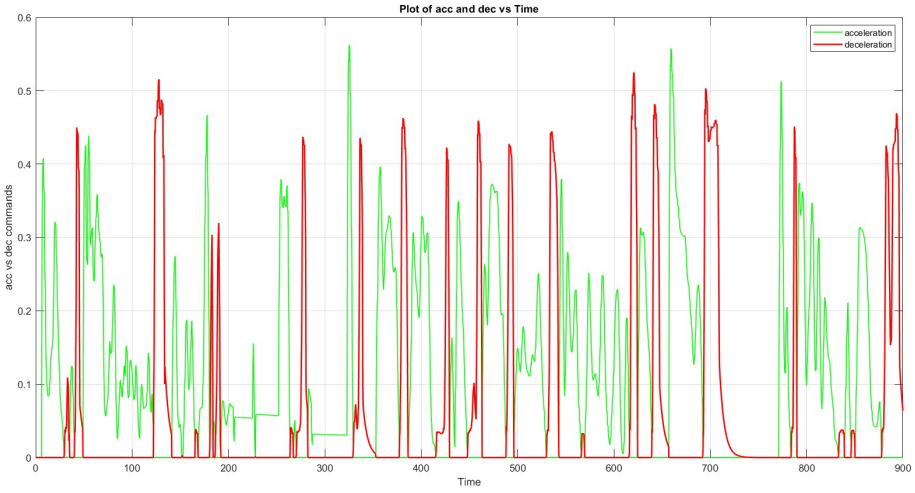


Figure 4.35: Plot of Acceleration and Deceleration vs Time.

4.7.4 Signals Transmitted through CAN vs Time.

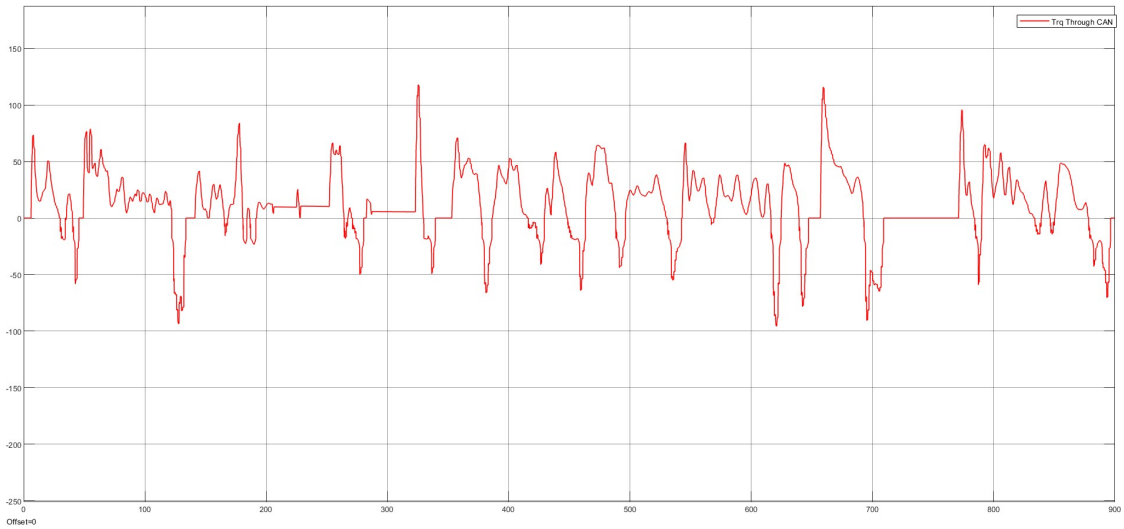


Figure 4.36: Plot of Torque Signal Through CAN.

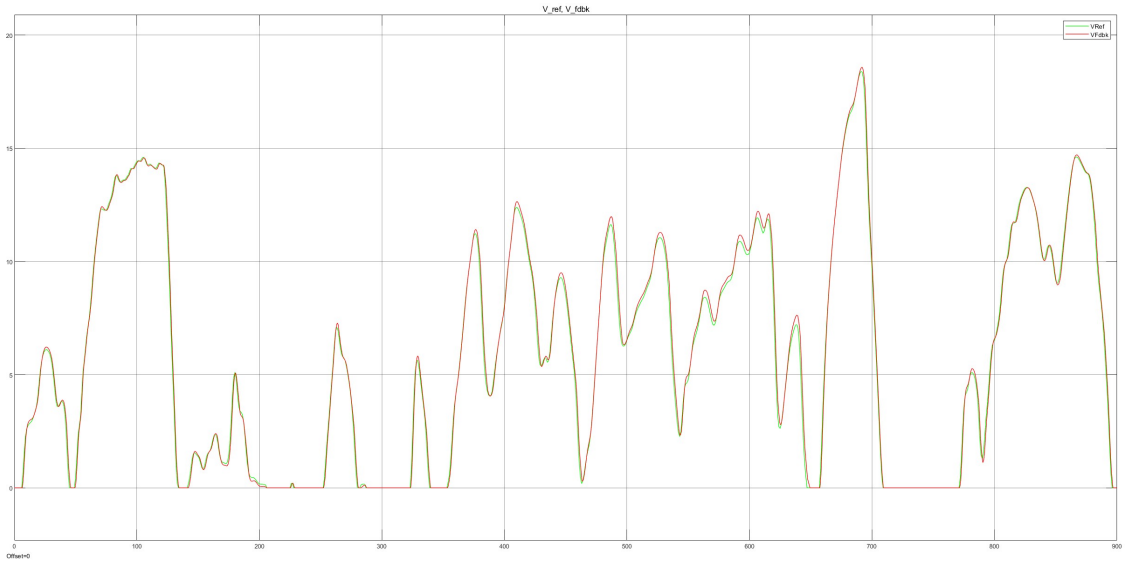


Figure 4.37: Plot of Velocity Signal Through CAN.

4.7.5 Messages Transmission through CAN APP and Matlab Workspace.

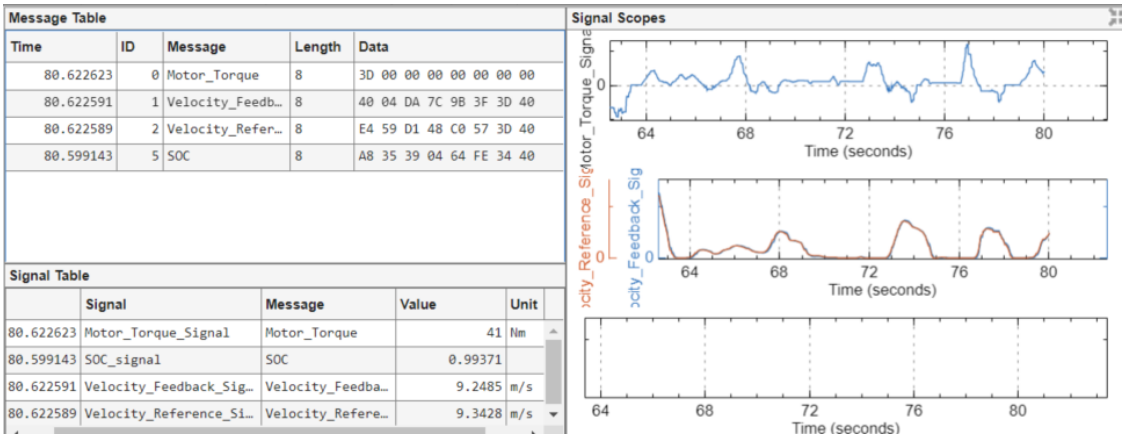
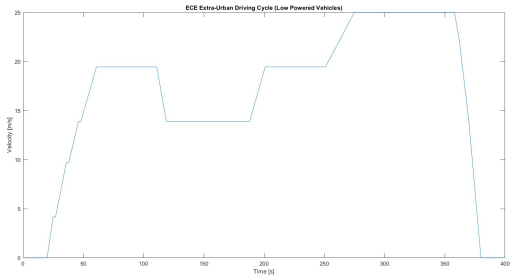


Figure 4.38: CAN Explorer App in action

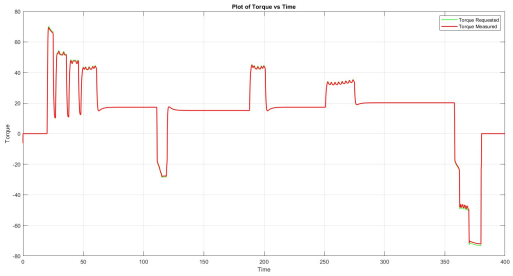
63.384 sec	0	false	{'Motor_Torque' }	{[219 255 255 255 255 255 255 255]}	8	{1×1 struct}	false	false
63.387 sec	5	false	{'SOC' }	{[112 112 238 11 46 249 52 64]}	8	{1×1 struct}	false	false
63.387 sec	2	false	{'Velocity_Reference'}	{[245 73 159 244 73 159 52 64]}	8	{1×1 struct}	false	false
63.387 sec	1	false	{'Velocity_Feedback' }	{[222 220 149 22 108 75 53 64]}	8	{1×1 struct}	false	false
63.387 sec	0	false	{'Motor_Torque' }	{[219 255 255 255 255 255 255 255]}	8	{1×1 struct}	false	false
63.391 sec	2	false	{'Velocity_Reference'}	{[139 194 245 40 92 143 52 64]}	8	{1×1 struct}	false	false
63.391 sec	1	false	{'Velocity_Feedback' }	{[125 5 52 198 136 46 53 64]}	8	{1×1 struct}	false	false
63.391 sec	0	false	{'Motor_Torque' }	{[239 255 255 255 255 255 255 255]}	8	{1×1 struct}	false	false
63.411 sec	2	false	{'Velocity_Reference'}	{[34 59 76 93 110 127 52 64]}	8	{1×1 struct}	false	false
63.411 sec	1	false	{'Velocity_Feedback' }	{[89 142 87 191 60 20 53 64]}	8	{1×1 struct}	false	false
63.411 sec	0	false	{'Motor_Torque' }	{[249 255 255 255 255 255 255 255]}	8	{1×1 struct}	false	false
63.415 sec	2	false	{'Velocity_Reference'}	{[185 179 162 145 128 111 52 64]}	8	{1×1 struct}	false	false
63.415 sec	1	false	{'Velocity_Feedback' }	{[100 230 22 32 123 255 52 64]}	8	{1×1 struct}	false	false
63.415 sec	0	false	{'Motor_Torque' }	{[249 255 255 255 255 255 255 255]}	8	{1×1 struct}	false	false
63.418 sec	2	false	{'Velocity_Reference'}	{[100 44 249 197 146 95 52 64]}	8	{1×1 struct}	false	false
63.418 sec	1	false	{'Velocity_Feedback' }	{[153 134 62 35 220 234 52 64]}	8	{1×1 struct}	false	false
63.418 sec	0	false	{'Motor_Torque' }	{[249 255 255 255 255 255 255 255]}	8	{1×1 struct}	false	false
63.421 sec	2	false	{'Velocity_Reference'}	{[250 164 79 250 164 79 52 64]}	8	{1×1 struct}	false	false

Figure 4.39: Snapshot of the Timetable of the CAN Messages in Matlab Workspace.

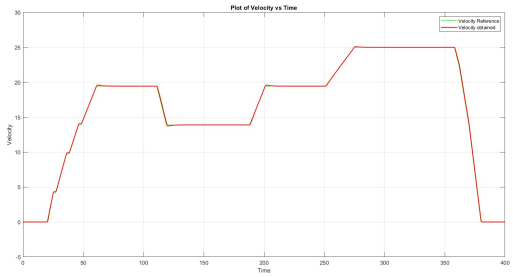
4.8 Extra Urban Driving Cycle Simulation Results



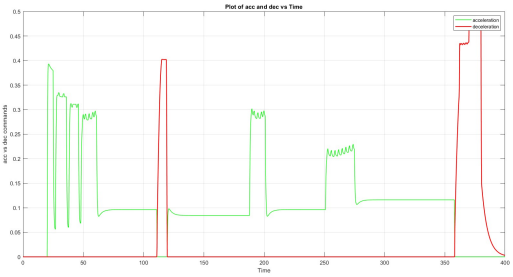
(a) Reference Velocity Cycle



(b) Plot of Torque vs Time



(c) Plot of Velocity vs Time



(d) Plot of Acceleration and Deceleration vs Time.

Figure 4.40: Different Cycle Results

4.9 Cruise Control Simulation Results

The last test was to check the model response to perform cruise control at 100 Km/h (around 28 m/s). Also, in this case, the model gave us good results with a little overshoot, as expected. For this test, we will limit the plots to the velocity plot.

4.9.1 Plot of Velocity vs Time.

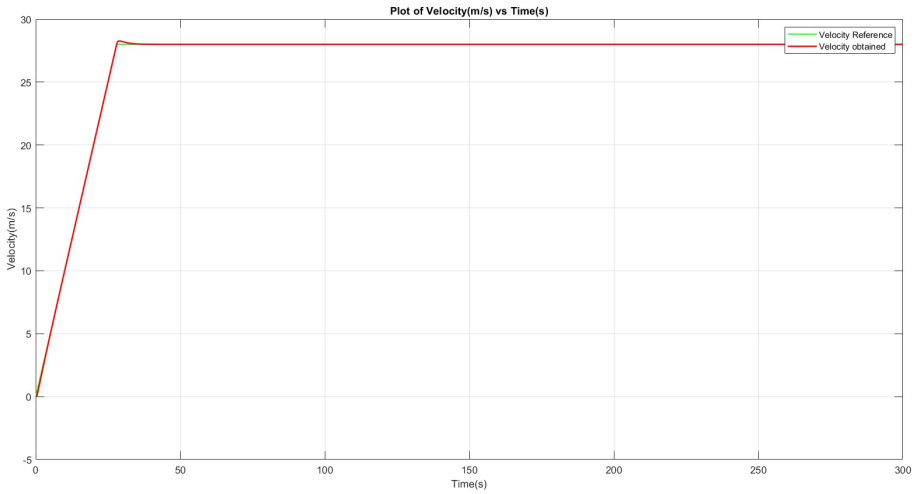


Figure 4.41: Velocity profile during Cruise Control.

4.10 CAN Scenario Attacks

4.10.1 First Attack

To simulate different scenarios of CAN attacks, we will use the signal builder explained in 4.19. For the first attack, we will use a similar signal to the signal created in 4.20, which simulates a braking torque, and observe what happens.
The first attack will be performed on the "ECE Extra Urban Driving Cycle" as it has a clear velocity trajectory, which allows us to notice the attack well. As mentioned, we first set up the attack signal using the signal builder. It is a signal that emulates a braking torque, and we will make it apply on the cycle when the vehicle is not braking to see its effect on the velocity trajectory, as shown in Figure 4.42.

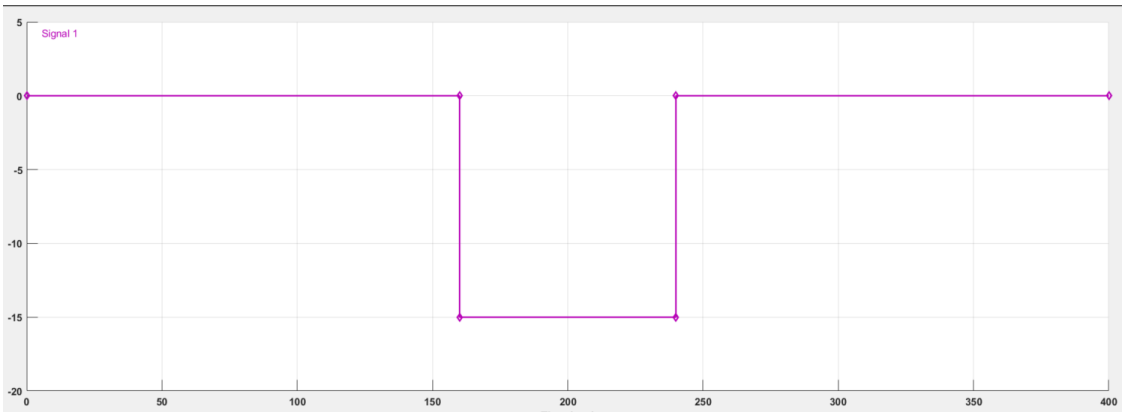


Figure 4.42: Signal created to simulate Braking Torque.

After running the simulation, we obtain the following graphs:

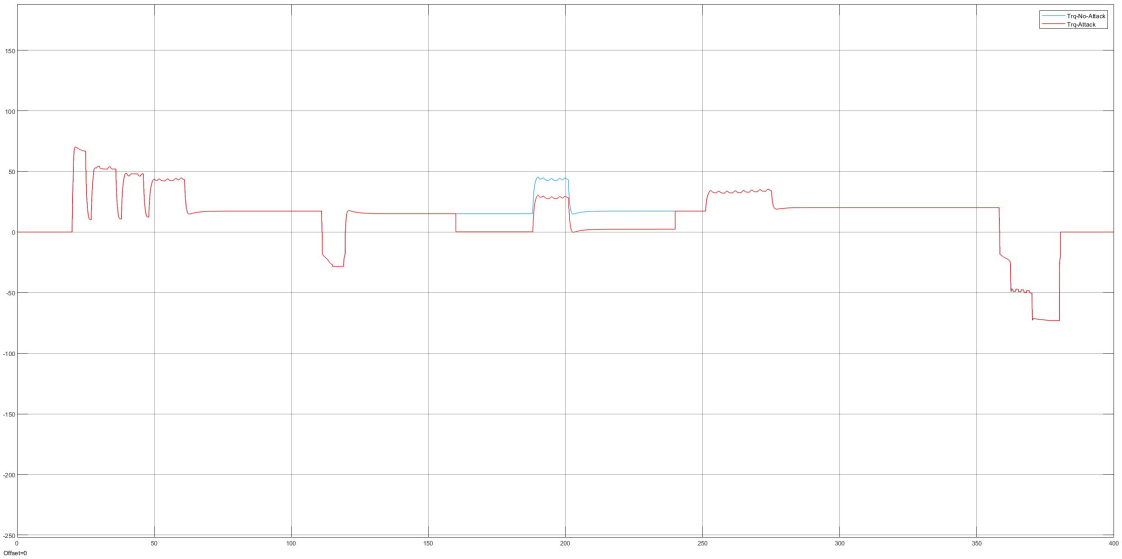


Figure 4.43: Torque without any attack vs Torque with the attack.

The attack signal was created so it takes place at time t=160s and up to t=240s, and this is also clear by looking at the graph comparison between the Torque with and without any attack, From t=0s up to t=160s since no attack was taking place both the torque signal without attack represented in blue. The torque signal with attack represented with yellow had the same values. At t=160 seconds, the torque in blue had a constant value of around 15 Nm; with the attack applied, the graph shifted downstairs by -15 Nm, the same magnitude as the signal created. We can now observe the plot of the velocity present in Figure 4.44 to notice its effect.

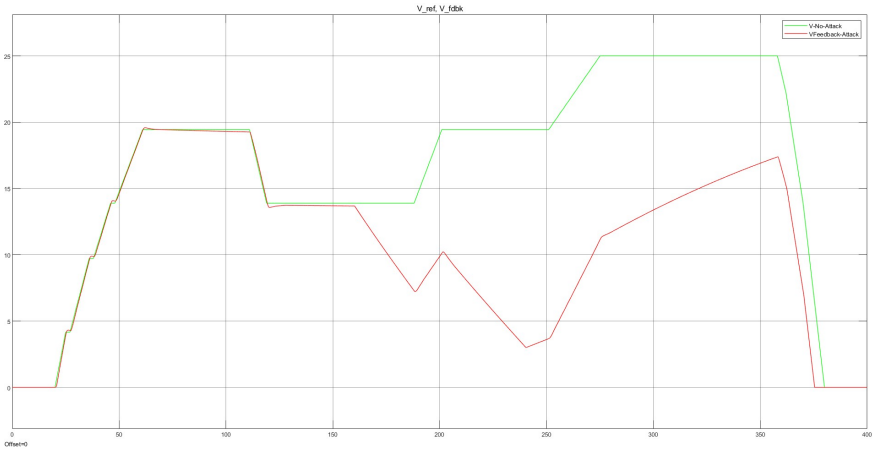


Figure 4.44: Velocity Trajectory with and without the attack.

As shown in Figure 4.44, as in the case of the torque, both signals had the same trend until around t=160s when the attack took place. We can see that the vehicle started decelerating instead of maintaining a constant velocity and could not follow up on the requested velocity trajectory values. This is due to the braking torque we forced on it while the vehicle tried accelerating to follow the velocity trajectory.
Now we can observe the Messages and the signals and try to capture the attack from the messages in the CAN. To do that, we use the timetable mentioned before in Figure 4.31. To do that, we access the "msgs" and "sigs" variables in the workspace. In the "msgs", we will find a timetable that contains the ID of the messages, whether it is an extended ID or not, the Name of the message, the Data contained in the message, the length of the Data, the signals if it has an Error, and finally if it's a remote frame or not as shown in Figure 4.45.

Time	1 ID	2 Extended	3 Name	4 Data	5 Length	6 Signals	7 Error	8 Remote
9.845 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0
9.8451 sec	2	0	'Velocity_Reference'	[0,0,0,0,0,192,54,64]	8	*1x1 struct*	0	0
9.8451 sec	1	0	'Velocity_Feedback'	[40,171,103,228,2,120,...	8	*1x1 struct*	0	0
9.8451 sec	0	0	'Motor_Torque'	[88,0,0,0,0,0,0,0]	8	*1x1 struct*	0	0
9.849 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0
9.849 sec	2	0	'Velocity_Reference'	[86,85,85,85,85,85,213,54...	8	*1x1 struct*	0	0
9.849 sec	1	0	'Velocity_Feedback'	[248,130,18,19,185,14...	8	*1x1 struct*	0	0
9.8491 sec	0	0	'Motor_Torque'	[88,0,0,0,0,0,0,0]	8	*1x1 struct*	0	0
9.8526 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0
9.8526 sec	2	0	'Velocity_Reference'	[171,170,170,170,170,...	8	*1x1 struct*	0	0
9.8526 sec	1	0	'Velocity_Feedback'	[158,206,110,178,110,...	8	*1x1 struct*	0	0
9.8526 sec	0	0	'Motor_Torque'	[88,0,0,0,0,0,0,0]	8	*1x1 struct*	0	0
9.8564 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0
9.8564 sec	2	0	'Velocity_Reference'	[0,0,0,0,0,0,55,64]	8	*1x1 struct*	0	0
9.8564 sec	1	0	'Velocity_Feedback'	[202,149,141,189,35,1...	8	*1x1 struct*	0	0
9.8565 sec	0	0	'Motor_Torque'	[87,0,0,0,0,0,0,0]	8	*1x1 struct*	0	0
9.86 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0
9.86 sec	2	0	'Velocity_Reference'	[85,85,85,85,85,21,55,...	8	*1x1 struct*	0	0
9.86 sec	1	0	'Velocity_Feedback'	[65,97,128,47,216,210,...	8	*1x1 struct*	0	0
9.86 sec	0	0	'Motor_Torque'	[87,0,0,0,0,0,0,0]	8	*1x1 struct*	0	0
9.8754 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0
9.8754 sec	2	0	'Velocity_Reference'	[171,170,170,170,170,...	8	*1x1 struct*	0	0
9.8754 sec	1	0	'Velocity_Feedback'	[48,178,189,71,87,233,...	8	*1x1 struct*	0	0
9.8754 sec	0	0	'Motor_Torque'	[87,0,0,0,0,0,0,0]	8	*1x1 struct*	0	0
9.8798 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0
9.8799 sec	2	0	'Velocity_Reference'	[0,0,0,0,0,64,55,64]	8	*1x1 struct*	0	0
9.8799 sec	1	0	'Velocity_Feedback'	[59,55,215,99,161,255,...	8	*1x1 struct*	0	0
9.8799 sec	0	0	'Motor_Torque'	[87,0,0,0,0,0,0,0]	8	*1x1 struct*	0	0
9.884 sec	5	0	'SOC'	[0,0,0,0,0,0,53,64]	8	*1x1 struct*	0	0

Figure 4.45: Timetable.

Inside the "sigs" variable, we will find the signals' values that are stored in "Signals" in the timetable for each message; we access it, filter out only the signal related to torque, and then plot the signal and observe where the attack took place in the CAN time-frame.

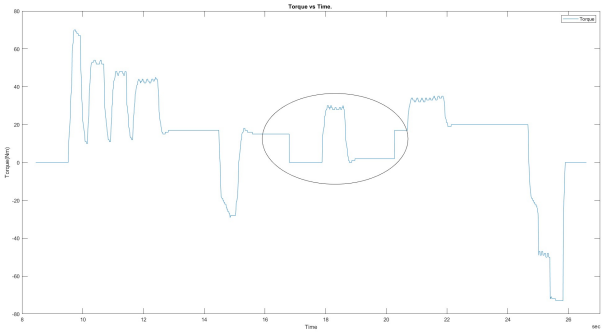
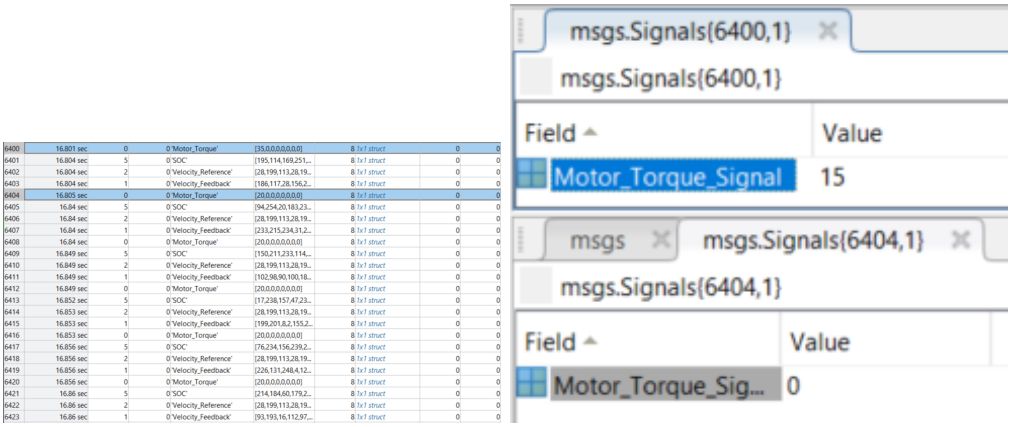


Figure 4.46: Plot of the CAN Torque signal using Matlab's timetable to identify when the attack occurred.

As seen in Figure 4.46, Inside the ellipse drawn is where the attack took place, around t=16.8s in this graph; this was expected as the sample time chosen was 0.1. Now we can go to t=16.8s in the messages of the Torque mentioned before in 4.45, and there we can find the messages relative to the attack.



(a) Messages when the attack starts. **(b)** Torque signal value drop.

In figure 4.47a, we marked with blue where the attack starts if we go and check the values of the signal at those two instants, we will find that the torque value dropped directly from 15 Nm to 0 Nm (4.47b) when it should have continued to be constant at 15 Nm, and this is because of the attack signal that forced a negative 15 Nm torque causing the vehicle to decelerate instead of maintaining a constant velocity as we have seen in Figure 4.44.

4.10.2 Second Attack

In the second example of an attack, we will simulate an attack while the vehicle is in cruise mode at 100 Km/h, as shown in Figure 4.41.

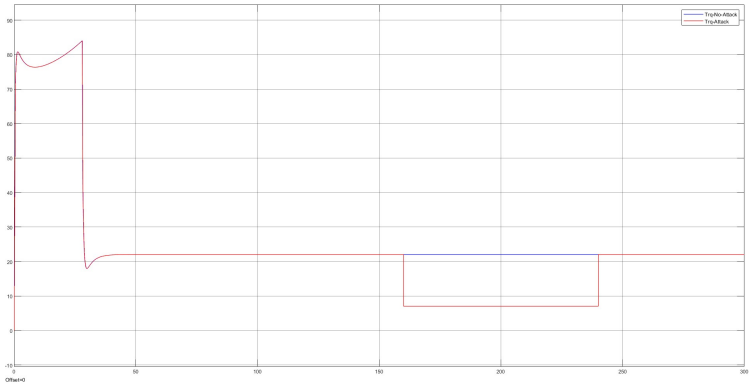


Figure 4.48: Torque without any attack vs Torque with an attack during Cruise Mode.

Also, this time, the attack signal was created, so it takes place at time t=160s and up to t=240s, and this is also clear by looking at the graph comparison between the Torque with and without any attack, From t=0s up to t=160s since no attack was taking place both the torque signal without attack represented in blue. The torque signal with attack represented with yellow had the same values. At t=160 seconds, the torque in blue had a constant value of around 22 Nm; with the attack applied, the graph shifted downstairs by -15 Nm, the same magnitude as the signal created. We can now observe the plot of the velocity present in Figure 4.49 to notice its effect.

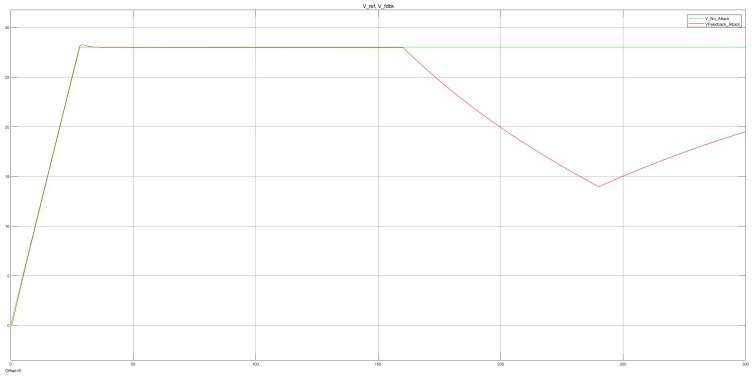


Figure 4.49: Velocity Trajectory with and without the attack.

As shown in Figure 4.49, as in the case of the torque, both signals had the same trend until around t=160s when the attack took place. The car accelerates until it reaches cruise mode at around t=30s, where it should maintain a constant velocity of 100 Km/h. However, as the attack takes place at t=160s, the car suddenly starts to decelerate undesirably until t=240s and loses its trajectory of the cruise mode. As soon as the attack was stopped, the vehicle started accelerating again, trying to reach the target velocity but needed more time.

This is due to the braking torque we forced on it while the vehicle tried to keep a constant velocity.
Now we can observe the Messages and the signals and try to capture the attack from the messages in the CAN using the variables "msgs" and "sigs" in MATLAB's Workspace as explained before in the first attack example.

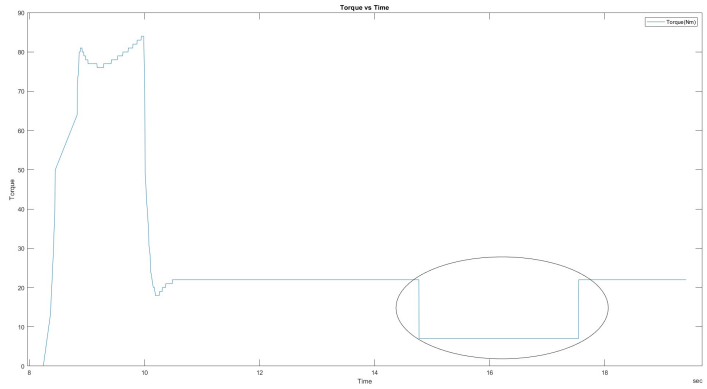
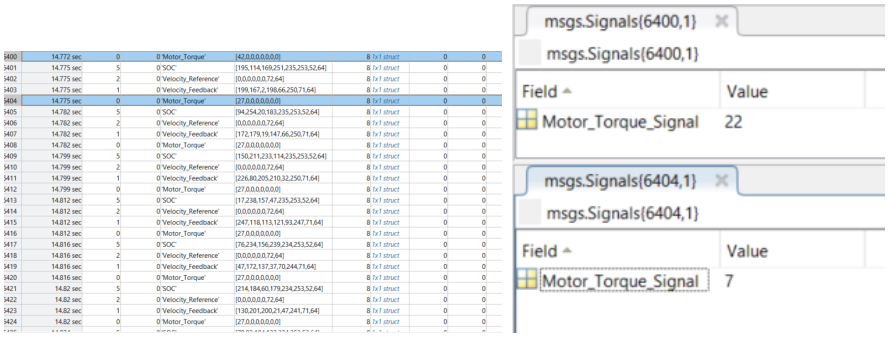


Figure 4.50: Plot of the CAN Torque signal using Matlab's timetable to identify when the attack occurred.

As seen in Figure 4.50, Inside the ellipse drawn is where the attack took place, around t=14.8s in this graph. Now, we can go to t=14.8s in the messages of the Torque, and there, we can find the messages relative to the attack.



(a) Messages when the attack starts. **(b)** Torque signal value drop.

In Figure 4.51a, we marked with blue where the attack starts; if we go and check the values of the signal at those two instants, we will find that the torque value dropped directly from 22 Nm to 7 Nm when it should have continued to be constant at 22 Nm, and this is because of the attack signal that forced a negative 15 Nm torque, causing the vehicle to decelerate instead of maintaining a constant velocity and losing the cruise mode.

Chapter 5

Conclusion and Future Work

In this thesis, we extensively explored BEVs and their CAN communication systems, aiming to bridge the gap between vehicle modelling and cybersecurity. Through a systematic approach, we developed a detailed Simulink model of a BEV, capturing the intricate interactions between its components and simulating its behaviour under various driving conditions; we delved into the theoretical framework of CAN, shedding light on its architecture, message protocols, and security vulnerabilities. Additionally, we explored the theoretical framework of CAN, discussing its architecture, message protocols, and security vulnerabilities. During my research for this thesis, We learned several important concepts. Thanks to the vast literature online, such as the importance of understanding the holistic dynamics of BEVs, from battery management to drivetrain operation, to accurately model their performance and energy consumption. Secondly, through a literature review of previous CAN various attacks, we highlighted the vulnerabilities inherent in CAN communication networks, illustrating the potential risks posed by different CAN cyberattacks.

Despite the progress made, our study also encountered limitations. While necessary for simulation purposes, the simplifications made in the BEV model may have overlooked specific nuances present in real-world vehicles. Similarly, while informative, the simulated CAN attacks may not fully capture the complexity and diversity of potential cyber threats faced by modern automotive systems.

Looking ahead, numerous avenues for future research warrant exploration. Enhanced BEV modelling techniques with CAN incorporated by modelling more accurately the features of the BEV model, such as battery management system and motor control, while also performing different types of CAN attacks could further refine the accuracy of simulations. Moreover, it could also be interesting to try incorporating intrusion detection systems into the model by leveraging machine learning algorithms and real-world validation, which promise to mitigate cyber threats and safeguard vehicle integrity.

In conclusion, this thesis represents a stepping stone towards a deeper understanding of BEV modelling and CAN security in the context of modern automotive systems. By addressing the challenges and opportunities identified herein, we can pave the way for safer, more efficient, and more resilient electric vehicles in the future.

Bibliography

[1] Brooke Crothers. *Tesla Model 3 Now Among Cheapest EVs With Price Cut And Tax Credit*. Accessed: January, 2024. 2023. URL: https://www.forbes.com/sites/brookecrothers/2023/06/04/tesla-is-doubling-down-on-model-3-discounts-creating-problems-for-ev-rivals/?sh=4dc6465b9057 (cit. on p. 14).

[2] Sujith. *Global Electric(EV) Market*. Accessed: January, 2024. 2023. URL: https://www.blackridgeresearch.com/reports/global-electric-vehicle-market (cit. on p. 14).

[3] Unkown. *How Do All-Electric Cars Work?* Accessed: January, 2024. 2023. URL: https://afdc.energy.gov/vehicles/how-do-all-electric-cars-work (cit. on p. 16).

[4] W. Lawrenz. *CAN System Engineering: From Theory to Practical Applications*. SpringerLink : Bücher. Springer London, 2013. ISBN: 9781447156130. URL: https://books.google.it/books?id=IUy6BAAAQBAJ (cit. on p. 17).

[5] Yuri Vershinin and M. Lorch. «Automotive Telemetry System based on Controller Area Network». English. In: *Proceedings of the First international Mobile-Machine Conference (MMC-2013)*. Germany: CAN in Automation (CiA), 2013, pp. 06/14–06/20 (cit. on p. 17).

[6] Karl Henrik Johansson, Martin Törngren, and Lars Nielsen. «Vehicle Applications of Controller Area Network». In: *Handbook of Networked and Embedded Control Systems*. Ed. by Dimitrios Hristu-Varsakelis and William S. Levine. Boston, MA: Birkhäuser Boston, 2005, pp. 741–765. ISBN: 978-0-8176-4404-8. DOI: 10.1007/0-8176-4404-0_32. URL: https://doi.org/10.1007/0-8176-4404-0_32 (cit. on p. 17).

[7] Texas Instruments. *Report: Introduction to the Controller Area Network (CAN)*. Tech. rep. SLOA101B. Revised May 2016. Texas Instruments, 2002. URL: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf?ts=1705204850408 (cit. on pp. 19, 22).

[8] Thomas Huybrechts, Yon Vanommeslaeghe, Dries Blontrock, Gregory Van Barel, and Peter Hellinckx. «Automatic Reverse Engineering of CAN Bus Data Using Machine Learning Techniques». In: Jan. 2018, pp. 751–761. ISBN: 978-3-319-69834-2. DOI: 10.1007/978-3-319-69835-9_71 (cit. on p. 20).

[9] Wikipedia. *CAN-Bus*. 2024. URL: https://en.wikipedia.org/wiki/CAN_bus#References (cit. on p. 19).

[10] GalileoSky. *Using CAN BUS - GalileoSky*. 2023. URL: https://base.galileosky.com/articles/#!en-documentation/using-can-bus (cit. on p. 20).

[11] Wei Ng, Chee Kyun Ng, Nor Noordin, and Borhanuddin Ali. «Performance Analysis of Wireless Control Area Network (WCAN) Using Token Frame Scheme». In: *Proceedings - 3rd International Conference on Intelligent Systems Modelling and Simulation, ISMS 2012* (Feb. 2012). DOI: 10.1109/ISMS.2012.74 (cit. on p. 21).

[12] Manuel Domínguez-Morales, Angel Jiménez-Fernandez, Rafael Paz-Vicente, Alejandro Linares-Barranco, Daniel Cascado-Caballero, Juan Coronado, J. Muñoz, and Gabriel Jimenez. «An AER to CAN Bridge for Spike-Based Robot Control». In: vol. 6691. June 2011, pp. 124–132. ISBN: 978-3-642-21500-1. DOI: 10.1007/978-3-642-21501-8_16 (cit. on p. 24).

[13] Professor Cibrario Bertolotti Ivan. *Networking technologies for connected vehicles*. Lecture Notes of the course "Networking technologies for connected vehicles" in the first semester of the 2nd year. 2023. URL: https://www.polito.it/ (cit. on p. 25).

[14] Door Lee H. Goldberg. *CAN Bus: Taking a Larger Role in EVs and PHEVs*. Accessed: January, 2024. 2012. URL: https://www.digikey.be/nl/articles/can-bus-taking-a-larger-role-in-evs-and-phevs (cit. on p. 27).

[15] The MathWorks Inc. *Vehcile Body*. Natick, Massachusetts, United States, 2023. URL: https://it.mathworks.com/help/sdl/ref/vehiclebody.html?searchHighlight=vehicle%20body&s_tid=srchtitle_support_results_1_vehicle%20body (cit. on pp. 28, 29, 31).

[16] EV Specifications. *2019 Tesla Model 3 Standard Range Plus RWD- Specifications and Price*. Accessed: January, 2024. 2024. URL: https://www.evspecifications.com/en/model/bbc397 (cit. on pp. 30, 34).

[17] The MathWorks Inc. *Tre Magic Formula*. Natick, Massachusetts, United States, 2023. URL: https://it.mathworks.com/help/sdl/ref/tiremagicformula.html#References (cit. on p. 32).

[18] Professor Alessandro Vigliani. *Tyre Mechanics*. Lecture Notes of the course "Technologies for Autonomous Vehicles" in the second semester of the 2nd year. 2023. URL: https://www.polito.it/ (cit. on p. 33).

[19] The MathWorks Inc. *Ideal Torque Sensor*. Natick, Massachusetts, United States, 2023. URL: https://www.mathworks.com/help/simscape/ref/idealtorquesensor.html#bqji4pp-1_seealso (cit. on p. 35).

[20] David McDonald. «Electric Vehicle Drive Simulation with MATLAB/Simulink». In: 2012 (cit. on pp. 35, 36).

[21] AMIT JHA. *Let's discuss motors in Electric vehicles continued.....* 2024. URL: https://etn-demeter.eu/lets-discuss-motors-in-electric-vehicles-continued/ (cit. on p. 37).

[22] The MathWorks Inc. *PMSM*. Natick, Massachusetts, United States, 2023. URL: https://it.mathworks.com/help/sps/ref/pmsm.html (cit. on pp. 37, 41).

[23] Kim Sang-Hoon. *Electric Motor Control*. Elsevier, 2017, pp. 203–246 (cit. on pp. 38, 40).

[24] Aykut Bıçak and Ayetül Gelen. «Modified Super-Twisting Algorithm-Based Model Reference Adaptive Observer for Sensorless Control of the Interior Permanent-Magnet Synchronous Motor in Electric Vehicles». In: *Machines* 11.9 (2023), p. 871 (cit. on p. 41).

[25] The MathWorks Inc. *Motor and Drive (System Level)*. Natick, Massachusetts, United States, 2023. URL: https://www.mathworks.com/help/sps/ref/motordrivesystemlevel.html (cit. on p. 43).

[26] The MathWorks Inc. *Longitudinal Driver*. Natick, Massachusetts, United States, 2024. URL: https://www.mathworks.com/help/vdynblks/ref/longitudinaldriver.html (cit. on p. 45).

[27] The MathWorks Inc. *1-D Look Up Table*. Natick, Massachusetts, United States, 2023. URL: https://www.mathworks.com/help/simulink/slref/1dlookuptable.html (cit. on p. 46).

[28] The MathWorks Inc. $HV_battery_Charge_Discharge$. Natick, Massachusetts, United States, 2023. URL: https://it.mathworks.com/help/sps/ug/hv-battery-charge-discharge.html (cit. on p. 51).

[29] The MathWorks Inc. *CAN Configuration Block*. Natick, Massachusetts, United States, 2024. URL: https://www.mathworks.com/help/vnt/ug/canconfiguration.html (cit. on p. 55).

[30] The MathWorks Inc. *CAN Pack Block*. Natick, Massachusetts, United States, 2024. URL: https://www.mathworks.com/help/vnt/ug/canpack.html (cit. on pp. 55, 56).

[31] The MathWorks Inc. *CAN Trasmit Block*. Natick, Massachusetts, United States, 2024. URL: https://www.mathworks.com/help/supportpkg/raspberrypi/ref/cantransmit.html (cit. on p. 56).

[32] The MathWorks Inc. *CAN Receive Block*. Natick, Massachusetts, United States, 2024. URL: https://www.mathworks.com/help/vnt/ug/canreceive.html (cit. on pp. 56, 57).

[33] The MathWorks Inc. *CAN Unpack Block*. Natick, Massachusetts, United States, 2024. URL: https://www.mathworks.com/help/vnt/ug/canunpack.html (cit. on p. 58).

[34] Tobias Hoppe and J. Dittman. «Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy». In: *Workshop on Embedded Systems Security* (Jan. 2007), pp. 66–72 (cit. on p. 60).

[35] Karl Koscher et al. «Experimental Security Analysis of a Modern Automobile». In: Jan. 2010, pp. 447–462. DOI: 10.1109/SP.2010.34 (cit. on p. 60).

[36] Samuel Woo, Hyo Jo, and Dong Lee. «A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN». In: *IEEE Transactions on Intelligent Transportation Systems* 16 (Sept. 2014), pp. 1–14. DOI: 10.1109/TITS.2014.2351612 (cit. on p. 61).

[37] Stephen Checkoway et al. «Comprehensive Experimental Analyses of Automotive Attack Surfaces». In: Aug. 2011 (cit. on p. 61).

[38] Charlie Miller Chris Valasek. «A Survey of Remote Automotive Attack Surfaces». In: (2014). URL: `https://ioactive.com/pdfs/IOActive_Remote_Attack_Surfaces.pdf` (visited on 02/10/2024) (cit. on p. 61).

[39] WIRED. *Hackers Remotely Kill a Jeep on a Highway | WIRED*. 2015. URL: `https://youtu.be/MK0SrxBC1xs?si=BiQyTw0eqP3dsA6Q` (visited on 02/10/2024) (cit. on p. 61).