POLITECNICO DI TORINO

Department of Electronics and Telecommunications

Master's degree programme in
ICT For Smart Societies

Master Thesis

Development of a framework for sensor- and communication- assisted vehicle dynamic

Supervisor
prof. Carla Fabiana Chiasserini

Candidate
Dinesh Cyril Selvaraj

APRIL 2020

Faith, Hope, Love

Acknowledgements

First and foremost, the biggest Thanks to my supervisor, mentor and guide prof. Carla Fabiana Chiasserini. Throughout the whole period of working on this thesis she was always available and very quick in responding to any of my queries while providing valuable insights at each step of the way. She let me be independent, yet gently guided me at the same time. I will always be grateful for the amount of effort she put in for my Thesis and the trust she showed in me.

I am grateful for all the professors of Inter-department Research Centre "CARS@Polito", prof. Nicola Amati, prof. Francesco Paolo Deflorio for their guidance throughout all the long meetings.

A special Thank you to Shailesh Hedge, Ph.D. student from CARS@Polito team, whose technical and moral support was highly valuable and essential in the completion of this Thesis.

I would also like to thank all my colleagues and friends from ICT for smart Societies who made my experience throughout the course of the Master's not only intellectually rewarding but also thoroughly enjoyable.

And last, but by no means least, I would like to thank my family, my cousins for their everlasting encouragement and support which gave me the strength needed to complete not just this Thesis, but also my Master's degree.

Contents

List of Tables V

List of Figures VI

1 Introduction 1

2 State of the Art 5
2.1 Vehicular Communication Standards 6
2.1.1 Long Term Evolution (LTE) 7
2.1.2 Dedicated Short Range Communication (DSRC) 12
2.1.3 V2X Messages 16
2.2 Vehicle Dynamics 19
2.3 Control Strategies 21
2.3.1 Adaptive Cruise Control 22
2.3.2 Automatic Emergency Braking 23
2.3.3 Collision Avoidance 24
2.3.4 Co-operative Adaptive Cruise Control 24
2.4 Literature Review 25
2.5 Simulation Tools 30
2.5.1 Network Simulator 31
2.5.2 CarMaker 33

3 Co-Simulation Framework Architecture 38
3.1 Architecture Feasibility Study 38
3.1.1 Feasibility Study 1 39
3.1.2 Feasibility Study 2 40
3.2 System Architecture 41
3.3 System Definition 43
3.3.1 CarMaker: Vehicle and Mobility Model 43
3.3.2 Simulink: Simulation Control and Interface 45
3.3.3 ns3: Communication Model 48
3.4 Co-Simulation Interface 49

3.4.1 Interaction between CarMaker and Simulink 49
3.4.2 Interaction between Simulink and MATLAB 52
3.4.3 Interaction between MATLAB and Python Engine 52
3.5 Information Flow 53
3.6 Control Strategy 55

4 Simulation Scenarios and Results 61
4.1 Simulation Scenarios 61
4.1.1 Scenario-1: Vehicles crossing in a T-Junction 62
4.1.2 Scenario-2: Preceding vehicle Cut-out from the lane 63
4.1.3 Scenario-3: Slower/Stationary Vehicle in the Corner 64
4.2 Results 66
4.2.1 Scenario 1 66
4.2.2 Scenario 2 69
4.2.3 Scenario 3 71
4.2.4 General Discussion 74

5 Conclusion 76

Bibliography 78

List of Tables

2.1 GSM,UMTS,LTE Comparison 7
2.2 Comparison between IEEE 802.11a and IEEE 802.11p [13] 14
3.1 Data extracted from each vehicle and their Importance 42
3.2 CarMaker Variable Dictionary [30] 50
3.3 Ego Car Action based on Acceleration sign 55
3.4 ACC Gain and Acceleration Limits [27] 57
4.1 Comparison of basic quantities between with and without OBU modes . 74

List of Figures

2.1 V2V, V2I, and V2X communications [7] 6
2.2 LTE Architecture 7
2.3 ProSe Communication Scenarios 9
2.4 LTE ProSe Architecture [9] 9
2.5 Scenario 1 V2I Operation 11
2.6 Scenario 2 V2I Operation 11
2.7 Scenario 3 11
2.8 DSRC Communication [11] 12
2.9 WAVE Protocol Stack [12] 13
2.10 CAM Message Structure 17
2.11 DENM Message Structure 18
2.12 Basic representation of Vehicle Dynamics 19
2.13 Longitudinal Vehicle Model 20
2.14 RADAR Types: LDR- Long Range RADAR; MDR-Mid Range RADAR; SDR-Short Range RADAR [16] 22
2.15 ACC Working Model 23
2.16 CACC Functional Elements [19] 25
2.17 Interaction between ns3 and MATLAB [20] 26
2.18 Simulation Experimental Structure [22] 27
2.19 ns3 and SUMO Interaction through MATLAB [23] 28
2.20 ns3 and VISSIM Interaction through MATLAB [24] 29
2.21 CarMaker GUI 33
2.22 CarMaker Traffic Window 34
2.23 CarMaker Scenario Editor Window 35
2.24 CarMaker IPGInstruments Window [27] 35
2.25 CarMaker IPGMovie Window [27] 36
2.26 CarMaker for Simulink [29] 37
3.1 Basic Framework Architecture 38
3.2 Framework Architecture with SUMO 39
3.3 Framework Architecture with EAI 40
3.4 Co-Simulation Framework Architecture 41
3.5 RADAR Observation Area [30] 43

3.6 CarMaker RADAR selection window 44
3.7 CarMaker Driver Window 44
3.8 Simulink Simulation Control Subsystem 46
3.9 Simulink Matrix Concatenate 47
3.10 Simulink TCP/IP Subsystem 47
3.11 CarMaker Read block in Simulink 50
3.12 CarMaker Write block in Simulink 51
3.13 CarMaker ACC Control Model 51
3.14 System User Interface 54
3.15 Information Flow 55
3.16 ACC Control Scheme [27] 57
3.17 Distances Representation in Control Algorithm 58
3.18 ACI Safety Distance [27] 58
3.19 Trend of Variable Proportional Gains 59
3.20 Control Algorithm Flow Chart 60
4.1 CarMaker Scenario Editor 62
4.2 Lead Car Manoeuvres 63
4.3 Pictorial Representation of Cut-Out Scenario 64
4.4 Lead Car Cut-Out Manoeuvre in CarMaker 64
4.5 Road Topology of Scenario3 65
4.6 Lead Car Manoeuvre in CarMaker 65
4.7 Scenario1: Ego Car Acceleration and Velocity variation based on RADAR and CAM Messages 66
4.8 Scenario1: ACC vs CACC Comparision 68
4.9 Scenario1: Acceleration Variation for different Prediction Time Instances 68
4.10 Scenario2: Ego Car Acceleration and Velocity variation based on RADAR and CAM Messages 70
4.11 Scenario2: ACC vs CACC Comparision 70
4.12 Scenario2: Acceleration Variation for different Prediction Time Instances 71
4.13 Scenario3: Ego Car Acceleration and Velocity variation based on RADAR and CAM Messages 72
4.14 Scenario3: ACC vs CACC Comparision 72
4.15 Scenario3: Acceleration Variation for different Prediction Time Instances 73

Chapter 1

Introduction

Nowadays almost half of the world population is living in the cities and it is expected to rise to 5 billion in 2030 [1]. To deal with the challenges brought by rapid urbanization, cities are turning towards technology and becoming Smart Cities. A smart city uses ICT solutions as a means to solve its sustainability challenges. Smart cities have a wide range of applications across multiple fields such as buildings, transportation, health, and public safety.

The technological advancements and research in the transportation sector are collectively called as Intelligent Transport System (ITS). Traveller information system, road pricing, Adaptive traffic signal control, Intelligent speed adaptation are some of the applications of ITS. Among them, a new domain is created to handle cooperative, connected and automated mobility called Cooperative Intelligent Transport Systems(C-ITS).

In today's world, modern vehicles are already well connected with highly advanced infotainment systems. With the increase of in-vehicle entertainments safety concerns also began to increase due to distracted driving. Nearly 1.25 million people die in road crashes each year. It is estimated that road traffic injuries will be the fifth leading cause of death by 2030 [2]. To fight these challenges, countries have started research programs to find a way to reduce the fatalities caused by traffic accidents. Projects like Vision Zero proposed by Swedish road safety aims at reducing road deaths to zero by 2050. As an intermediate target, the European Union focuses to reduce road deaths by 50% in 2020. However, based on the 2018 data published by The European Transport Safety Council (ETSC)[3], we can see a major deviation between desired and actual reduction in deaths related to road traffic. It is believed that C-ITS can help to revive a positive dynamic in the reduction of road fatalities. Therefore, C-ITS focuses on extending the in-vehicle connectivity to the next level by sharing the vehicle information

with other vehicles and infrastructure to coordinate their movements. These co-ordinated movements can improve road safety, traffic efficiency by helping the driver to make the right decisions and adapt to traffic situations before-hand. In terms of economy, C-ITS implementation helps government/organisations/individuals to save a lot of money by reducing road accidents and increasing traffic efficiency. As per annual global road crash statistics [2], road crashes cost USD 518 billion globally, costing individual countries from 1-2% of their annual GDP. Traffic congestions also take a major toll on the global economy in terms of wasted time and fuel. Based on recent traffic index posted by Tom-Tom [4] covering 416 cities across 57 countries on 6 continents, 239 cities show a considerable increase in traffic in 2019 compared to 2018 whereas only 63 countries show a reduction in road traffic. As per Inrix Annual Global Traffic Scorecard [5], UK drivers lost an average of 178 hours a year due to congestion, costing them £7.9 billion in 2018.

To support C-ITS, the automotive industry also started integrating new technologies in their vehicles to support autonomous features like Adaptive Cruise Control, Lane Assist System, collision avoidance, only to mention some. Vehicles are now equipped with precision positioning systems which help drivers to navigate through the cities by considering real-time traffic information. To support the autonomous features, vehicles are now loaded with various object sensors like Radar, LIDAR, Camera and Ultrasonic sensors. With the help of these sensor inputs and their corresponding algorithms, Advanced Driver Assistance Systems (ADAS) help the drivers to react to their surroundings. However, these sensors won't be able to detect objects which are present beyond their horizon range especially in a blind bend corner or in the presence of buildings or large vehicles blocking the field of view. C-ITS comes to the rescue by implementing communication between vehicles, infrastructure, and other road users. Vehicle-to-Everything (V2X) communication complements these sensors and helps the vehicle/driver to see beyond their line-of-sight.

V2X communications help to exchange useful information such as speed, heading, position with other vehicles (Vehicle-to-Vehicle) and/or infrastructures (Vehicle-to-Infrastructure). These data points are used by algorithms to detect possible collisions in the future and send appropriate corrective actions to the vehicles to avoid them. Some of the applications of V2X communications are Lane change assist, Stationary/Slow vehicle detection, Intersection Movement Assist, Cooperative Adaptive Cruise Control. There are two main types of technologies used in Vehicular Communications. The first one is the Dedicated Short Range Communication (DSRC) standard which is based on IEEE 802.11p wireless communication technology. IEEE 802.11p does not require a cellular

coverage, it uses onboard units (OBUs) and road-side units (RSUs) to exchange information between vehicles and infrastructure. The second one is cellular network-based using Long Term Evolution(LTE) as a communication standard. The main aspect of vehicular communications is latency. For safety applications like forward collision avoidance, the latency should be less than 100ms whereas for latency tolerant safety messages which concerns long-range traffic conditions like Cooperative Adaptive Cruise Control latency can be as long as 1 second [6]. Multiple studies have been performed to analyse the performance of both standards for safety-related applications.

As we can see C-ITS involves entities from various disciplines such as traffic system, communication framework, vehicle dynamics and their sensors and infrastructure to process the data points. Due to the involvement of multidisciplinary entities and their design parameters, implementation of a comprehensive C-ITS systems becomes very complex. Since C-ITS plays a major role in safety related applications, any issues in the deployment may lead to loss of life. To ensure the working of different functionalities, it is suggested to test the full system in a virtual environment i.e. Simulator. Additionally, simulator helps us to identify design related issues in the early stages which could probably save millions if issues are identified at the deployment stage. Simulators are used in many fields including Vehicle Dynamics (CarMaker, Carla), Network communication (ns3, OMNeT++), Traffic simulator (SUMO, Aimsun). Clearly, when it comes to testing the functionality of full C-ITS system, multiple simulators have to talk with each other which calls for a common framework to combine them.

The purpose of this thesis work is to create a co-simulation framework that represents all components of C-ITS: vehicle connectivity, vehicle dynamics, vehicle onboard sensors, vehicle traffic, and road scenarios. The framework uses Vehicle to Infrastructure(V2I) communication model to implement the Cooperative Adaptive Cruise Control(CACC) system by taking vehicle traffic and vehicle on-board sensors into consideration. The network infrastructure node periodically receives Cooperative Awareness Messages (CAM) from connected vehicles that contain information about each vehicle position, heading, acceleration, velocity. With the received information, the infrastructure node will run a trajectory-based collision avoidance algorithm which forecasts possible collision between vehicles. When a collision is detected, the node will send the updated acceleration to the Ego car. In this way, we can analyse the performance of ADAS and Communication models in safety-related applications with various traffic scenarios.

The outline that this thesis will follow is the following: The Chapter 2 covers the existing solutions for the V2X communication and ADAS related applications. Furthermore, a general review of related research in the co-simulation framework is provided. The Chapter 3 focuses on the co-simulation framework architecture, adopted

simulation tools and the description behind the coupling of adopted simulators In the Chapter 4, the simulation scenarios are presented, and an assessment of the results that have been obtained. Finally, the Chapter 5 closes drawing conclusions on the work and presenting some future developments.

Chapter 2

State of the Art

In recent years, technological innovations are playing a major role in enhancing day-to-day life events in multiple areas. In particular, the automotive industry is at the forefront among others if we consider the usage of new technologies in their products. Nowadays, vehicles are not only a mode of transport used for transiting from point A to point B. They are now equipped with a wide array of sensors providing multiple services for users ranging from safety-related to multimedia services. There have been multiple kinds of research carried out to improve the user experience in the automotive domain. With the constant increase in traffic congestion, researchers figured out that sensors in the vehicles are not sufficient to provide safety services. With the constant evolution of technology in the communication domain, multiple research works are carried to provide reliable communication between vehicles. Vehicular communication became the vital cog of the wheel to extend the sensing ability of vehicles beyond their horizon. To combine the data from vehicle sensors and communication. To provide safe travel, multiple control strategies have been developed to use the data collected through the communication framework and Vehicle sensors. In order to verify and improve the functionalities of this multidisciplinary system, researchers have also developed a few integrated simulators to represent the behaviour of Communication, Vehicle Dynamics, and Traffic scenarios in one combined simulation. In this section, we are going to review communication standards, vehicle dynamics and their control strategies, and research works related to the Co-Simulation Framework.

2.1 Vehicular Communication Standards

Vehicular communication systems are used to exchange information between vehicles and roadside units such as safety warnings and traffic information. Vehicular communication aims at reducing road accidents and increasing traffic efficiency. Among various types of vehicular communication methods, Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) used more. V2V involves exchanging information between vehicles whereas V2I allows vehicles to interact with infrastructures to exchange information between them. Vehicle-to-Everything (V2X) is a generalized representation of vehicular communications. Apart from V2V and V2I, V2X includes other methods of communication such as Vehicle-to-Pedestrian (V2P), Vehicle-to-Roadside (V2R), Vehicle-to-Device (V2D), and Vehicle-to-Grid (V2G). A broad representation of V2V, V2I, and V2X is shown in Figure 2.1. Dedicated Short-Range Communication (DSRC) and 4G-LTE are the widely adopted standards for the above-mentioned types of Vehicular communications. ETSI ITS-G5 is a standard adopted by the European Union for Vehicular Communications. It is an extension of existing communication standards, that has been modified and optimized for the dynamic automotive environment.

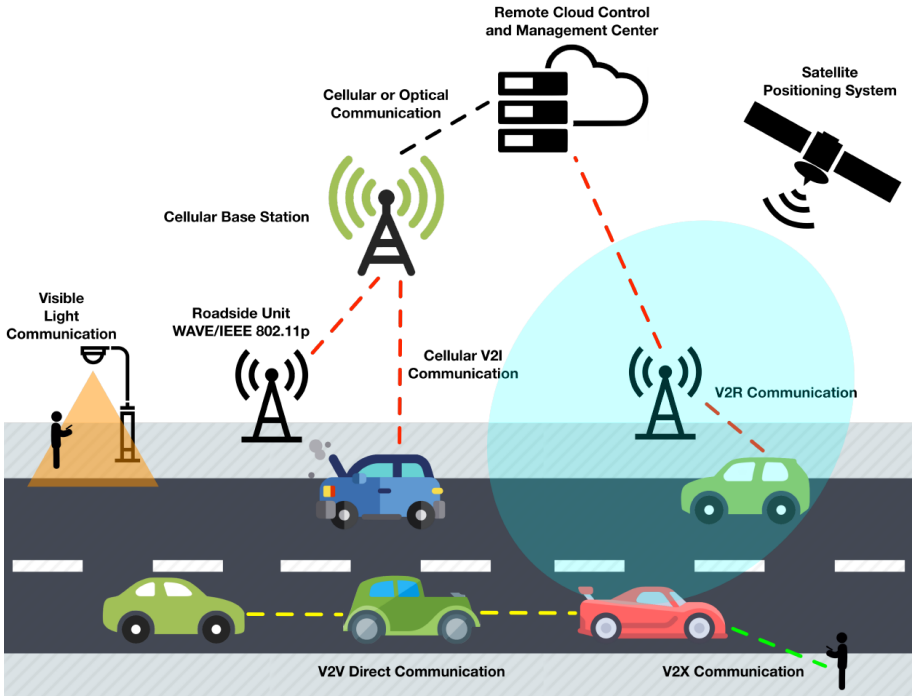

Figure 2.1. V2V, V2I, and V2X communications [7]

2.1.1 Long Term Evolution (LTE)

LTE is the fourth generation wireless communication standard designed and developed by 3rd Generation Partnership Project (3GPP). It is considered as the evolution of GSM (Global System for Mobile Communications) and UMTS (Universal Mobile Telecommunications System). These standards have different air interfaces and core network architecture.

Standard	Air Interface#1	Core Network#2
GSM	TDMA	Circuit Switching
UMTS	WCDMA	Packet(Data) and Circuit(Voice) Switching
LTE	OFDMA	Packet Switching (Data and Voice)

Table 2.1. GSM,UMTS,LTE Comparison

From Table 2.1, we can understand that cellular technologies are moving from voice-centric application to data-centric applications where voice is considered as one of the applications. Figure 2.2 shows the LTE architecture.

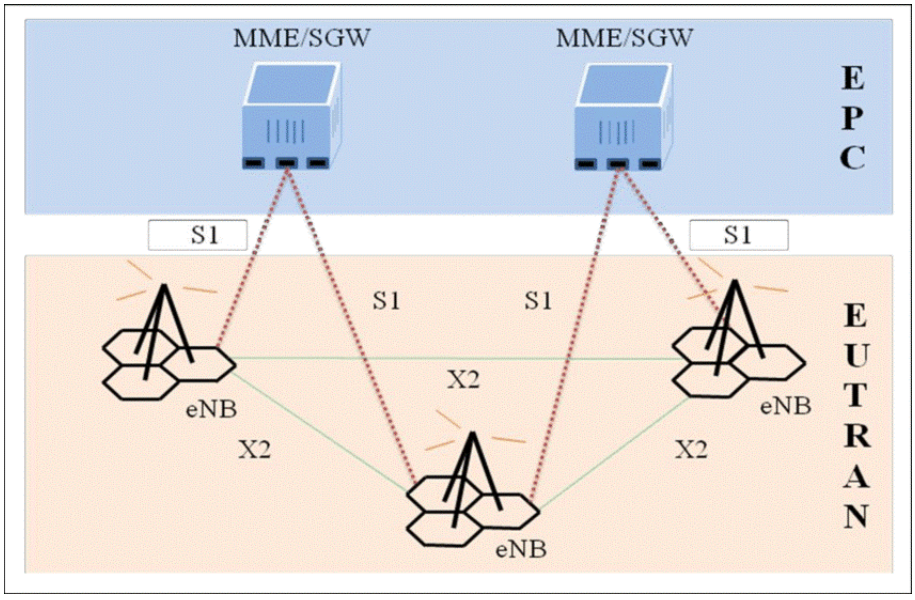

Figure 2.2. LTE Architecture

A standard LTE network consists of Evolved UMTS Terrestrial Radio Access Network (E-UTRAN) and System Architecture Evolution (SAE). SAE uses flat IP-based architecture which leads to the evolution of core network architecture and it is referred

to as Evolved Packet Core (EPC). EPC uses Internet Protocol (IP) as the key protocol to transport all services. This allows the network to handle data efficiently and cost-effectively.

The main units of EPC are MME, S-GW, P-GW, and HSS.

Home Subscriber Server (HSS) is a central database that contains user-related and subscriber-related information for user authentication and access authorization. It also handles functions related to mobility management, call and IP session setup.

Mobility Management Entity (MME) is responsible for handling control signals related to mobility and security for E-UTRAN. It is also responsible for tracking and paging idle UEs.

Serving Gateway (S-GW) handles user data traffic. It acts as an interconnection point between radio-side and EPC. S-GW serves UE by routing incoming and outgoing IP packets. It acts as an anchor for the UE when it moves from one eNodeB to another.

Packet Data Network Gateway (P-GW) helps to route packets between EPC and external IP networks (Packet Data Networks). It also allocates IP addresses to all UEs and enforces different policies based on their IP data traffic.

The E-UTRAN is comprised of UE's (user equipment) and eNodeB's (evolved-NodeB's). Its radio interface is called as E-UTRA, the Evolved Universal Terrestrial Radio Access. The eNodeB acts as a base station which helps UE to connect with the LTE network. eNodeB uses Uu, X2 and S1 interfaces to communicate with UE, other eNodeBs and EPC respectively. In the previous generation, NodeB's are controlled by a Central Radio Network Controller (RNC). The RNC is responsible for the allocation of radio resources to NodeB's and their mobility. In LTE, eNodeB has to handle all air interface communications, radio resource allocation, header compression, security, modulation, interleaving, handover and retransmission control. The absence of central controller and distributed radio control functions help LTE to provide a Round Trip Time theoretically lower than 10 ms, and transfer latency in the radio access up to 100 ms [8]. This is beneficial for safety-related applications especially vehicular communications.

Device to Device Communication in LTE

To support safety-critical communication systems, 3GPPP updated the LTE architecture to include the Proximity-Based-Services (ProSe) that enable direct communication between two UEs, also known as Device-to-Device (D2D) communication. Reserved LTE resources are used for D2D communication. It allows interaction between UEs nearby using a direct link instead of communicating with other UE using a base

station or the core network. Due to the reduced signal traverse path, it can achieve low latency requirements demanded by safety-related applications. ProSe introduced to work in places where coverage is not guaranteed. The figure 2.3 shows the ProSe communication scenarios [9].

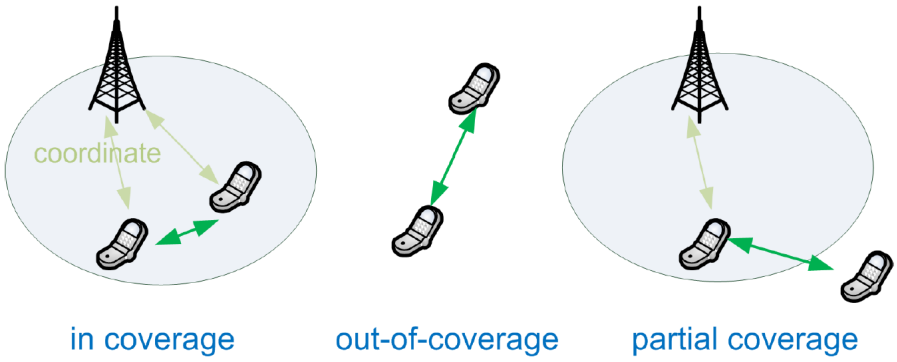

Figure 2.3. ProSe Communication Scenarios

In Coverage scenario, ProSe communication resources are allocated by the LTE network. To reduce the cellular traffic interference and optimize the ProSe communication, the network either assigns a pool of resources selected by the UE or assigns specific resources to the transmitting UE. In out of coverage scenarios, control from the network is not possible. However out-of-coverage doesn't mean there is no eNodeB in the range. There is a possibility of eNodeB presence from a different cellular network provider. In partial coverage scenario, with pre-configured values, out-of-coverage UE can communicate with UE in coverage where it uses the resources from the eNodeB.

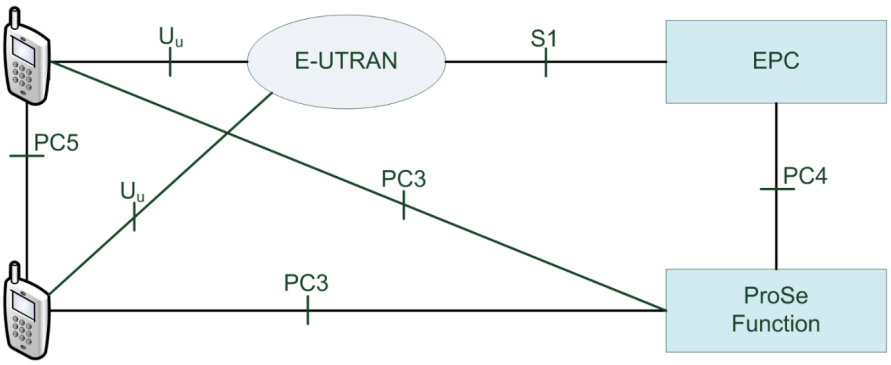

Figure 2.4. LTE ProSe Architecture [9]

Figure 2.4 shows interfaces used to communicate between LTE modules. When a UE wants to use the ProSe functionality, it should first contact ProSe function through the PC3 logical interface for authorisation and security paraments. After receiving security parameters through the PC3 interface, UE initiates discovery procedures to look for another ProSe enabled UE nearby using the PC5 interface. When two or more UEs have discovered each other, they can initiate a direct link between them. Like conventional cellular networks, usage of Uplink and Downlink to exchange data between UE and eNodeB is extended and adopted in ProSe communication to exchange information between UEs. The physical interface between ProSe enabled UE's is called SideLink (SL). Since it's a safety-related communication, SL transmissions are based on multicasting so, transmitters do not receive any feedback from receivers. ProSe communication was further enhanced in release 13 by introducing a multi-hop D2D network where UEs act as a relay between eNodeB and other UEs.

LTE based V2X

With D2D technology as a steppingstone, 3GPPP has introduced LTE-based Vehicle to Everything (V2X) service in its 14th release. User equipment involved in the V2X service could be pedestrian hand-held devices, Roadside Units (RSU), Infrastructure nodes or vehicles. V2X messages can be sent either using the Uu interface which represents the radio interface between eNodeB and UE or as direct communication between UE's using the PC5 interface. Using LTE-Uu interface, UE can either transmit/receive unicast V2X messages or transmit a unicast message to eNodeB and receive a broadcast message via Multimedia Broadcast Multicast Service (MBMS) delivery. Using PC5 UE can send/receive V2X messages using SideLink. Three main scenarios of LTE based vehicular services are discussed in the "Feasibility Study on LTE-based V2X Services" [10]

In the first scenario as shown in 2.5, V2X operations are handled only by the PC5 interface. That means D2D is the only way of exchanging messages. For V2I, either vehicle can transmit to RSU or RSU can transmit data to the group of vehicles using SideLink.

In the second scenario as shown in 2.6, the V2X operation can be handled only by the Uu interface. In this case, all the V2X messages are sent through eNodeB. For V2I, the vehicle sends a V2X message to eNodeB using Uplink where eNodeB takes care of sending the message to the destination UEs in the local area by downlink.

In the third scenario as shown in 2.7, it's a combination of the above two scenarios where it supports V2V operation using both Uu and PC5 interface. A vehicle

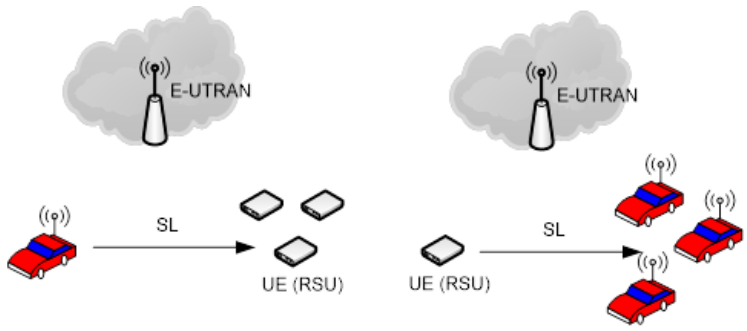

Figure 2.5. Scenario 1 V2I Operation

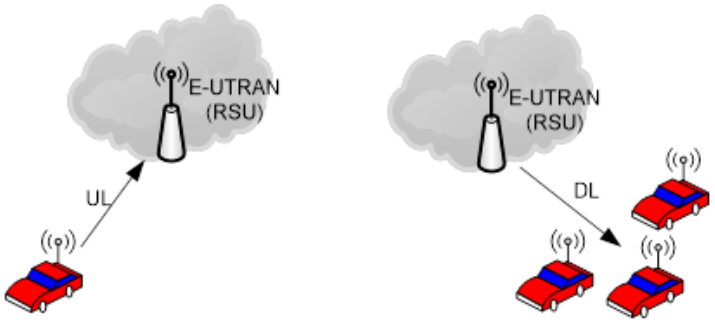

Figure 2.6. Scenario 2 V2I Operation

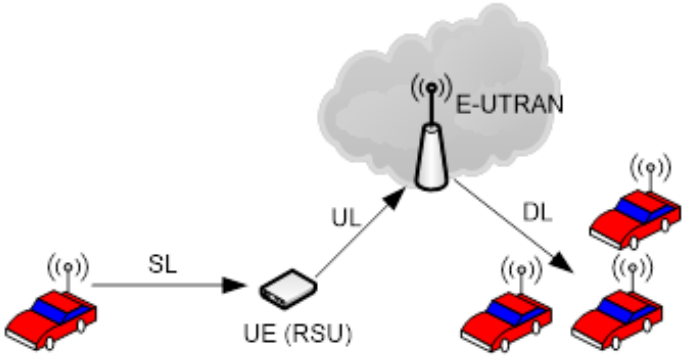

Figure 2.7. Scenario 3

UE sends a message to other RSU type UE using the PC5 interface. RSU UE sends the V2X message to E-UTRAN in an uplink. E-UTRAN receives the V2X message from the RSU UE and then transmits it to a group of UEs in close vicinity using downlink. In this scenario, E-UTRAN performs both uplink and downlink for V2X messages and downlink, E-UTRAN used a broadcasting mechanism.

The evolutions in LTE released by 3GPPP lead to considering LTE as a possible solution for safety-critical communications in the ITS field.

2.1.2 Dedicated Short Range Communication (DSRC)

Dedicated short-range communication (DSRC) is a wireless communication technology developed by the US Department of Transportation (USDOT) which plays a major role in the Intelligent transportation system (ITS). It is designed to support a vehicle to communicate with other vehicles and/or infrastructures. The characteristics of DSRC like low latency, secure, fast network acquisition, and handover, network reliability against interference in adverse weather conditions make its ideal for vehicular communications. DSRC can support both vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) type of communications. On-Board Units (OBU) acts as transponders in vehicles where Road-side Units (RSU) acts as fixed access points and usually permanently mounted along the roadside. Both units are equipped with a satellite positioning system which is mainly used for time synchronisation between devices. In V2V, DSRC allows vehicles to communicate with each other especially the exchange of safety-related messages. In V2I, a vehicle OBU communicates with surrounding infrastructure equipped with an RSU. This can be used to collect tolls, transferring multimedia, and also safety-related messages like road conditions. The basic functionalities handled by DSRC is shown in 2.8

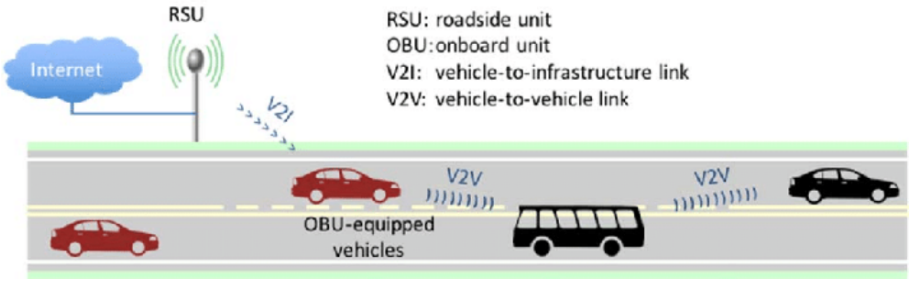

Figure 2.8. DSRC Communication [11]

The PHY and MAC layers of DSRC are defined by IEEE802.11p while DSRC upper layers are handled by the IEEE 1609 family of standards, which is commonly referred to as Wireless Access in Vehicular Environment (WAVE). The IEEE 802.11p standard is a part of the 802.11(Wi-Fi) family which is developed to support vehicular communications.

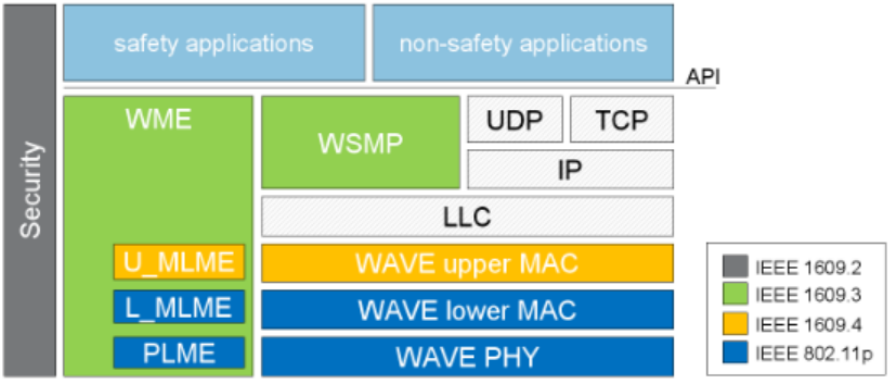

Figure 2.9. WAVE Protocol Stack [12]

WAVE Physical Layer

As shown in Figure 2.9, the IEEE802.11p standard is used to represent the WAVE PHY layer. The IEEE802.11p PHY layer is similar to the IEEE 802.11a PHY layer as shown in Table 2.2, where both the version operates at the same frequency range of 5 GHz, specifically 802.11p operates between 5.850 and 5.925 GHz. This is reasonable because changes in the MAC layer are mostly software related but major changes in the Physical layer could demand entirely new wireless air-link technology. The modulation technique adopted by 802.11p as well as 802.11a is Orthogonal Frequency-Division Multiplexing (OFDM). An OFDM signal consists of several closely spaced modulated carriers. Usually, when signals are transmitted close to one another they should be spaced out so that the receiver would be able to separate the signals and there should be a guard band between them to prevent the interference. However, subcarriers in OFDM are orthogonal to each other which prevents the cross-talk between subchannels and the need for inter-carrier guard bands. This reduces complexities in the design of the transmitter and receiver.

	IEEE 802.11a	IEEE 802.11p
Data Rate (Mbps)	6, 9, 12, 18, 24,36, 48, 54	3, 4.5, 6, 9, 12, 18, 24, 27
Modulation	BPSK, QPSK, 16-QAM, 64-QAM	BPSK, QPSK, 16-QAM, 64-QAM
ODFM Symbol Duration	4.0 µs	8.0 µs
Guard Period	0.8 µs	1.6 µs
Occupied Bandwidth	20 MHz	10 MHz
Frequency	5 GHz ISM band	5.850-5.925 GHz (dedicated)

Table 2.2. Comparison between IEEE 802.11a and IEEE 802.11p [13]

IEEE 802.11p uses 10 MHz wide channel while IEEE 802.11a uses a 20MHz wide channel. The main reason for this change is to address increased RMS delay spread in the vehicular environment. This change also prevents the inter-symbol interferences by having longer guard intervals compared to IEEE802.11a. The 75MHz band in IEEE 802.11p is divided into 7 channels with 10 MHz width, One Control Channel (CCH) and 6 Service changes (SCH). CCH is located at the centre whereas SCH are at the sides. CCH used solely for security communications. SCH can be used to transmit all information with lower priority. It is possible to combine two SCHs into one 20 MHz-wide SCH thus doubling the data rate.

WAVE Lower MAC

Similar to WAVE PHY layer, the IEEE 802.11p MAC layer represents WAVE lower MAC which is again a modified version of previous standards to support vehicular networks. WAVE Lower MAC includes some of the features from IEEE 802.11e such as channel coordination and Enhanced Distributed Channel Access (EDCA). EDCA is a modified version of Distributed coordination function (DCF) which operates on the contention period. EDCA adopts contention-based channel access and uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). CSMA/CA tries to reduce the frequency of packet collisions occurred due to simultaneous transmissions. It follows the idea of the "Listen Before Talk" (LBT) principle. After receiving the packet from upper layers, a node transmits only if the medium is free for an amount of time equal to an AIFS (Arbitration Inter-Frame Space). If the medium is not free, the node waits for a random back-off time before transmitting. The main feature of EDCA is traffic differentiation using multiple access categories. Each access category has a different access priority, back-off time interval. IEEE 802.11p MAC categorizes the messages based on

its priority and allocates proper access parameters for safety-related messages to ensure their quick and successful transmission. Vehicular safety-related applications demand instantaneous data exchange capabilities and do not have enough time to perform standard authentication and association to join a BSS. The WAVE standard introduces a new BSS type: WBSS (WAVE BSS). A station forms a WBSS by first transmitting an on-demand beacon. The demand beacon contains the information like services offered by WBSS and necessary configuration information to become a member of WBSS. Based on the information provided by the beacon, a station can decide to join a WBSS with no further interactions. This reduces the necessity of association and authentication processes by offering extremely low overhead for the WBSS setup.

WAVE Upper MAC

The WAVE upper MAC layer defined by the IEEE 1609.4 standard which is responsible for the multi-channel coordination. WAVE has two different channel types: Control Channel (CCH) and Service Channel (SCH). IEEE 1609.4 specifies how to use them using Channel Coordination. Channel Coordination is designed to support data exchanges involving one or more switching devices with alternating operation on CCH and SCH channels. A single radio device can use different channels by alternating between SCH and CCH with 50ms dedicated to each of them. Time is divided into sync intervals. In 1 sec, there are 10 sync intervals with each sync interval lasting for 100ms. Each sync interval composed of A CCH interval and A SCH interval. A guard interval is present at the beginning of each channel interval (CCH interval or SCH interval) which is used to account for radio switching and timing inaccuracies among different devices. The synchronization function uses UTC (Coordinated Universal Time) as the common time base for CCH and SCH intervals which can be obtained by Global Positioning System (GPS) receivers. GPS receivers typically provide a precise one pulse per second (PPS) UTC signal with an error of less than 100 ns, and these precise one PPS signals can be used for timing and synchronization. Timing Advertisement frame can be used by the nodes without local timing source. Those devices use the timestamp field in the Timing Advertisement frame as an input to estimate the UTC. Nodes that are not synchronized can only monitor the CCH channel for safety messages. The Control Channel (CCH) is used to transmit management packets and WAVE Short Messages (WSM) carried by WAVE Short Message Protocol (WSMP) whereas Service Channel (SCH) is used to transmit all packet types, including IPv6 packets. The type of access given to CCH and SCH channels are categorized into three types: continuous channel access (CCH or SCH), alternating access between two channels (CCH/SCH or two SSHs),and immediate channel

access.

WAVE Short Message Protocol (WSMP)

DSRC protocol stack is created to support both standard and vehicular communications. For non-safety IP based messages, default protocols such as the IP (Internet Protocol) and the TCP (Transport Control Protocol) or UDP (User Datagram Protocol) can be exploited by the network and transport layer respectively. Due to their overhead, these protocols are not suitable for V2X communication scenarios. WAVE Short Message Protocol (WSMP) is a networking protocol specifically designed for V2X communications. WSMP also allows the applications to choose the desired physical characteristics that should be used in transmitting the messages. The PHY and MAC layer read the contents of each packet and adjust the radio power, data rate accordingly before their transmission. The WAVE Management Entity (WME) is responsible for the management of networking services function, provided by the IEEE 1609.3 standard. This entity takes care of frame queuing, priority channels and handling of safety messages. Server applications registers with WME using a Provider Service Identifier (PSID) and user applications register their interests using WME. Based on the PSID, WSMs are delivered to the respective application(s). If the application-service is not interested in the PSID value of a received WSM, the receiver ignores the message. WAVE Security Entity (WSE) is responsible for the management of data encryption mechanisms and key management. WSM follows IEEE 1609.2 standard, that defines the format for secure messages. Since WSMs play a major role in Vehicular safety-related applications, protecting them from cyber-attacks is very crucial.

2.1.3 V2X Messages

C-ITS aims at creating a co-operative vehicular environment where vehicles take actions based on the information gathered from surrounding traffic actors and infrastructures. C-ITS applications are categorized into three categories: safety, traffic efficiency, infotainment. The communication strategy of most common C-ITS applications can be grouped into two types: Periodic status exchange and Asynchronous notification [14]. Periodic status exchange handles messages that are sent periodically to announce a vehicle's location, speed, the status of RSU's. It can be used by traffic efficiency-related applications for monitoring vehicle movements. It can also be a part of safety applications where vehicle information can be used to predict potential vehicle collisions in the future. Asynchronous notification is generated to inform the occurrence of specific events. It's mostly used by safety applications to broadcast information about road accidents,

slippery roads and so on. To standardise the application support messages, ETSI has defined two basic messaging services: Cooperative Awareness Basic Service for Cooperative Awareness Messages (CAM), and the Decentralized Environmental Notification Basic Service for Decentralized Environmental Notification Message (DENM).

Cooperative Awareness Messages (CAM)

The Cooperative Awareness Message (CAM), defined in the ETSI EN 302 637-2, can be considered as the heartbeat messages of the ITS scenarios where nodes periodically send their information to the neighbouring nodes. Some of the information included in CAM's are heading, acceleration, intended route, timestamp. By gathering information from CAM's, the vehicle can track the movements of surrounding vehicles and look for any potential collision occurrence in the future. If vehicles are in a collision course, cooperative manoeuvres are enabled to avoid the collision.

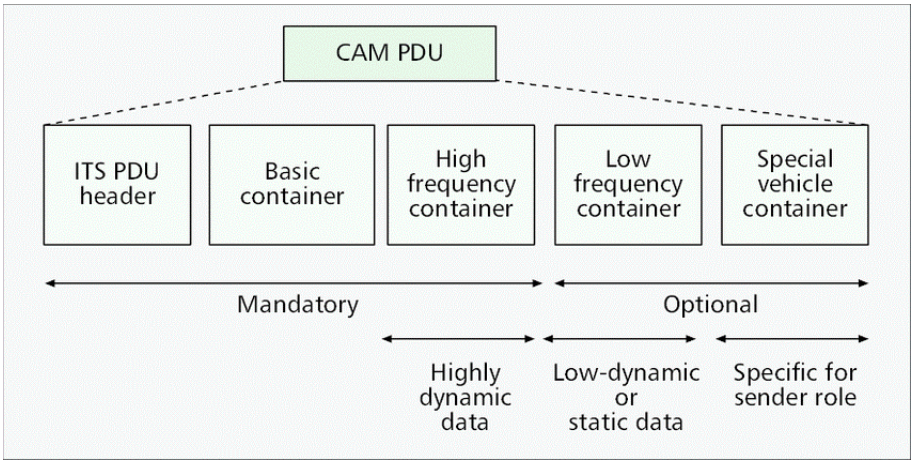

Figure 2.10. CAM Message Structure

The figure 2.10 represents the CAM message structure. The header contains details of the message such as version, message identifier and generation time. The containers in the mandatory section handle the information like sender ID, type of ITS station, position, heading. Some other optional parameters can be included by the ITS station based on the recommendations made by the standard. The frequency of the CAM message can be varied from 10Hz to 1Hz. The dynamic generation of CAM is also possible by considering the change of position and speed of the vehicle.

CAMs are generated by the CAM Management and messages are passed to lower layers when any of the following conditions satisfied:

- the maximum time interval between CAM generations: 1 second;
- the absolute difference between current heading (towards North) and last CAM heading > 4°;
- distance between the current position and last CAM position > 5 m;
- the absolute difference between current speed and last CAM speed > 1 m/s; These generation rules are checked every 100 ms.

Distributed Environmental Notification Message (DENM)

DENM service generates a DENM based on the occurrence of certain events defined by the application. It aims to alert other vehicles about the event which has an impact on road safety. According to standard, an event contains the following details: event type, position, detection time, the destination area, transmission frequency. The application sends the event details to the DENM service which starts transmitting DENM messages periodically over the specified destination area. It also notifies the DENM service about any changes in the events. Once the time is expired or the application cancels the event, the DENM service stops transmitting the DENM messages. The figure 2.11 represents the DENM message structure.

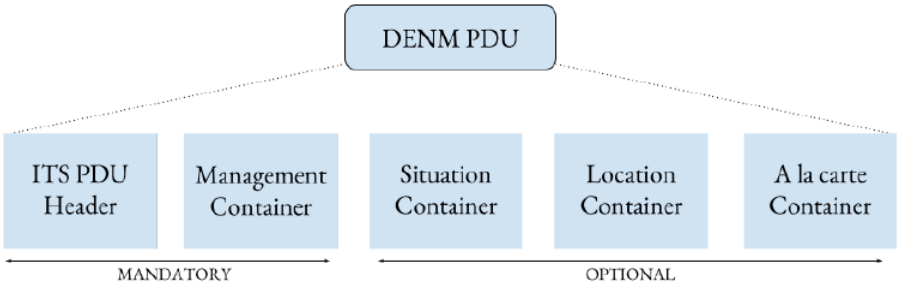

Figure 2.11. DENM Message Structure

2.2 Vehicle Dynamics

In this section, we are going to talk about the basics of Vehicle Dynamics. In a simple way, a vehicle is broadly classified into three modules: power, chassis, and body. The power module is to take care of the engine, gearbox, axles, etc. The chassis module includes multiple subsystems such as suspension, steering, tires, and so on. Vehicle dynamics focuses on representing all these modules, subsystems, interactions with the external forces as a mathematical model. The main aim of vehicle dynamics is to study the safety, and comfort of the vehicle occupants by understanding the behaviour of the vehicle under different scenarios through the mathematical models. The vehicle dynamics play an important role in the development of a vehicle.

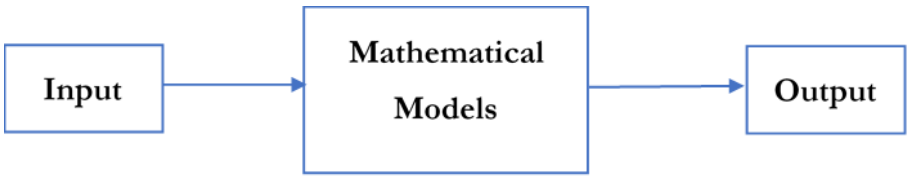

Figure 2.12. Basic representation of Vehicle Dynamics

In a simple mathematical model as shown in Figure 2.12, the definition of the vehicle is not anymore in the form of subsystems instead of vehicles that are defined in terms of mass, moment of inertia, stiffness, damping, compliance, etc. Vehicle-road coupling is an important aspect of vehicle dynamics. The input can be a wide variety of actions performed by a driver such as steering input, braking, accelerating through respective subsystems whereas outputs represent the quantities related to safety, comfort such as braking distance, which is the main aspect in safety, acceleration/deceleration that deals with the comfort of the passenger. A mathematical model should be complex enough to consider driving dynamics such as manoeuvres, self-steer behaviour, oscillatory motion behaviour, load shifts, and so on. The complexity of a mathematical model increases with the Degree of Freedom (DOF). Vehicle Dynamics is broadly classified into three categories:

- Longitudinal Dynamics: forces in the longitudinal direction. e.g. acceleration;
- Lateral Dynamics: forces in the lateral direction. e.g. cornering, handling, stability;
- Vertical Dynamics: forces in the vertical direction. e.g. vibration, road/tire contact.

For this thesis, we are only going to see about longitudinal dynamics and control.

Longitudinal Dynamics

Longitudinal Dynamics study about the forces and motions in longitudinal direction. It can be used to predict top speed, acceleration and braking performances, gradeability, fuel consumption. Figure 2.13 shows types of forces that act on a vehicle in an inclined road.

The front and rear longitudinal tire forces are generated from the vehicle power train. Based on Newton's second law, tire forces should overcome the resistance forces such as aerodynamic force (F_{aero}), gravitational force ($mgsin\alpha$) and rolling resistances (R_{xf},R_{xr}). The imbalance between them decides the resultant longitudinal acceleration of the vehicle.

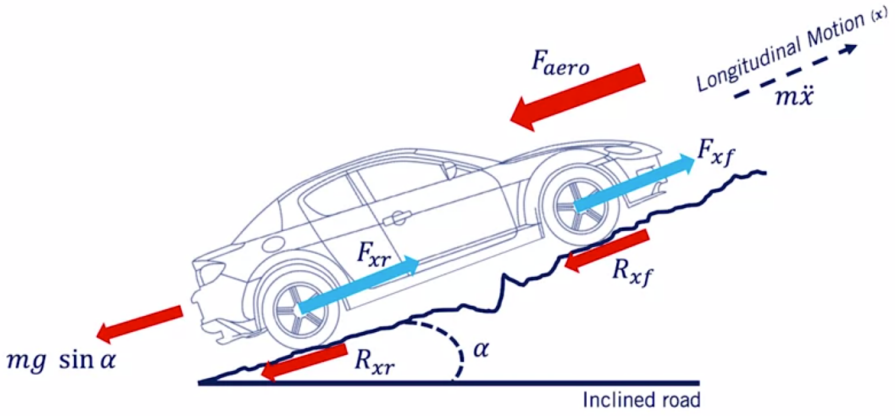

Figure 2.13. Longitudinal Vehicle Model

A longitudinal model can be represented by:

$$m\ddot{x} = F_{xf} + F_{xr} - F_{aero} - R_{xf} - R_{xr} - mgsin\alpha \tag{2.1}$$

where:

- $m\ddot{x}$ is the vehicle acceleration,
- F_{xf}, F_{xr} are front and rear forces respectively,
- F_{aero} represents the aerodynamic forces,
- R_{xf}, R_{xr} are front and rear rolling resistance,
- $mgsin\alpha$ represent the gravitational force on inclined surface.

We can simplify the model by combining tire forces, rolling resistance forces, road inclination angle approximation to achieve a basic dynamic model for the longitudinal motion.

$$m\ddot{x} = F_x - F_L \tag{2.2}$$

$$F_L = F_{aero} - R_x - mg\alpha \tag{2.3}$$

where:

F_x represent traction force, F_L represents the total resistance forces acting on the vehicle

We can develop different mathematical models to represent each of the forces mentioned in the equation 2.2 based on our requirements.

2.3 Control Strategies

The automotive industry is transitioning from hardware- to software-defined vehicles, the relevance of software for core technology trends is increasing rapidly [15]. To provide safe and comfortable driving, cars are now equipped with multiple devices such as RADAR, camera, ultrasonic sensor, lidar, positioning systems to understand the surroundings of a vehicle. Sensors play a major role to fulfil the future requirements of partially and fully autonomous vehicles. For example, RADAR has introduced multiple safety applications in the automotive industry. RADARs have the capability to detect objects in a wide area based on their operation range. Apart from these sensors, we can also acquire vehicle data from the Inertial measurement unit (IMU), and internal buses (Controller Area Network (CAN)). With the abundant availability of data giving information about the vehicle surroundings as well as the current vehicle behaviour, multiple safety applications have been created to process and analyse the data efficiently. Those applications are collectively called as Advanced Driver Assistance Systems (ADAS). ADAS plays a preventive role in mitigating crashes and accidents by providing a warning or additional assistance in steering/controlling the vehicle. ADAS is considered as an evolution of Driver Assistance Systems (DAS). DAS uses data from sensors measuring vehicle internal values such as velocity, acceleration or wheel rotational velocity. A few notable DAS based applications are Anti-lock Braking System (ABS), the Electronic Stability Control (ESC) and the Traction Control System (TCS). Some of the ADAS based applications are Adaptive Cruise Control (ACC), Autonomous Emergency Braking (AEB), Electronic Stability Control (ESC), Lane Keeping Assist (LKA). However,

ADAS related applications can only react to the objects that are in the range of the sensors.

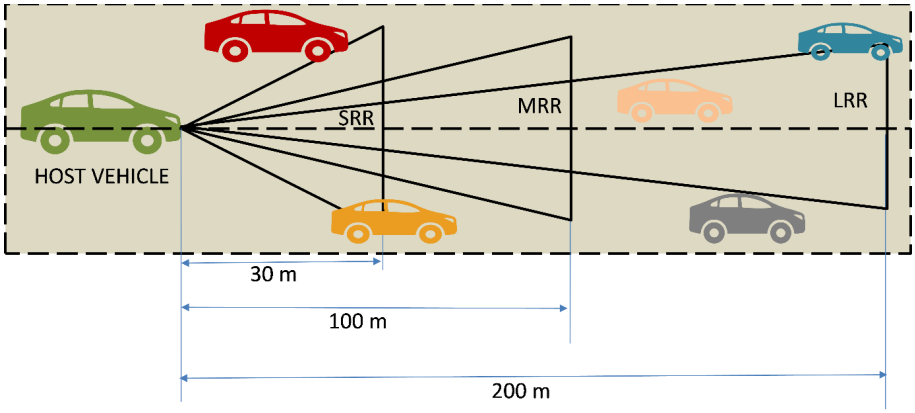

Figure 2.14. RADAR Types: LDR- Long Range RADAR; MDR-Mid Range RADAR; SDR-Short Range RADAR [16]

For example, RADAR cannot detect the slow-moving vehicle that presents before the target vehicle and its range depends on the type of RADAR used as shown in Figure 2.14. This kind of scenario can potentially create a collision between vehicles. With the advancement in V2X technologies, we can overcome these situations. Collision Avoidance, Cooperative Adaptive Cruise Control (CACC) are some of the applications that consider data from on-board sensors as well as data received through communication frameworks. According to, human errors are the critical reason for 94% of car crashes[17]. Among them, 41% due to recognition error which includes driver's inattention, internal and external distractions, and 33% are caused by decision errors such as driving too fast for conditions, illegal manoeuvre, misjudgement of the gap [17]. Several studies showed that ACC and CACC help to improve traffic efficiency by reducing the number of accidents, increasing vehicle average velocity. According to the study, CACC also increases the string stability in platooning, thus improving the traffic flow.

2.3.1 Adaptive Cruise Control

Adaptive Cruise Control (ACC) is an enhancement of conventional Cruise Control. The conventional Cruise Control can only maintain the user set speed by accelerating/decelerating the vehicle. The Driver has to intervene if the preceding vehicle velocity is lower than the user set speed. ACC is developed to overcome this drawback by adapting the vehicle's speed to the traffic environment. The detection of the preceding

vehicle is usually handled by RADAR based system. The RADAR gives two data points as input to ACC: ds represents the relative distance between the own vehicle and target vehicle; dv represents relative velocity between own vehicle and the target vehicle. If a slower moving vehicle is detected, the ACC system decelerates the vehicle to maintain the time gap between the target vehicle and the host vehicle. If there is no vehicle present in the range of the RADAR, the vehicle accelerates to its set cruise speed. In this way, ACC can control the vehicle without the intervention from the driver. However, the driver can de-activate ACC at any time by giving any kind of input to the system. Figure 2.15 shows simplified ACC working model.

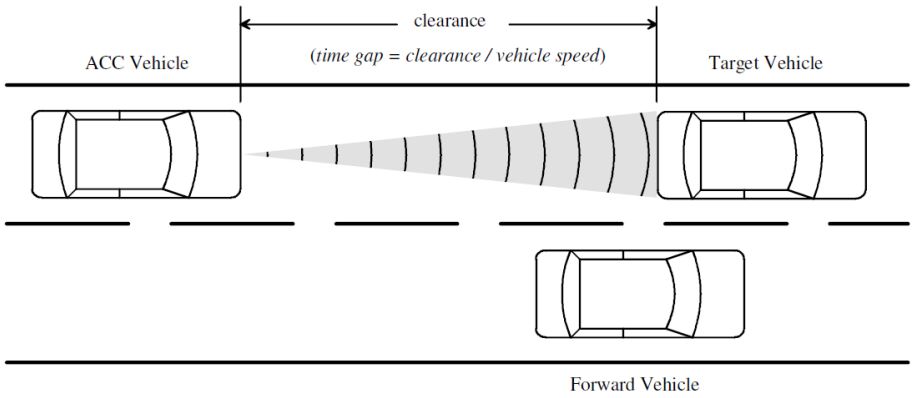

Figure 2.15. ACC Working Model

2.3.2 Automatic Emergency Braking

The purpose of Autonomous Emergency Braking (AEB) is to avoid or reduce the impact of a collision which generally caused by decision errors made by the driver. AEB requires two inputs: the surrounding vehicle's information and its vehicle state. The surrounding vehicle's information is gathered by RADAR and/or Camera.The AEB working principle can be grouped into four steps [18]:

1. **Identify critical situations:** AEB finds a potential collision situation by combining the information from exteroceptive sensors and own vehicle states.
2. **Prepare the braking system and warn the driver:** After the identification of a potential collision situation, AEB starts filling the brake circuit with fluid. This helps the system to reduce the preparation time to apply the brake, in turn, reducing the braking distance. It also warns the driver about the possible collision through visual and auditory warning-signals.

3. **Soft braking:** if the driver didn't respond to the warnings and potential collision event is still valid, then AEB starts applying the brake with a deceleration request up to -4 m/s2.

4. **Hard autonomous braking:** if the collision is imminent and the driver is not responding to any warning provided earlier, AEB gets activated and it takes control of the brake system and apply the emergency brake with deceleration forces up to -9.8 m/s2 . AEB tries to avoid the collision or at least reduce the impact of a collision by applying the emergency brake.

2.3.3 Collision Avoidance

Collision detection is the first and foremost step to avoid a collision. In this version of the Collision Avoidance algorithm, an Infrastructure gathers information of all the vehicles around it through V2I communication systems. A generic trajectory-based algorithm is used to detect any vehicle that is on the collision course with another. The algorithm requires velocity, heading angle, position of the vehicles to detect a potential collision occurrence. Since it's a trajectory-based algorithm, it calculates the future position of each vehicle using these inputs. It just assumes the future possible position with the current velocity and heading angle. As of now, the algorithm has calculated the position of all vehicles for the next 5 seconds. At this point, it calculates the distance between each vehicle position with other vehicles. If the distance between any vehicle is less than the predefined threshold, those vehicles are considered to be in a collision course with each other. To avoid this collision, a corrective action like change in acceleration is communicated to one of the vehicles that are identified to be in the collision course with another vehicle. The choice of vehicle could be based on multiple parameters like vehicle priority, distance, comfort

2.3.4 Co-operative Adaptive Cruise Control

With the technological advancements in V2X communications like DSRC, a host vehicle has the opportunity to communicate with other vehicles and/or infrastructures to get information about the surrounding environment. ACC can be co-operative by using information received through communication systems and react to the objects that are beyond the horizon of driver and sensors perception. Co-operative Adaptive Cruise Control (CACC) exploits sensor data, host vehicle data, over the air data from surrounding vehicles and/or infrastructures to control the vehicle acceleration, as shown in Figure 2.16.

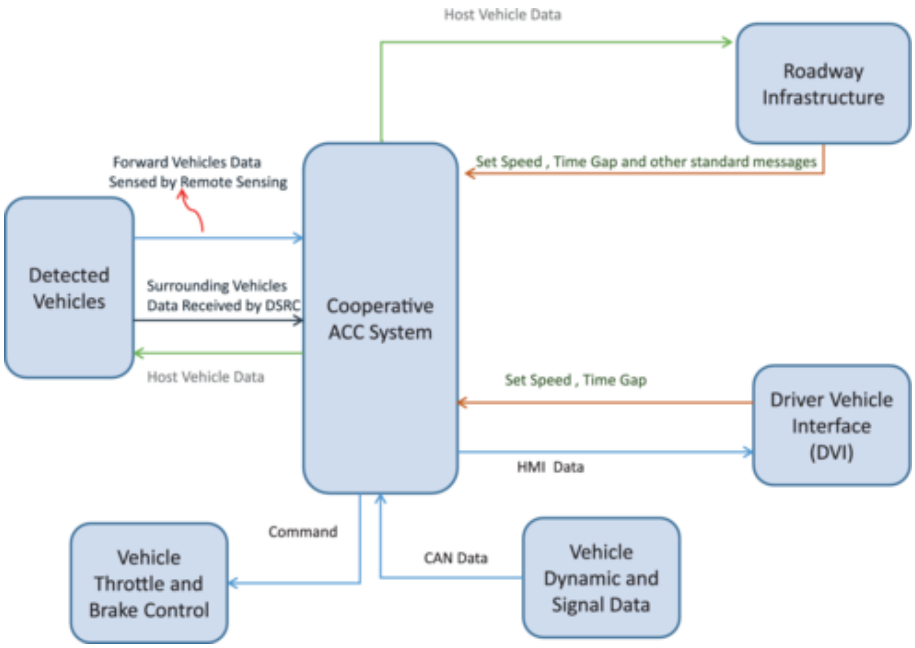

Figure 2.16. CACC Functional Elements [19]

Through V2X communications, CACC can get frequent updates about the surrounding vehicle behaviour which help to maintain the desired time gap between vehicles without oscillations and respond quickly to any changes in the preceding vehicle behaviour. Due to the fast response, we can also reduce the desired time gap and follow the vehicle closely without compromising driver safety. One of the main applications of CACC is Platooning. A platoon is a group of vehicles that travel together. The lead vehicle changes its speed and direction where other vehicles follow their preceding vehicle.

2.4 Literature Review

The goal of co-simulation frameworks is to combine the strength of multiple simulation tools and to develop a solution for real-world problems. There are few integrated simulators developed to achieve the interaction between traffic and network simulation environment; traffic flow and vehicle dynamics; Control, Communication, and Traffic flow. In this section, we are going to discuss the existing Co-Simulation frameworks and their functionalities.

The work made by Dmitry Kachan [20] integrates ns3 with MATLAB/Simulink to handle dynamic vehicle motion and wireless communication. MATLAB is used to create dynamic traffic movements where ns3 is used to simulate wireless communication

networks. Socket programming is leveraged to combine MATLAB/Simulink and ns3. At the end of each simulation step, Simulink does the calculations of future possible events and MATLAB sends those instructions to ns3 through a socket connection. And ns3 executes those instructions until the expiration of the simulation step. At the end of the ns3 simulation step, it sends the report to MATLAB. On each simulation step both simulators exchange messages which include status information for each node. The figure 2.17 shows the Interaction between ns3 and MATLAB. This work mainly focusses on creating an interface between two simulators but didn't define any control strategies and didn't test the framework by using different vehicle movements or communication scenarios. But we have created a framework that combines multiple simulators that includes mobility modelling and also tested our framework with different traffic scenarios.

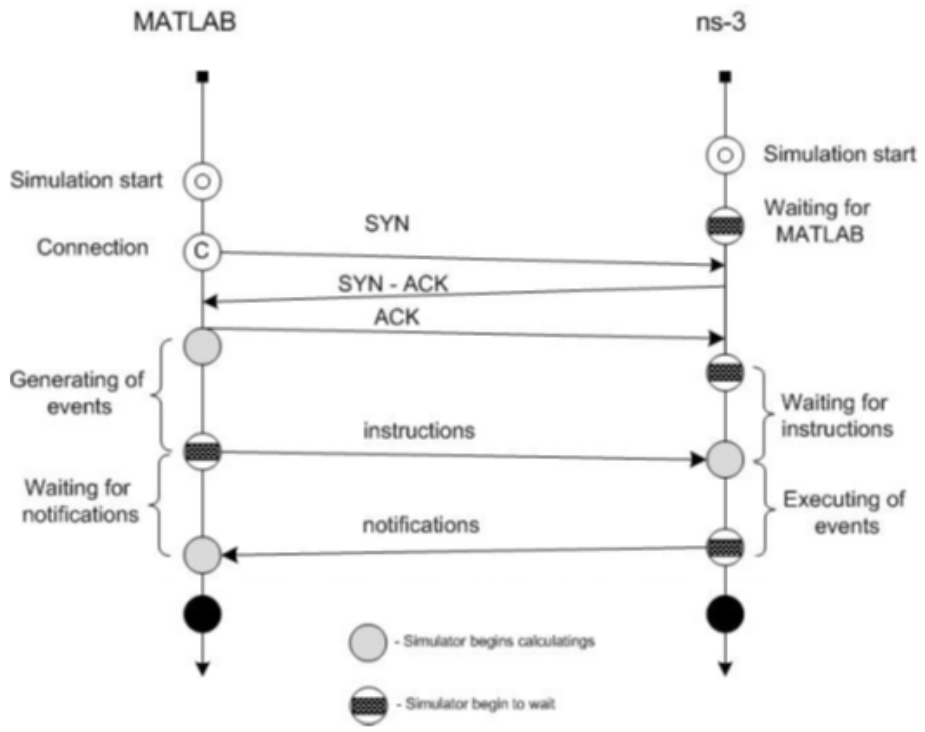

Figure 2.17. Interaction between ns3 and MATLAB [20]

The framework made by Jakob Kaths[21] et.all involves IPG CarMaker and Simulation of Urban MObility (SUMO) for integrating microscopic traffic simulation and vehicle dynamics simulation. The integrated framework is developed with MATLAB-Simulink as a mediator. CarMaker for Simulink helps CarMaker to communicate with Simulink whereas, for traffic flow simulation, the programming interface TraCI4Matlab

allows the interaction with SUMO through MATLAB/Simulink. In the simulation, SUMO is used to simulate all vehicles including Ego car from CarMaker. The position of Ego car is updated in SUMO through the TraCI command *"moveToVTD"*. During the simulation, the IDs, positions, and speeds of the surrounding vehicles are retrieved and sent to CarMaker to update them within the CarMaker environment. They have tested their framework by using high and low traffic volumes from SUMO and ACC implemented to adapt the speed of Ego Car in CarMaker using ADAS functionalities. Their main focus is to analyse the vehicle dynamics based on the traffic flow. While we have created a framework that focuses on vehicular safety applications by identifying and avoiding collisions through CACC as well as ACC. We have also used CarMaker for both traffic and ADAS functionalities.

The work done by Chenxi Lei[22] integrates SUMO, Simulink and OMNeT++ 2.18 for analysing String Stability. The term *"string stability"* means that any non-zero position, speed, and acceleration errors of an individual vehicle in a string do not amplify when they propagate upstream.

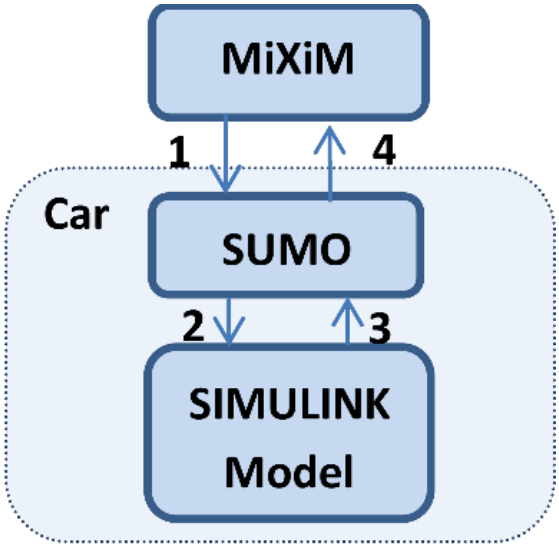

Figure 2.18. Simulation Experimental Structure [22]

Each simulator is devoted to one specific functionality. Simulink handles Vehicle behaviour including the ACC and CACC, SUMO takes care of Mobility, OMNeT++ with MiXiM simulates communication networking behaviour. To creating a coupling between SUMO and Simulink, Real-Time Workshop tool in Simulink is used to convert the Simulink model into a C++ shared library and called in the source code of SUMO. The bidirectional coupling between OMNeT++/MiXiM and SUMO is achieved through

TRAffic Control Interface (TraCI) by transmitting TCP messages. In their work, MiXiM collects information of all nodes(vehicles) at every simulation step and send it to SUMO through TraCI where acceleration data is given as input to the Simulink model for calculating the next acceleration and velocity. Based on that, vehicles are moved to SUMO. The new trace is sent back to MiXiM where it moves the communication nodes according to the vehicles' position information from SUMO. They have used the framework to analyse String Stability by varying time headway, beacon rate, packet loss ratio and ACC versus CACC. As we said earlier, his main focus is to study the string stability which impacts the platooning related applications. They also struggled to switch between ACC and CACC based controller while we have successfully implemented the same. We can switch between ACC and CACC controllers by considering the desired acceleration calculated by them.

Amr Ibrahim et. all [23] made a co-simulation framework consisting of MATLAB for control algorithms, ns3 for network simulation and SUMO for traffic behaviour. This framework runs in two different systems where MATLAB, SUMO runs on the Windows operating system and ns3 runs on a Linux virtual machine. MATLAB acts as an interface between ns3 and SUMO. The TCP connection is established between SUMO and MATLAB using TraCI (Traffic Control Interface). Socket programming is used for the TCP connection between MATLAB and ns3. The co-simulation framework is shown in figure 2.19.

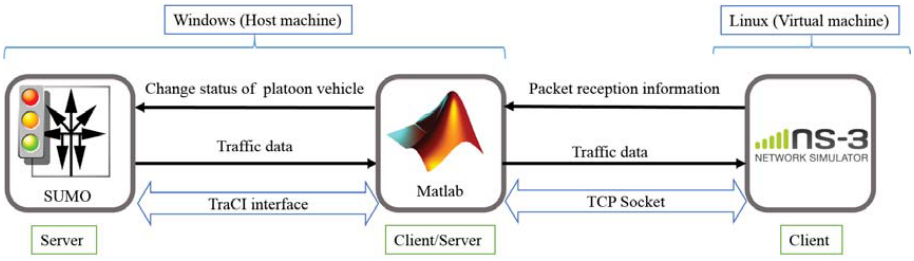

Figure 2.19. ns3 and SUMO Interaction through MATLAB [23]

During the simulation, MATLAB retrieves the vehicle's information such as the current position, speed from SUMO. The vehicle information is forwarded to ns3 through TCP socket connection where ns3 simulates the packet broadcast of each vehicle through a constant velocity mobility model. The packet reception information of each node is stored in the lookup table. MATLAB accesses the lookup table and runs the control algorithm to define the new position, velocity. This new vehicle information is updated in the SUMO through TraCI. This framework is used to validate different

platooning setups. Apart from the choice of simulators, their main focus is to study the impact of Packet Reception Ratio and delay on the overall platoon performance. They have executed their framework for different traffic densities in a realistic highway scenario. We have tested the framework for different scenarios in an ideal communication environment without considering channel load. We have also used LTE based communication model while they have employed WAVE standard to communicate between vehicles. To communicate between ns3 and MATLAB, we have exploited the MATLAB commands and Sockets while they have used only TCP/IP sockets to transfer information. Using MATLAB commands gave us flexibility to transfer vehicle information in multiple formats which is not possible with TCP Socket.

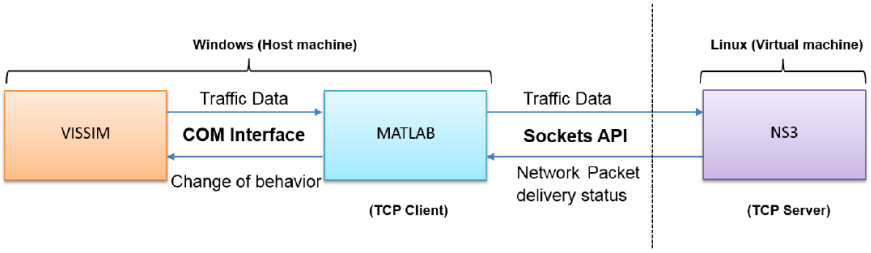

Figure 2.20. ns3 and VISSIM Interaction through MATLAB [24]

Apratim Choudhury et.all [24] developed a framework with VISSIM (for traffic simulations), MATLAB and ns3. VISSIM and MATLAB communicate through VISSIM's COM interface in a Windows host machine. ns3 is installed in a virtual Linux machine connected via virtual network. The sockets API is used for the communication between MATLAB and ns3. Figure 2.20 shows the framework developed by Apratim et. all. This simulation platform is used to model platoon control algorithms, the control objective of this algorithm is to compute the necessary acceleration that the car needs to follow the leader. The waypoint mobility model is used in ns3 for moving nodes. Therefore, to move from point to another point, a node requires an origin and a destination waypoint. To get these waypoints, VISSIM has to run the simulation beforehand and send the origin and destination waypoint to ns3. Thus, control action is implemented in VISSIM with an approximate delay of 0.1 seconds. They have used 802.11p standard to establish V2V communication model in order to exchange CAM messages between vehicles. We have used LTE standard to establish V2I communication to implement the CACC algorithm. Their focus is on the platooning related applications while we focused on the enhanced CACC system that predicts the possible collision in the future.

Based on the research made on Co-Simulation frameworks, we can see that most of them used Sumo as their traffic simulator and used TRACI to connect with other simulators, ns3 or OMNeT++ as network simulator, Simulink or CarMaker as vehicle dynamics simulator. Among them, Dmitry Kachan framework depends on the creation of new C++ methods to make ns3 communicate with an external simulator. i.e. MATLAB. We have to change some core functionalities of ns3 to achieve the connectivity between ns3 and MATLAB. Whereas framework introduced by Lei converts Simulink model into C++ code to create a communication between SUMO and OMNeT++. The introduction of new methods and the conversion of a base model has two disadvantages. One, we lose the flexibility of the framework. If we want to add new functionality to the framework like adding vehicle dynamics which doesn't exist in both of the frameworks, we have to either use simulators which are compatible to the framework or convert the existing simulation model into a model that is suitable for the framework. Second, while converting models, we lose some core functionality of the base simulator like while converting Simulink code Lei loses some of the functionalities that are only available in Simulink. While the framework proposed by Jakob studied only the behaviour of the vehicle in various traffic scenarios using vehicle ADAS functionality. The framework didn't consider the usage of connected vehicles in the traffic scenarios. With the automotive world moving toward connected vehicles, this seems to be a big drawback of this framework. While Apratim and Amr Ibrahim, focuses on creating a communication between vehicles but they didn't consider the dynamics of the vehicle. While introducing the communication between the vehicles, we have to consider also the response of the vehicle which helps the framework to create a virtual simulation close to real-world environment. In the proposed framework, we have addressed all the disadvantages expressed by the above frameworks. We have used CarMaker to study the dynamics of the vehicle, to emulate traffic scenarios and vehicle sensors, and ns3 for creating communication between the vehicles. Moreover, we have combined these simulators using a Python-based interface which doesn't require conversion any core simulators. In this way, the proposed framework, able to use full functionalities of all the simulators and also able to replicate a real-world environment in a virtual simulation. As a control strategy, we have used enhanced CACC which makes the vehicle follow the preceding car as well as to avoid a collision that's going to happen in the future using a trajectory-based algorithm.

2.5 Simulation Tools

The objective of this paper is to develop a simulation framework that represents all the following system components realistically: vehicle connectivity, vehicle

dynamics, vehicle on-board sensors, vehicle traffic, and road scenarios. To achieve the project objective, the framework has to integrate multiple simulators, each devoted to the representation of one specific component.

2.5.1 Network Simulator

Network simulators are used to implement and analyse the behaviour of networks before deploying in the real world. Network simulators can evaluate the performance of network frameworks under dynamic changes e.g., the traffic conditions, the communication channel conditions. In this framework, Long Term Evolution is used for receiving information about the surrounding vehicles. The operation of the LTE is modelled using a network simulator.

Among multiple such network simulators, ns3 and OMNeT++ were considered implementing our framework. Both OMNeT++ and ns3 can emulate the behaviour of the LTE framework. However, OMNeT++ uses SimuLTE to model the data plane of the LTE/LTE-A Radio Access Network and Evolved Packet Core whereas ns3 uses LENA implementation of LTE to do the same. We have developed our framework with ns3 as our network simulator. The main reason for selecting ns3 over OMNeT++ is due to the availability of ns3 python bindings.

ns3

ns3 [25] is an open-source discrete-event realistic network simulator, which is developed to provide an open, extensible network simulation platform, for networking research and education. ns3 provides models of how packet data networks work and perform and provide a simulation engine for users to conduct simulation experiments. ns3 helps us to perform studies that are more difficult or not possible to perform in real systems.In ns3, simulation scripts are written in C++, with support for extensions that allow simulation scripts to be written in Python. ns3 is a user-space program that runs on Unix- and Linux-based systems. ns3 has a modular implementation where functionalities of ns3 are grouped into few libraries such as core library supporting generic aspects of the simulator, simulator library defining simulation time objects, schedulers, and events and node library defines abstract base classes for fundamental base objects in the simulator, such as nodes, channels, and network devices. The modular implementation allows for smaller compilation units.

The primary simulation objects used in the framework are Node, NetDevice & Channel, Packet, and sockets.

Node: Node helps us to form the network. For example, in LTE networks nodes can be represented as eNodeB as well as user equipment(UE). Nodes are connected through channels.

NetDevice & Channel: A key node object is NetDevice, which represents a physical interface on a node, for example, an Ethernet interface. Channels are closely coupled with the attached NetDevices. NetDevice subclasses are matched to a particular corresponding channel type. That is, for example, a PointToPointNetDevice is attached to a PointToPointChannel. In this way, we can avoid incompatibility between Channel and NetDevice.

Packet: ns3 Packet objects contain a buffer of bytes: protocol headers and trailers. ns3 Packets are designed to match with the real packet on a real network protocol. Briefly, packets are the data sent across networks in the same way as real network protocol would do.

Sockets: The sockets API exported to ns3 attempts to mimic the standard BSD sockets API. ns3 handles data received through sockets by using receive callback. For example, when a node receives a packet through a socket, it invokes the receive callback to handle the received data. In the same way, other common socket APIs like send(), connect(), and bind() are used to invoke their respective callback functions.

ns3 uses WAF as its build system. It is the new generation of Python-based build systems which helps to process the source code.

ns3 Python Bindings

As we discussed earlier, ns3 python bindings [26] are the main reason to choose ns3 over OMNeT++. Python bindings allow the programmer to write complete simulation scripts in Python. Python bindings for ns3 are developed using a tool called PyBindGen. PyBindGen is a tool that is used to generate Python bindings for C or C++ APIs and their headers. This tool is based on gccxml and pygccxml that scans ns3 API's and generates their python bindings. In this way, we can use ns3 libraries and write complete simulation scripts in Python. However, ns3 libraries are only accessible insider WAF build system, therefore simulation scripts are forced to execute only through WAF system using *./waf –pyrun filename* as a command.

The major drawback of python bindings is the restriction of user-defined packet data. In C++, we can send user-defined data through packets. However, in python, we can only define the size of the packet where ns3 fills those packets with empty data. Since we have to send information of the vehicles across the network, without a user-defined data packet we can't move forward with ns3. Thanks to ns3 modular bindings, we can modify

specific ns3 modules based on our needs. We have made changes to network modules where functionalities of packets are residing. In the packet source and header file, We have added new functions so that network packets in python bindings can handle user-defined data instead of just empty data packets provided by ns3. After making necessary changes in a module, we can use *WAF –apiscan=network* to incorporate the changes into the ns3 network library. However, these changes are not global, so we have to repeat these steps to handle user-defined data's in a different workstation.

2.5.2 CarMaker

The simulation tools with a detailed vehicular model and vehicle dynamics are used in automotive research and development. The detailed vehicular model includes the axle kinematics, suspension, steering and a tire model representing the driving dynamics of the vehicle. IPG CarMaker is one of the tools which gives us a complete vehicle model to analyse the dynamics of the vehicle with different test scenarios. The CarMaker is a simulation tool that can be used for testing light-duty vehicles in a virtually realistic environment. However, Carmaker also can simulate multiple sensor models to emulate Advanced driver-assistance systems and able to implement real-world traffic scenarios through traffic and scenario manager. Therefore, on the whole, CarMaker is a test platform that allows recreating real-world test scenarios in a virtual environment, simulating every type of road and traffic, and performing realistic execution through the event and manoeuvre-based testing method.

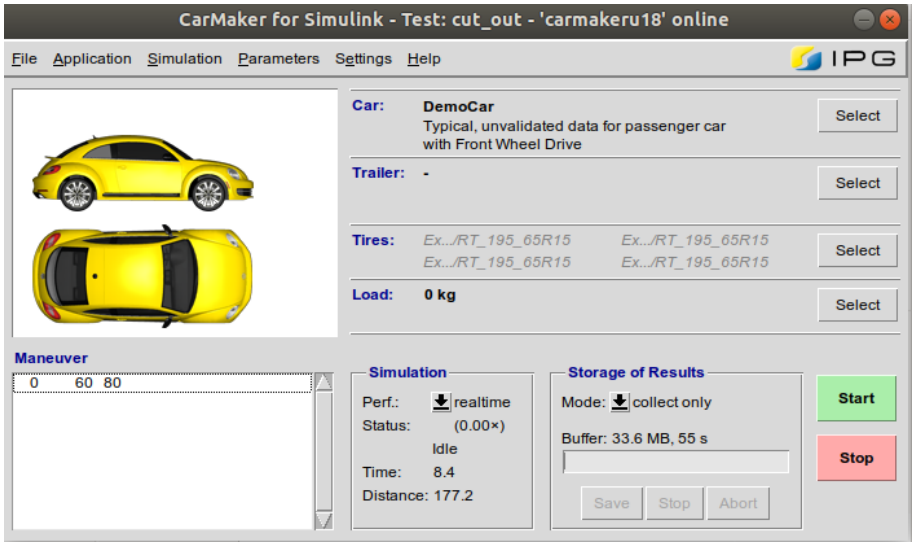

Figure 2.21. CarMaker GUI

To perform a simulation in CarMaker, type of vehicle, tires, driver, test track and a manoeuvre has to be defined. CarMaker provides several predefined models for each of these requirements. Once a data set is selected, it is possible to launch a TestRun. TestRun represents a test scenario in which all parameters of the virtual environment (vehicle, driver, road, manoeuvre, etc.) are sufficiently defined. The CarMaker can make the decision based on vehicle surroundings with the help of multiple sensors like Object Sensor, Radar, Lidar, Camera. The main GUI of CarMaker is shown in Figure 2.21

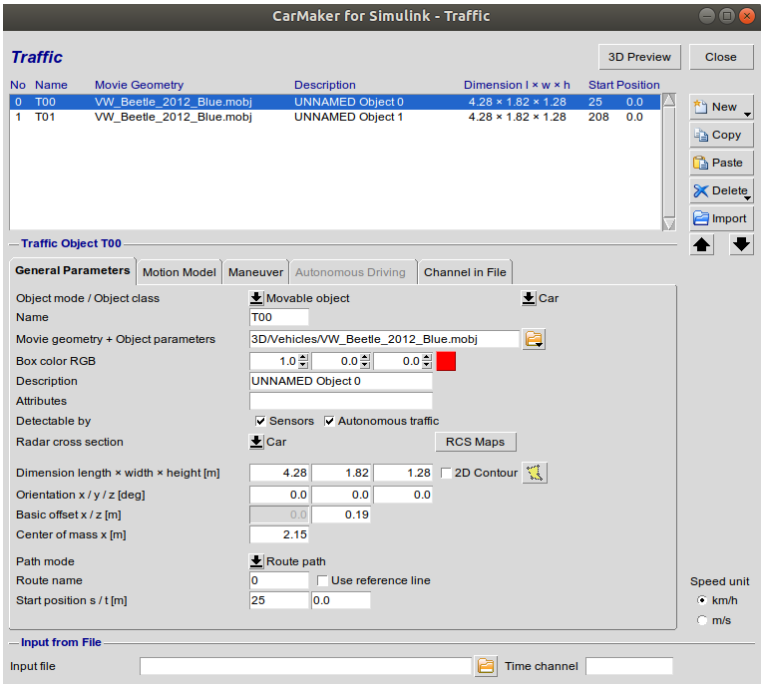

Figure 2.22. CarMaker Traffic Window

The next important section of CarMaker is the traffic section as shown in Figure 2.22. We can define static or moving traffic objects with a predefined model of car, truck, motorcycle or pedestrian (including bicycle and animals). We can also define their motion model parameters (starting position, velocity, route) and the manoeuvre.

The CarMaker offers the possibility to create a Road Network Using the Scenario Editor as shown in figure 2.23. Based on our requirement, we can exploit options available in the editor, which include the type of road (straight, turn, junction, slope, bump), the accessories (traffic light, sign or barrier) and the scenery (bridge, tunnel, environment objects).

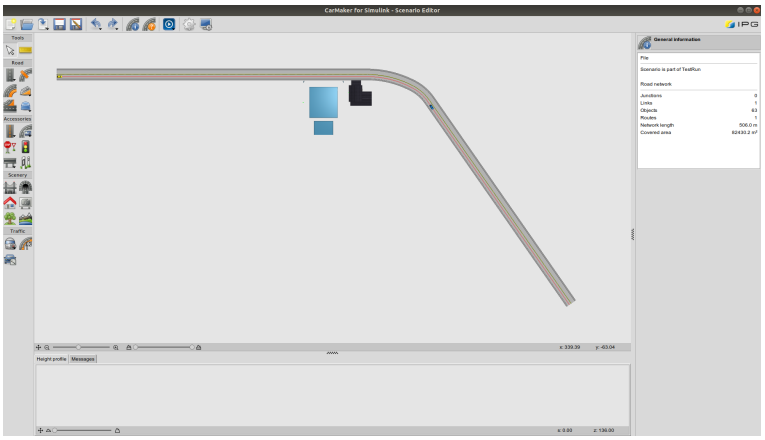

Figure 2.23. CarMaker Scenario Editor Window

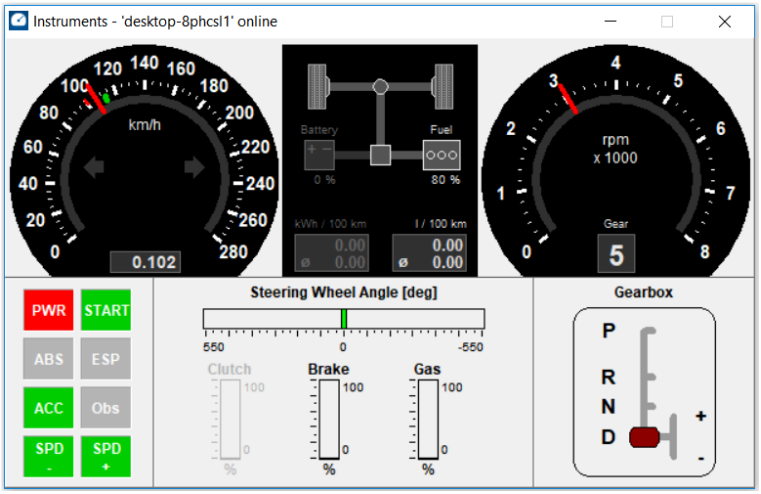

Figure 2.24. CarMaker IPGInstruments Window [27]

The CarMaker also comes with some other tools to better analyse the results when the simulation is running or concluded. IPGMovie enables the user to watch an animation of the current simulation. Through IPGMovie , we can visualise the results of the current test scenario which helps us to understand/improve the scenario better. IPG-Control in which it is possible to analyse variables behaviour through various plots. IP-GInstruments, which shows the actual dashboard of the car, with tachometer, rev counter, powertrain energy flow, fuel consumption, current gear number, driver's steering wheel movements, and pedal actions, as well as the presence of active ADAS systems. Figure 2.24 and 2.25 shows IPG Instruments and Movie window respectively.

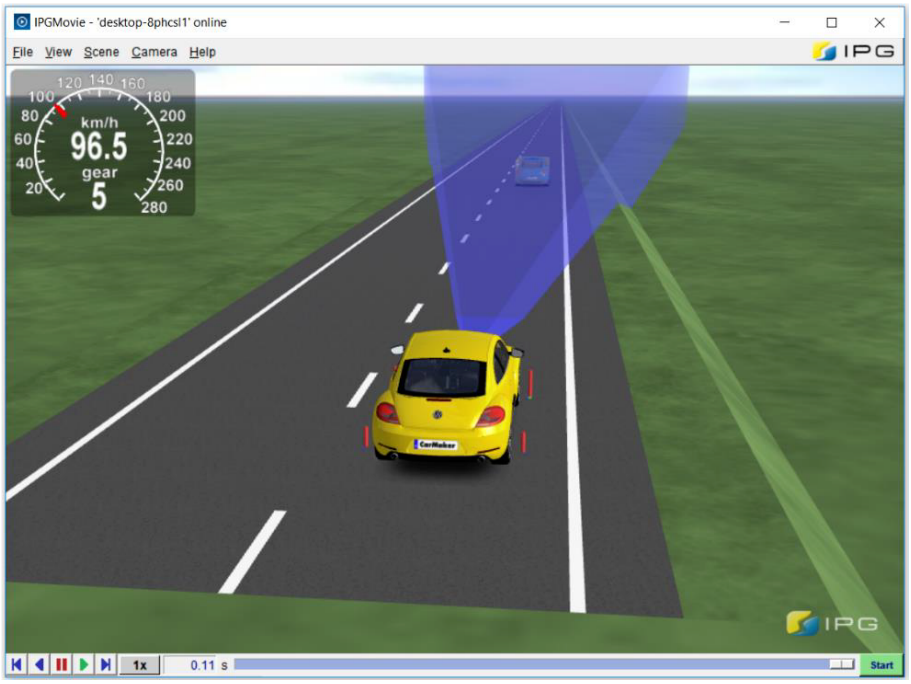

Figure 2.25. CarMaker IPGMovie Window [27]

CarMaker for Simulink

CarMaker for Simulink integrates CarMaker into the MathWorks' modelling and simulation environment MATLAB/Simulink. The features of CarMaker are added to the Simulink environment using an S-Function implementation and the API functions that are provided by MATLAB/Simulink. CarMaker for Simulink is a tightly coupled co-simulation framework resulting in a simulation environment that has both good performance and stability. This integration allows the use of the power and functionality of CarMaker in the intuitive and full-featured environment of Simulink. The CarMaker GUI can be used for simulation control and parameter adjustments, as well as defining manoeuvre and road configurations, IPGMovie can still be used to bring the vehicle model to life with realistic animation.

A new model can be built by extending the "generic" model that is available in CarMaker for the Simulink folder. However, we can also create a model from scratch based on our requirement with the help of CarMaker for Simulink blockset available in Simulink library. The blockset contains useful blocks that can directly connect a Simulink model with CarMaker. These blocks can extract information from CarMaker in real-time and use them in the Simulink environment to create user-defined models.The generic Simulink model of CarMaker and their blockset libraries are shown in figure 2.26

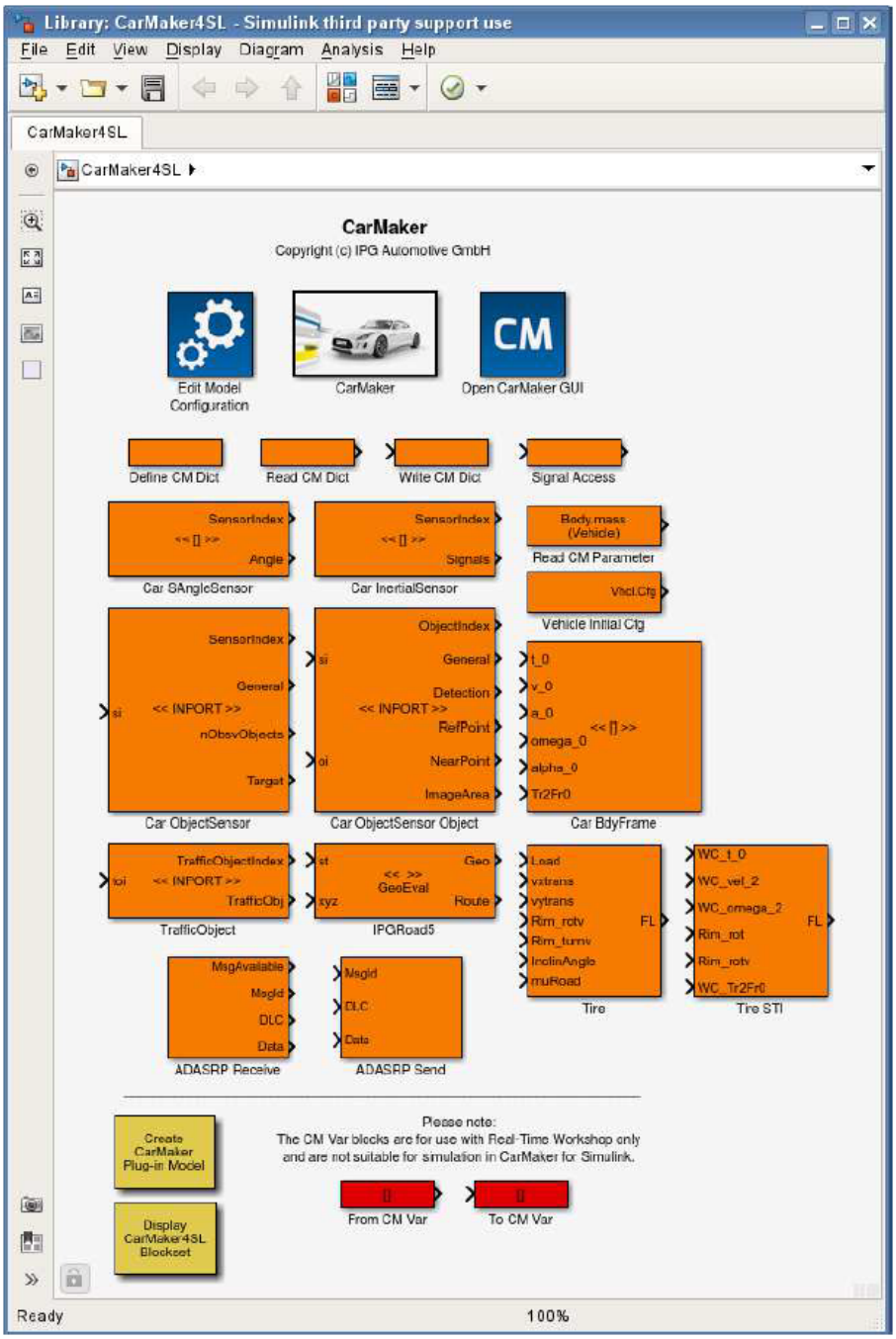

Figure 2.26. CarMaker for Simulink [29]

Chapter 3

Co-Simulation Framework Architecture

3.1 Architecture Feasibility Study

As we discussed in the previous section, the objective of this thesis is to create a co-simulation framework that represents vehicle dynamics and sensors, traffic, and communication. Through analytical and experimental methods, we have explored multiple architectures with various simulators to fulfil the purpose of this thesis work. We have explained the drawbacks of the architectures that were not able to communicate with other simulators to create a co-simulation environment.

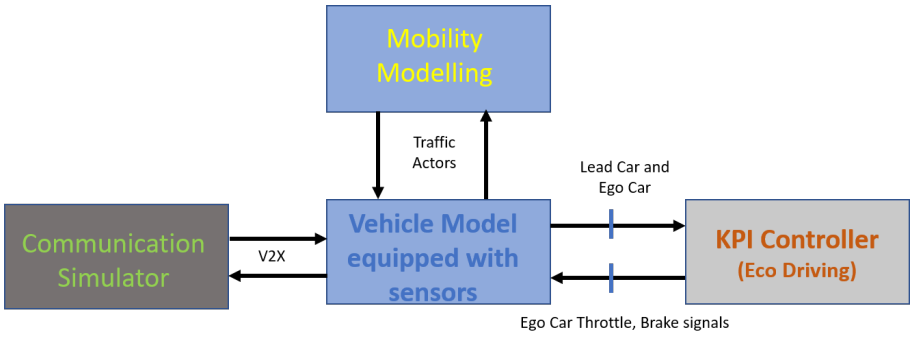

Figure 3.1. Basic Framework Architecture

Figure 3.1 shows the base architecture of the framework. Aimsun was used to simulate traffic vehicles (Lead Cars) and their mobility behaviours. We have used CarMaker to study the dynamics of vehicles and emulate the vehicle sensors whereas the LTE simulator focussed on providing communication functionality to each car involved

in the simulation. We have considered using SimuLTE (OMNeT++) and ns3 as LTE simulators. Simulink was used to implement the control strategies which take vehicle data as inputs and define the ego car actions while considering the surrounding traffic actors. The idea of this architecture is to get traffic vehicles (lead cars) from the Aimsun simulator and introduce them in the CarMaker environment or vice versa. By this method, traffic vehicles can see the ego car whereas ego car simulated in CarMaker able to see the presence of other cars through sensors. LTE simulator can help the cars involved in the simulator to communicate with each other and/or infrastructure through the LTE framework. With the help of this information, Simulink would decide the ego car's mode of action i.e. accelerating/decelerating the vehicle. The simplicity of the architecture also becomes the major drawback of this framework. Since each of these simulators are written in different programming language there is no common platform to combine them. Moreover, we have tried to establish communication between them through socket programming techniques. It didn't work as expected because each simulator uses different simulation step size some are event-driven simulators while others are time-driven simulators. Each of them runs their simulation in a different period without any synchronisation. To synchronise their communication between them, we needed a common platform that can act as a mediator between the simulators.

3.1.1 Feasibility Study 1

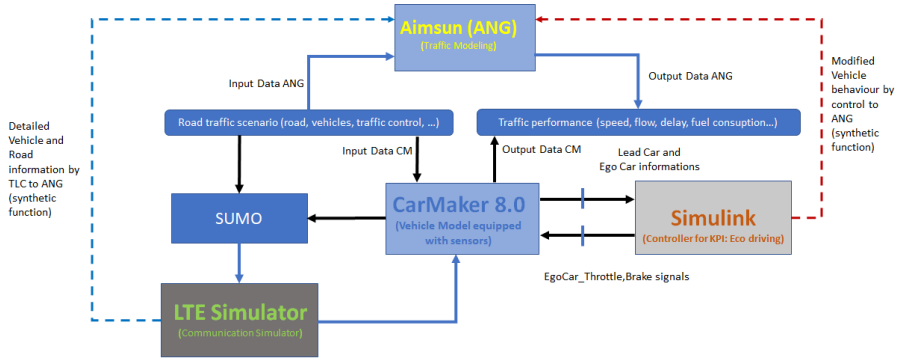

Figure 3.2. Framework Architecture with SUMO

We have included SUMO into the architecture as shown in figure 3.2, to act as a mediator between LTE and CarMaker through TraCI and APO library respectively. The idea of this architecture is to simulate the traffic vehicles in SUMO and AIMSUN through traces. With the help of TraCI, vehicles simulated in SUMO can use

the functionalities of the LTE simulator to establish vehicular communications. Through the CarMaker APO library, vehicles in SUMO can be re-created in the CarMaker environment. The main drawback of this framework is Aimsun was not able to communicate with SUMO in real-time, only traffic traces can be used in Aimsun. This behaviour ended up creating a one-way communication between simulators while we were unable to create feedback loop from Aimsun to update the vehicle status back n forth. This is one of the requirements to create a complete co-simulation framework where we should be able to simulate scenarios close to the real-world.

3.1.2 Feasibility Study 2

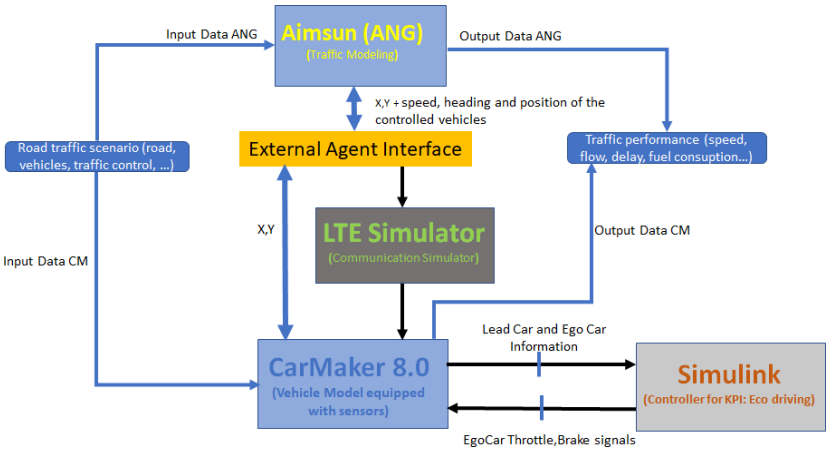

Figure 3.3. Framework Architecture with EAI

In Aimsun Next 8.4, they have introduced External Agent Interface (EAI) which can be used to exchange information with other simulators in real-time.The Framework Architecture with EAI is shown in figure 3.3. We tried to use this interface to overcome the drawback posed by previous architectural design. Even though, this EAI was working well on the Windows platform, we were unable to reproduce the same behaviour in the Ubuntu platform due to software issues. Apart from that, we were unable to create a link between the CarMaker and LTE simulator without SUMO. The main functionality of both Aimsun and SUMO is to simulate the mobility models. Using SUMO only to establish an interface between two simulators doesn't do justice for the simulator. So, we have started to explore other possibilities of creating a co-simulation framework. With the help of an extensive feasibility study, we were able to create a loosely coupled co-simulation framework that satisfies the thesis requirements. The system architecture and

their functionalities are explained in the following sections of this chapter.

3.2 System Architecture

The system architecture aims at creating a loosely coupled Co-Simulation framework that combines the strength of multiple simulators to study the impact of connected vehicles in safety applications. It is designed as a Python-based interface that acts as a middle-ware to enable the interoperability of multiple simulators.The finalised system architecture is shown in figure 3.4

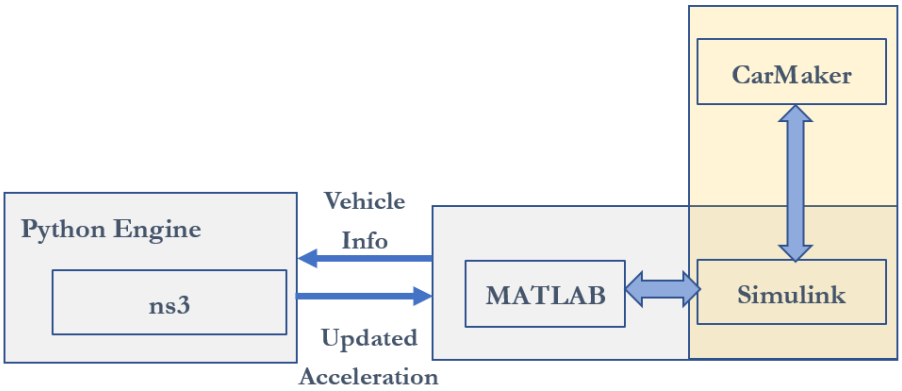

Figure 3.4. Co-Simulation Framework Architecture

The co-simulation framework composed of:

1. The ns3 module, which simulates the wireless communication network, including the network infrastructure and vehicles equipped with On-Board Units (OBU);

2. The Python engine, which takes as input the information provided by the other modules and implements the algorithms/logics for the decisions to be made at system/vehicle level;

3. The CarMaker module, which simulates the vehicle dynamics, the vehicle on-board sensors, and the vehicle traffic;

4. The MATLAB/Simulink module, which acts as an interface between the Python engine and CarMaker.

The Python engine has two main functions: (i) to connect Simulink (hence, CarMaker) with the ns3 simulator, and (ii) to host the control strategies. In more detail, CarMaker is used to model the Ego and Lead Car based on the predefined models. With the help of CarMaker built-in scenario builder and manoeuvre, we set our simulation in a predefined environment and monitor the movements of the Ego Car and Lead Cars. With the control strategy implemented in Python Engine, we must create an interface between CarMaker and Python Engine. However, there is no direct way to integrate CarMaker with Python, so we have used CarMaker for Simulink as an interface to extract the Vehicle Data from CarMaker environment. The data extracted from each vehicle is listed in table 3.1

Information	Usage in Control Algorithm
Vehicle Position Velocity Heading	For predicting vehicle position in next n seconds system
Link ID Lane ID Road ID	For positioning the vehicle in simulation road network
Steering Angle	To identify cut-in/cut-out scenarios
RADAR Object Detection RADAR relative distance RADAR relative speed	Available only for Ego car. To implement Adaptive Cruise Control

Table 3.1. Data extracted from each vehicle and their Importance

To make Simulink interact with the Python engine, we have exploited MATLAB commands along with MATLAB Engine API for Python. Once the vehicle data is accessible from Python Engine, we have used the ns3 network simulator to implement V2I communications to send the Vehicle data as CAM Messages to the Infrastructure Node. After receiving the data of all vehicles involved in the simulation, infrastructure node executes the control strategies and find the desired Ego Car acceleration. The desired acceleration is communicated to the ns3 Ego Car node. The Ego Car node sends the acceleration to the Simulink through TCP/IP connection while Simulink has been used to update it in the CarMaker environment. The two important points to be noted as a functionality of this simulation framework are: First, the exchange of vehicle information with infrastructure node and Ego Car acceleration is designed to happen every 100ms. We have set 100 ms as our simulation step, to support the fact that only 10 CAM messages are allowed in 1 second; Second, as per CarMaker functionality, we can

only update the acceleration of Ego car while parameters and manoeuvres of Lead Cars are fixed before starting of the simulation. Therefore, in this framework, we are only controlling the Ego car acceleration.

As a result, with the help of Python Engine as a middle-ware, we have leveraged all functionalities of CarMaker and ns3 to create a simulation framework that could analyse the impact of the connected vehicle in safety applications.

3.3 System Definition

3.3.1 CarMaker: Vehicle and Mobility Model

We have to define the vehicle model, traffic parameters and vehicle manoeuvres before running the simulation in CarMaker. The car used by the simulation framework as Ego Car and Lead Car is the "DemoCar", that is a Volkswagen Beetle. The car dynamic was maintained, except for the gearbox, which was changed from manual to automatic. Among multiple sensors in CarMaker, we have used RADAR to detect the surrounding objects which helps Ego Car to implement the ADAS functionalities. In CarMaker, we have added "RadarL" as an object sensor in front of the car with a coverage distance of 150m as shown in figure 3.6. The observation area of this RADAR model is showing in the figure 3.5.

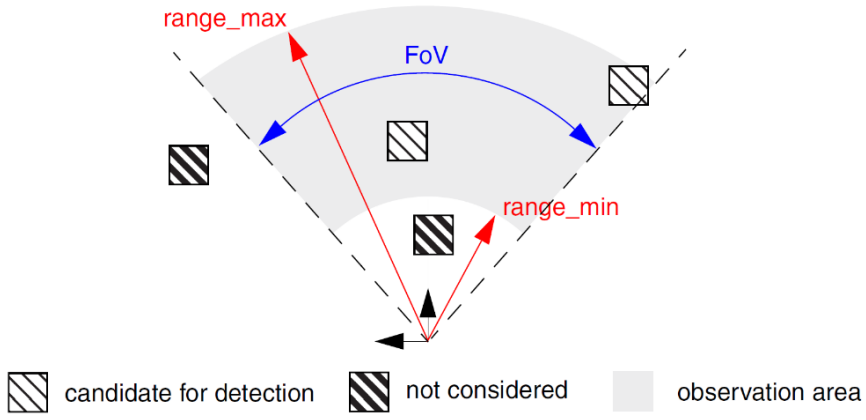

Figure 3.5. RADAR Observation Area [30]

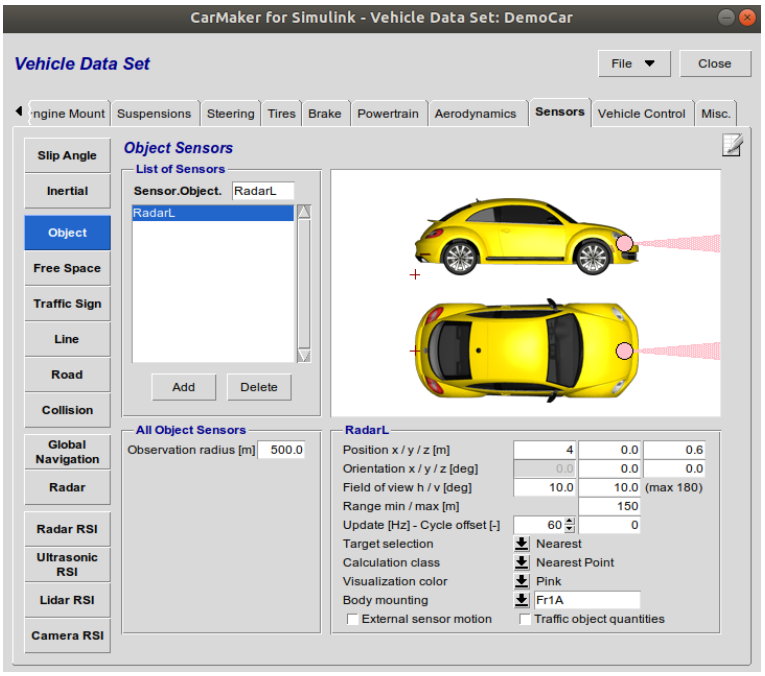

Figure 3.6. CarMaker RADAR selection window

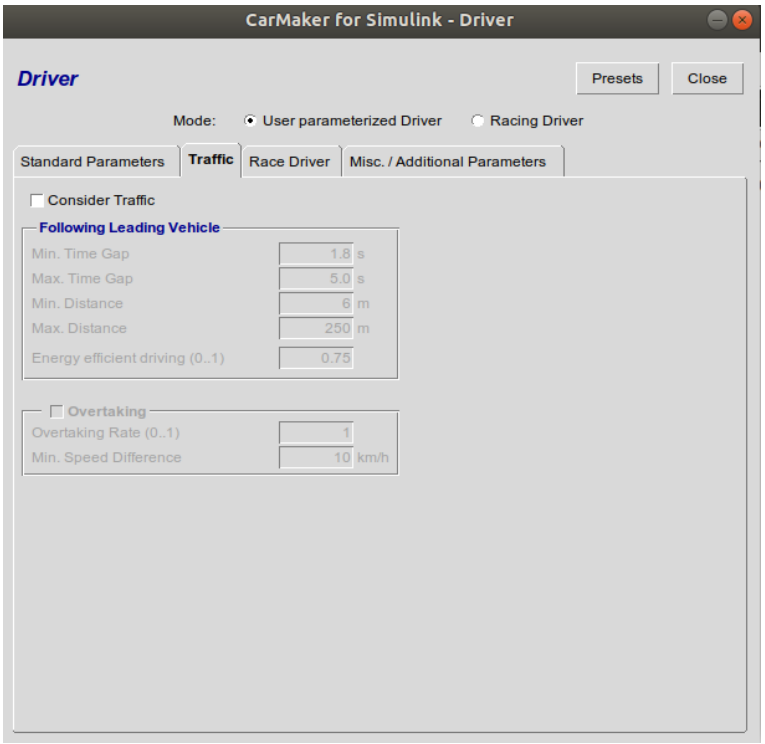

Figure 3.7. CarMaker Driver Window

Given that the Python Engine implements the control strategies based on the information received through the ns3 simulator, we have to avoid the control actions implemented by CarMaker. Normally in CarMaker, the IPGDriver module enables the control actions of a human driver such as steering, braking, gas pedal position, gear shifting, and clutch operation. In order to avoid the IPGDriver reacting to the Traffic objects, we have unchecked the "Consider Traffic" option given by CarMaker under the Driver section as shown in figure 3.7. By doing so,the IPGDrive just follows the manoeuvre mentioned in the test run without considering the traffic objects defined in the TestRun.

To make the CarMaker model ready for a TestRun, we have to define Ego car manoeuvre, Traffic Objects, and their manoeuvres along with the road topology used by the simulation. However, these definitions are not fixed, each of them will vary based on the simulation test scenarios. In this thesis, we have tested the framework with three different test scenarios which will be explained in the next chapter.

3.3.2 Simulink: Simulation Control and Interface

As we discussed in section 2.5.2, the Simulink model can be built as an extension of "generic" model provided by CarMaker. We have made changes in the generic model to make it interoperable with other simulators and to extract the specific data quantiles from the CarMaker environment.

Simulink being a discrete time based simulator , it computes block methods and states at the end of each simulation step. The simulation time step is decided by solver: fixed or variable step depends on the simulation environment. CarMaker for Simulink works with the standard Simulink solvers. The step size of simulation in CarMaker is 1ms. So, for every 1ms Simulink evaluates each block in the model and define their states. On the other hand, ns3 network simulator is a discrete event simulator which executes all scheduled events independent of the time. To make them communicate with each other, we have to pause the Simulink model execution after every predefined time step. We chose to pause the Simulink simulation after every 100 ms . Again, the choice of 100 ms is justified by the fact that only ten CAM messages are allowed for every second, that is 1 CAM for every 100 ms. We have created our own subsystem to pause the simulation after every 100ms. Using simulation clock block, we constantly monitor the simulation time and "pause" the simulation every 100 ms using assertion command.Figure 3.8 shows the Simulation control subsystem.

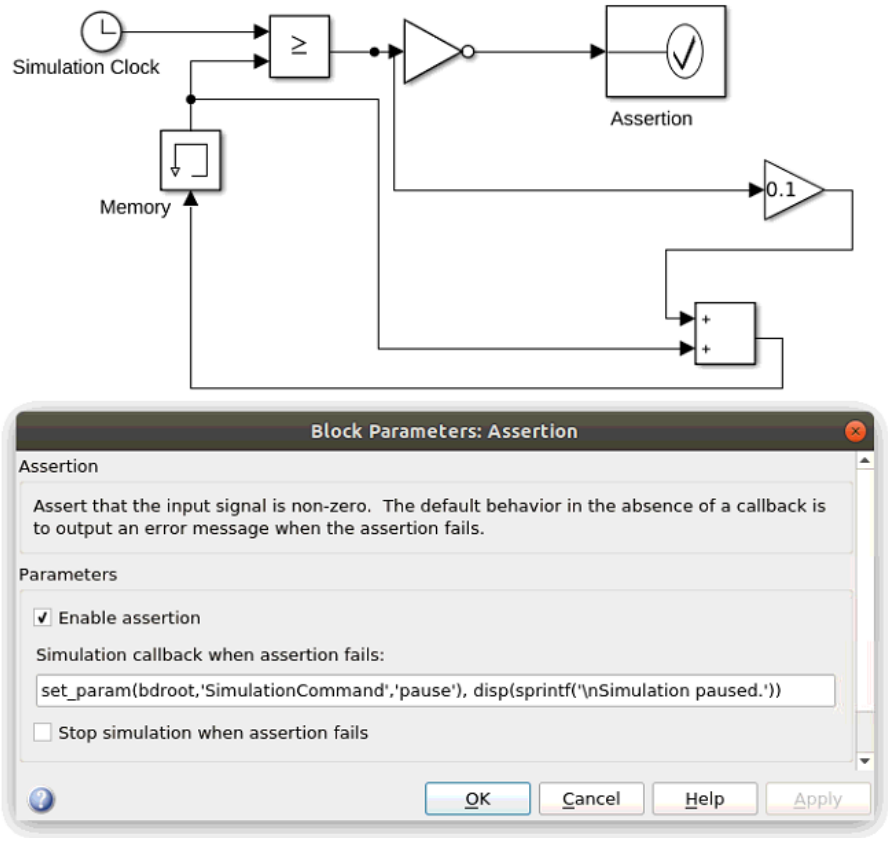

Figure 3.8. Simulink Simulation Control Subsystem

We have used CarMaker for Simulink blocks to get the CarMaker variables listed in table 3.1 in the Simulink environment. But Ego Car is the only fixed vehicle that can be available in all framework simulation scenarios while other traffic actors (Lead Cars) are highly variable based on the simulation scenario. To avoid manually changing the simulation model for each scenario, we have created a Matlab function to create those blocks and connect them automatically for every scenario.

To interact with other applications, Simulink suggests using TCP-IP Send/Receive blocks. Currently, this block only supports a few datatypes like Double, integer, ASCII. Since we have to exchange vehicle information of each vehicle with different data types to the python engine, we are forced to create multiple TCP connections between them which are not advisable. As a workaround, we have concatenated all the vehicle information into a matrix form using a matrix concatenate block as shown in figure 3.9. But Simulink TCP IP doesn't support to send data in a matrix format, so we have exploited

MATLAB RunTimeObject functionality to exchange the information with python engine.

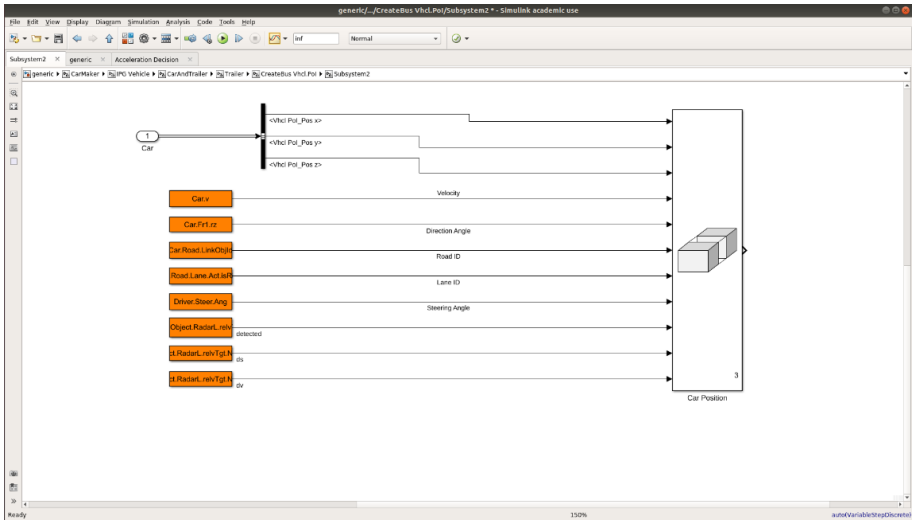

Figure 3.9. Simulink Matrix Concatenate

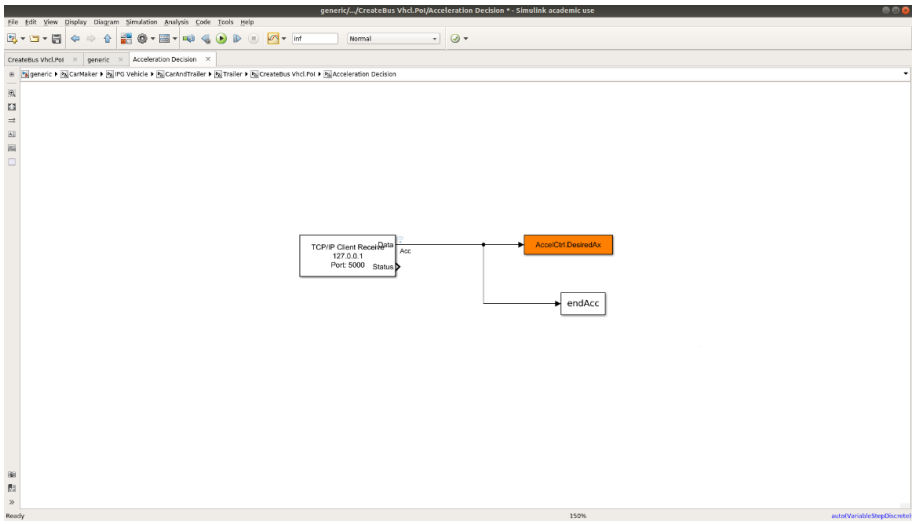

Figure 3.10. Simulink TCP/IP Subsystem

On the other hand, the decision made by the python engine to control the acceleration of the Ego car is a double value which can be received through Simulink TCP/IP block. So, we have used TCP/IP receive block as shown in figure 3.10 to get the desired Ego Car acceleration from the python engine. In the framework, Simulink

TCP/IP blocks acts a client whereas Python Engine acts as a server.By this way, Python engine can directly communicate with Simulink in real-time without using MATLAB as a middle-ware which comparatively consumes more time to transfer the data.

3.3.3 ns3: Communication Model

ns3 being a discrete event simulator, the notion of time doesn't work in the same way as other simulators used in the Framework. By default, ns3 simulator advances the simulation time to the next scheduled event and keep on executing the scheduled events in future without waiting. In order to integrate with other applications, ns3 has an option to run simulations in a real-time with the help of real-time scheduler. In real-time scheduler, instead of advancing the simulation time ns3 moves it synchronously with the machine clock. By this way, after executing an event, ns3 scheduler compares the current time and next scheduled event time. If the event is scheduled in the future, ns3 goes in to sleep state till the scheduled time and then executes the event. Since this is a co-simulation framework and it involves multiple simulators, we have used real-time scheduler in ns3 to schedule the events.

In this co-simulation framework, we have adopted cellular vehicle-to- Infrastructure as our communication technology. In ns3, LTE module is implemented by the LENA simulator.LENA is an open source LTE/EPC Network Simulator that helps us to design and evaluate the performance of LTE framework.The LENA code is merged with the official version of ns-3.

As we know, the primary nodes of LTE module are UserEquipment(UE) and eNodeB(eNB). In this framework, we have placed eNodeB at the centre of the topology in all scenarios. The traffic scenarios were built with the consideration of placing eNodeB at (0,0,0) position. In ns3 "ConstantPositionMobilityModel" was used to keep eNodeB at the centre of the topology for whole simulation period.

With the help of Evolved Packet Core(EPC) helper class, ns3 automatically create a PGW node to handle the traffic from/to LTE radio access network. In simple words. with the help EPC helper class , ns3 able to create end-to-end IP connectivity between multiple UE's and remote hosts. We have created a remote host to act as an infrastructure node to implement the longitudinal control algorithm. The connection between PGW node and the infrastructure node was established via a point-to-point link.

In this case vehicles were considered as UserEquipment's. The number of UE's varies according to the simulation scenario. In every simulation step, the position of UE's nodes is updated based on the vehicle position in CarMaker. The connection between UE's and remote host was established with the help of UDP sockets.

To recreate a real-world urban environment, we have created buildings around eNodeB by using the ns3 Buildings module. The buildings module also implements the propagation loss model to be used with the LTE module along with the consideration of wall penetration losses. Based on the node's position , buildings module decides whether the node is indoor or outdoor. In our case, all the nodes are considered as outdoor nodes and corresponding propagation loss model is used.

Since Python Engine plays a role of integrator between multiple simulators, all the above mentioned ns3 functionalities are scripted in Python with the help of ns3 Python bindings and simulation script is integrated into the Python Engine.

3.4 Co-Simulation Interface

3.4.1 Interaction between CarMaker and Simulink

As we discussed earlier in section 2.5.2, CarMaker for Simulink helps to create a tightly coupled simulation framework between CarMaker and Simulink. In this section, we have discussed the interaction between CarMaker and Simulink and how we were able to exchange the values with each other.The variable exchange between CarMaker and Simulink became possible with the help of Direct Variable Access Method. CarMaker for Simulink blockset has two blocks that is used to manipulate the variable in CarMaker environment. Those two blocks are *"Read CM Dict"* and *"Write CM Dict"*. However, these blocks can be used to get the variables that were already defined in CarMaker data dictionary. If we want to read other values from CarMaker environment, we have to use *"Define CM Dict"* to create those variables in its data dictionary.

"Read CM Dict" reads a variable in CarMaker dictionary and deliver it in the Output port of the block in real-time. For example, if we want to know the velocity of the Ego car, we can use *"Car.v"* as a variable name in the block and get the current velocity as shown in figure 3.11. The name of the variable is based on the property of the vehicle we would like to read from CarMaker, and this can be known easily from DVA section of CarMaker GUI. In this simulation framework, we are using RADAR object sensor to detect the traffic actors around Ego Car.Table 3.2 has the details of the variable name used to get value from CarMaker Ego Car and lead cars where [TName] varies with the name of the traffic object.

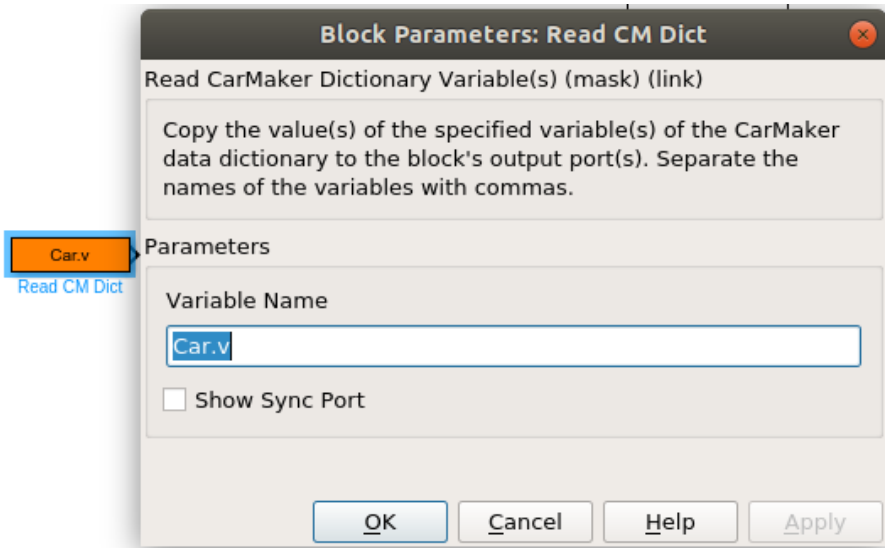

Figure 3.11. CarMaker Read block in Simulink

Information	Ego Car	Lead Car
<EPre>:Sensor.Object.RadarL.relvTgt;	<LPre>:Traffic.[TName]	
Vehicle Position	Car.tx Car.ty Car.tz	<LPre>.tx <LPre>.ty <LPre>.tz
velocity	Car.v	<LPre>.LongVel
Heading	Car.Fr1.rz	<LPre>.rz
Link ID	Car.Road.LinkObjId	<LPre>.LinkObjId
Lane ID	Car.Road.Lane.Act.isRight	<LPre>.Lane.Act.isRight
Steering Angle	Driver.Steer.Ang	<LPre>.SteerAng
Road ID	Car.Road.sRoad	<LPre>.sRoad
Radar Object Detection Radar relative distance Radar relative speed	<EPre>.dtct <EPre>.NearPnt.ds_p <EPre>.NearPnt.dv_p	Not Applicable

Table 3.2. CarMaker Variable Dictionary [30]

On the other hand,*"Write CM Dict"* is used to write updated value in CarMaker data dictionary. In this framework, we are updating the Acceleration of the Ego Car in CarMaker environment based on the output of control algorithm using *"AccelCtrl.DesiredAx"* as a variable name. Since this variable is not available in CarMaker data dictionary, we have created it using *"Define CM Dict"* and then update its value using *"Write CM Dict"* on every simulation step as shown in figure 3.12.

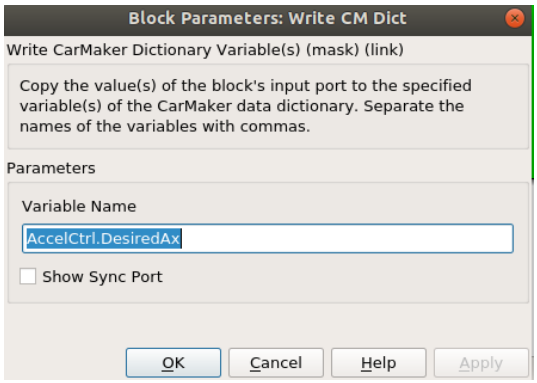

Figure 3.12. CarMaker Write block in Simulink

To make CarMaker aware of the fact that Ego Car Acceleration is controlled by an external environment, we have to mention the controller of Acceleration Control and ACC in CarMaker as DVA(Direct Variable Access). In this way, the acceleration of Ego Car in the CarMaker environment is controlled by the Simulink through *"AccelCtrl.DesiredAx"* variable. Figure 3.13 shows the method used to transfer the desired acceleration from Simulink to CarMaker.

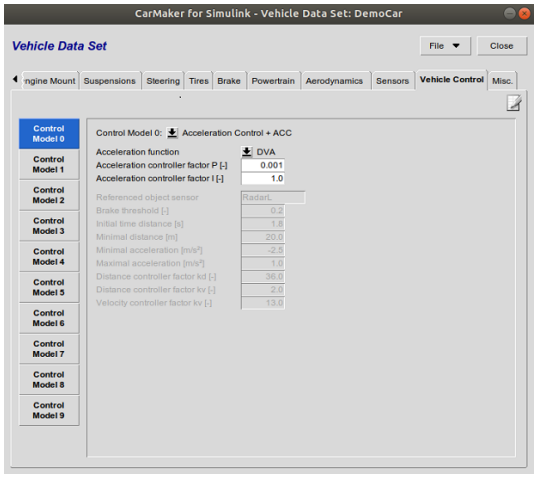

Figure 3.13. CarMaker ACC Control Model

3.4.2 Interaction between Simulink and MATLAB

Simulink provides an API called block run-time interface, to access the blocks data programmatically during the simulation. By using this interface, it's possible to access the run-time data from the MATLAB command line.

For accessing a Simulink block data in run-time from MATLAB, we have to create run-time object of that particular block. We can create a run-time object of a block in MATLAB by using *"get_param"* in command line. *"get_param(Object, Parameter)"* return the value of the specified block. In our case , object is the Simulink block name and parameter is *"RuntimeObject"*. In our case, we needed to access the blocks that contains the vehicle information. As discussed earlier in the Simulink Model, we have concatenated all the vehicle information into a Matrix form using Matrix Concatenate block. So, we created run-time objects for every matrix concatenate block that is based on the number of vehicles involved in the simulation. In this way, we have accessed the vehicle information from MATLAB command line in run-time.

We can also control the Simulink simulation execution from MATLAB command line by using *"set_param" command*. *"set_param(Object,ParameterName,Value)"* is used to set specified value to the blocks parameter. To control the simulation, we have to use simulation Model Name as Object, *"SimulationCommand"* as parametername and we can choose one of *["start","pause","continue","stop"]* as value. For example, *set_param("generic.mdl","SimulationCommand","pause")* is used to pause the running simulation.

With the help of these commands, we have manipulated the Simulink simulation process to synchronize with other simulators.

3.4.3 Interaction between MATLAB and Python Engine

MATLAB provides a Python package that enables Python to use MATLAB as a computational engine. The MATLAB package contains MATLAB Engine API for Python and a set of MATLAB arrays that can be used as Python Variables. The engine provides functions to call, and the array classes provide functions to create MATLAB arrays that can be used to send as arguments for functions. MATLAB Engine API provides two different ways to access the MATLAB.

start_matlab - starts a new MATLAB process, and returns engine object for communicating with the MATLAB. We can start the MATLAB process with/without desktop from Python by using start-up options. However, without-desktop(headless) option restrict us from using certain functionalities.

connect_matlab - connects to the shared MATLAB session which is already running on the local machine and returns an engine object as eng. Before using connect_matlab, we have to share the running MATLAB process by executing matlab.engine.ShareEngine command in its command window. We can identify the shared MATLAB process in python with the help of find_matlab. It finds all shared sessions on the local machine and returns their names in a tuple. Therefore, we can connect to the shared MATLAB process from Python using connect_matlab (find_matlab()[desiredIndex]) function.

After connecting to a MATLAB process, we can access its workspace variables from Python using eng.workspace[variableName]. We can also take advantage of all the functionalities of its command window from python with the help of eng.eval (command,nargout=) where eng represents connected MATLAB process, nargout used to specify the number of output arguments MATLAB returned to python environment.

We have combined eng.eval function with MATLAB commands explained in the previous section 3.4.2 to control the Simulink simulation from Python Engine.We have paused the Simulink simulation using following command:

$$eng.eval(''set\_param(''generic.mdl'',''SimulationCommand'',''pause'')'')$$

With the help of following commands, we have accessed the vehicle information in Simulink from Python engine.

$$eng.eval(''rto = get\_param(''generic/carInfo'',''RuntimeObject''), nargout = 0)$$

$$eng.eval(''rto.OutputPort(1).data''), nargout = 1)$$

3.5 Information Flow

In the previous section 3.4, we have discussed the interaction between the simulators while in this section we have explained the simulation control and its process along with the details about the data handled by each simulator. The simulation is primarily controlled by a ns3 event that is scheduled to run every $100ms$. Before starting the simulation, the Python engine initialises the simulation framework with four steps. First, it finds a shared MATLAB environment and connects to it. Second, it executes the MATLAB function to create a list of Simulink blocks that extract vehicle information from the CarMaker environment. Third, it establishes a TCP/IP connection with Simulink, where Python Engine and Simulink acts as a server and client respectively.

And finally, a list of ns3 nodes are created to represent the vehicles involved in the simulation. Once the initialisation phase is completed and successfully create a connection between all the simulators, the Python engine starts the simulation by scheduling a ns3 event. We have developed a system UI as shown in figure 3.14 to simplify the above-said simulation execution process. All the above four steps can be performed with the click of the initiate and run button. The initiate button identifies the number of cars that are going to be simulated from the CarMaker TestRun file while the Run button executes the above steps one by one in the above-mentioned order.

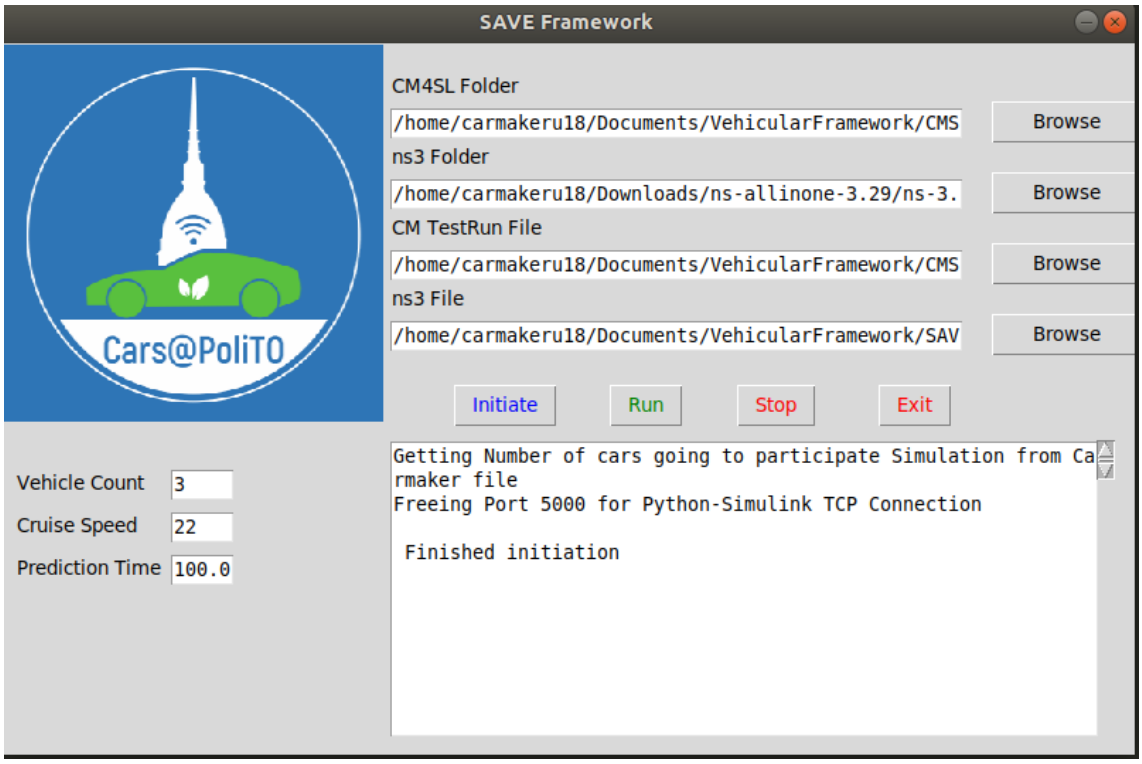

Figure 3.14. System User Interface

At every simulation step, ns3 scheduled event checks the status of the CarMaker simulation using MATLAB command through the Python engine. If the simulation is running, it extracts each vehicle data from Simulink through MATLAB command. In order to include real-world errors in simulations, the Python engine adds a random normal error to the vehicle position data extracted from CarMaker. After processing the vehicle information, ns3 schedules an event where a respective vehicle node is used to send that information to the infrastructure node using the LTE module.

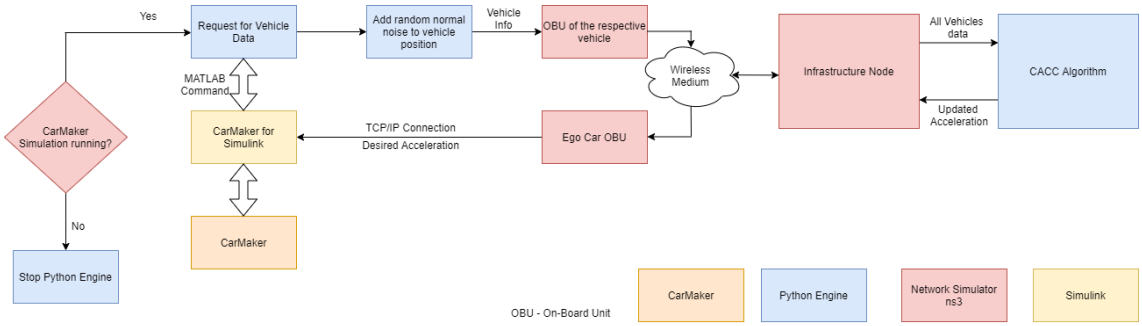

Figure 3.15. Information Flow

Once the infrastructure node receives information from all vehicles involved in the simulation, it executes the control strategies algorithm. The algorithm implements enhanced CACC control which uses both Ego car RADAR signal and surrounding vehicle information received by infrastructure node to calculate the desired acceleration for Ego Car. The details of the control strategies are explained in section 3.6. After finding the desired acceleration, the infrastructure node sends it to the ns3 Ego Car Node, again using the LTE module. While Ego car Node transfer that data to Simulink using TCP/IP connection and Simulink update it to the CarMaker environment. This flow of information concludes a simulation step that gets repeated until the completion of CarMaker TestRun. The information flow blocks along with their respective simulator are shown in figure 3.15.

3.6 Control Strategy

The main idea of the control strategy algorithm is to implement the Co-operative Adaptive Cruise Control system to avoid any collision by adjusting the speed of Ego car based on Radar sensor input as well as CAM messages received from the infrastructure node. In any case, control algorithm is executed for both of inputs and minimum acceleration between them is used as desired acceleration. The acceleration/deceleration of Ego Car depends on the sign of the chosen acceleration as shown in table 3.3.

Acceleration	Action
Positive	Accelerate
0	Maintian Current Velocity
Negative	Brake

Table 3.3. Ego Car Action based on Acceleration sign

As per Adaptive Cruise Control system, if no object is detected the Ego car maintains the desired cruise speed; if any object is detected it maintains the desired distance gap with the detected vehicle. In this framework, we have chosen desired time gap between vehicles as 0.6 seconds. The desired distance gap is calculated by multiplying desired time gap and current Ego car velocity.

The control algorithm works based on proportional control system with predefined proportional gains for RADAR detected objects and variable gains for Lead Cars detected via CAM messages. The proportional control is a feedback control system which is proportional to the difference between desired and actual value. The control algorithm is divided into two sections: Adaptive Cruise Control (ACC) and Automatic Emergency Braking (AEB).

As explained in section 2.3.2, AEB activated when the collision is imminent. It depends on the Time-To-Collision (TTC) value. TTC is the measure of the amount of time until the collision occurs. It is calculated by considering relative distance and velocity of the target vehicle with Ego Car. AEB with maximum deceleration is activated if TTC goes below 1 second.

$$Acceleration = -2.6m/s^2, \forall 1 \leq TTC \leq 2.4$$

$$Acceleration = -9.6m/s^2, \forall TTC < 1$$

ACC uses relative distance and velocity of the target vehicle, desired distance, and speed set by Ego car as input to calculate desired acceleration. The proportional gains used in ACC are k_{v1}, k_{v2}, k_d, k_{dd}. The Lead Car relative velocity with Ego Car uses $''k''_{v1}$ and the difference between the relative distance and the user defined desired distance uses $''k''_d$ as their proportional gain. In any case, Ego car velocity is not supposed to be higher than the desired velocity set by the user, so ACC calculates the difference between the desired velocity and current Ego car velocity with $''k''_{v2}$ as their proportional gain. The sign of their difference decides whether Ego car has to accelerate or decelerate. If target vehicle velocity is higher than the Ego car but the distance between them is less than the desired distance, Ego can decelerate with lower gain compared to $''k''_d$. Therefore, we have used $''k''_{dd}$ as a proportional gain which is considerably less than $''k''_d$. Figure 3.16 shows the ACC control scheme along with the importance of proportional gains [27].

However, the acceleration/deceleration rate is restricted to certain limit to ensure the comfort, safety of the passengers. The gains and acceleration/deceleration limits of ACC are mentioned in the table 3.4.

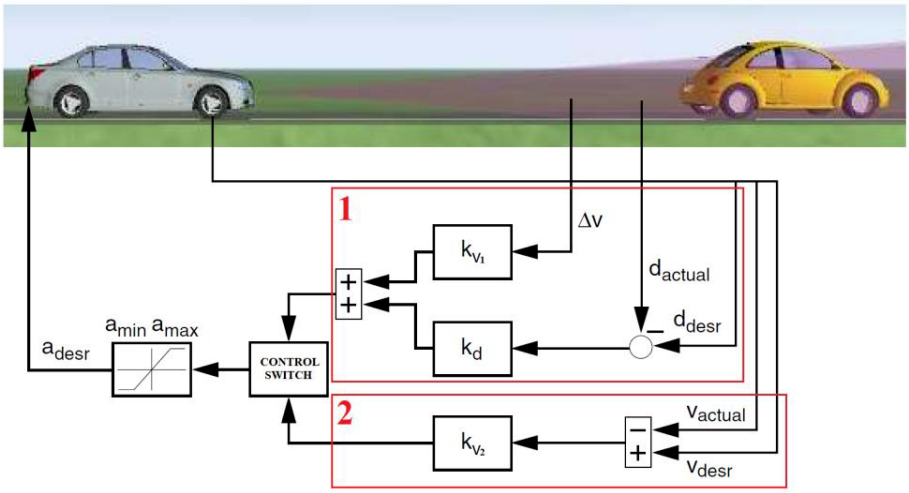

Figure 3.16. ACC Control Scheme [27]

Specifications	Value Limits
Max deceleration (ACC) $[m/s^2]$	-2.45
Max deceleration (AEB) $[m/s^2]$	-9.6
Max acceleration $[m/s^2]$	1
k_d	0.07
k_{dd}	0.01
k_{v1}	0.45
k_{v2}	0.3

Table 3.4. ACC Gain and Acceleration Limits [27]

The control algorithm is executed with the above-mentioned gains for radar detected objects. However, for Lead Cars that do not fall under the range of radar, we use a trajectory-based control system to predict the collisions. For every vehicle involved in the simulation, we predict the vehicle positions for the next 10 seconds with 0.1 seconds as a difference between two predicted time instances. To predict the positions, we are using current position, heading direction and velocity from CAM messages. With the help of LaneID, RoadID, LinkID information from CAM message, we are finding the Lead Cars that are in the collision course with Ego car in the next 10 seconds. Since a trajectory-based system can only detect the collisions that occurs in the straight path based on the current heading direction, we tend to miss the presence of vehicles in the corners. By combining road network information's like LaneID, RoadID and LinkID with the positions, we are able to detect the presence of Lead Cars in the corners.

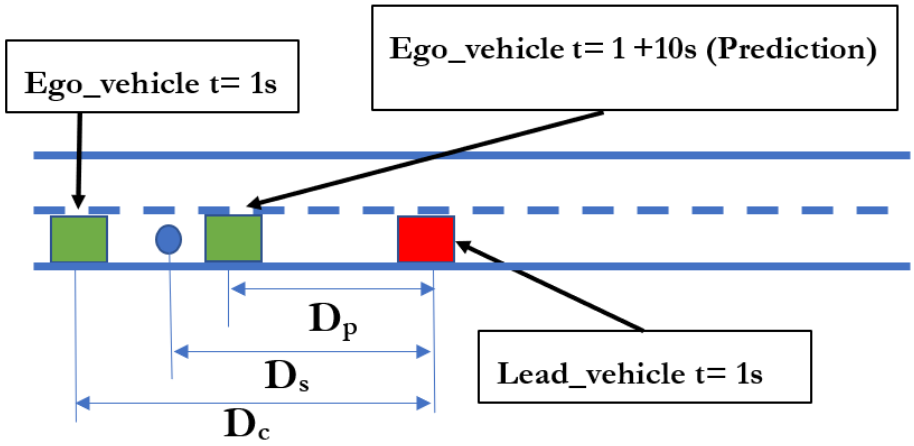

Figure 3.17. Distances Representation in Control Algorithm

After defining the position of the cars for next 10 seconds, we are calculating the distance between Lead Car and Ego car as shown in figure 3.18. If the calculated distance (D_p) is below the safe distance (D_s) at any point in next 10 seconds , we execute the control algorithm to find the desired acceleration. The safe distance is calculated based on the specifications mentioned by ACI (Italian Automotive Club)[28]. The trend of safety distance is shown in figure 3.18.

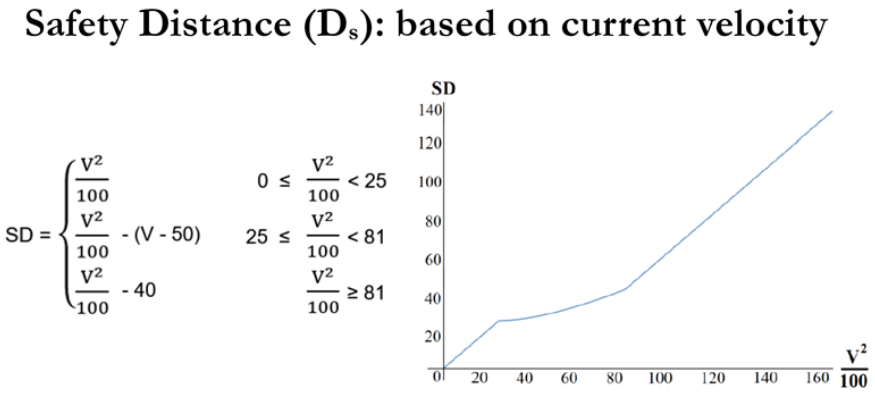

Figure 3.18. ACI Safety Distance [27]

However, we can't execute the control algorithm with normal gains mentioned in the above table because with the current distance(D_c) between Ego car and the vehicle in question normal gains doesn't have impact in defining the acceleration. To make the control algorithm react to the collision that might occur in the future, we have used the variable gains. Among four proportional gains, $''k''_{v1}$ and $''k''_{d}$ plays major role in

calculating the desired acceleration. Since, we are predicting for 10 seconds with 0.1 as a time difference, We have created 100 data points between 0 and 0.45 in case of $''k_{v1}''$ and between 0 and 0.07 for $''k_d''$ that follows the pattern shown in the figure 3.19.

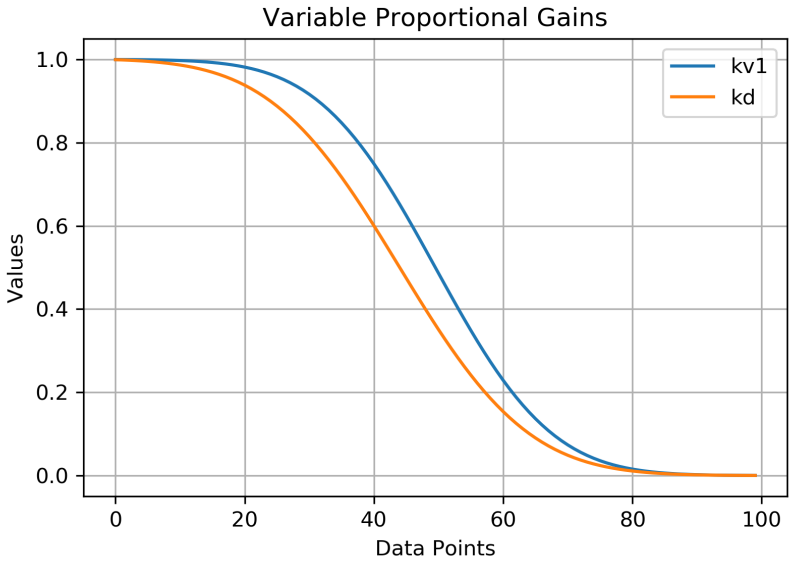

Figure 3.19. Trend of Variable Proportional Gains

Among the 100 data points, the control algorithm chooses the proportional gain based on the index of the minimum predicted distance between Ego Car and Lead Car. With those gain values, control strategy executes the algorithm with Lead vehicle current velocity, distance and Ego car velocity as input to identify the desired acceleration.

Currently, we have two accelerations; one from RADAR based data and another one from Lead Cars that doesn't fall under the RADAR range. The Control algorithm choose the one with minimum acceleration as the Ego Car desired acceleration for the next simulation step. The flow-chart of the entire control algorithm is illustrated in figure 3.20.

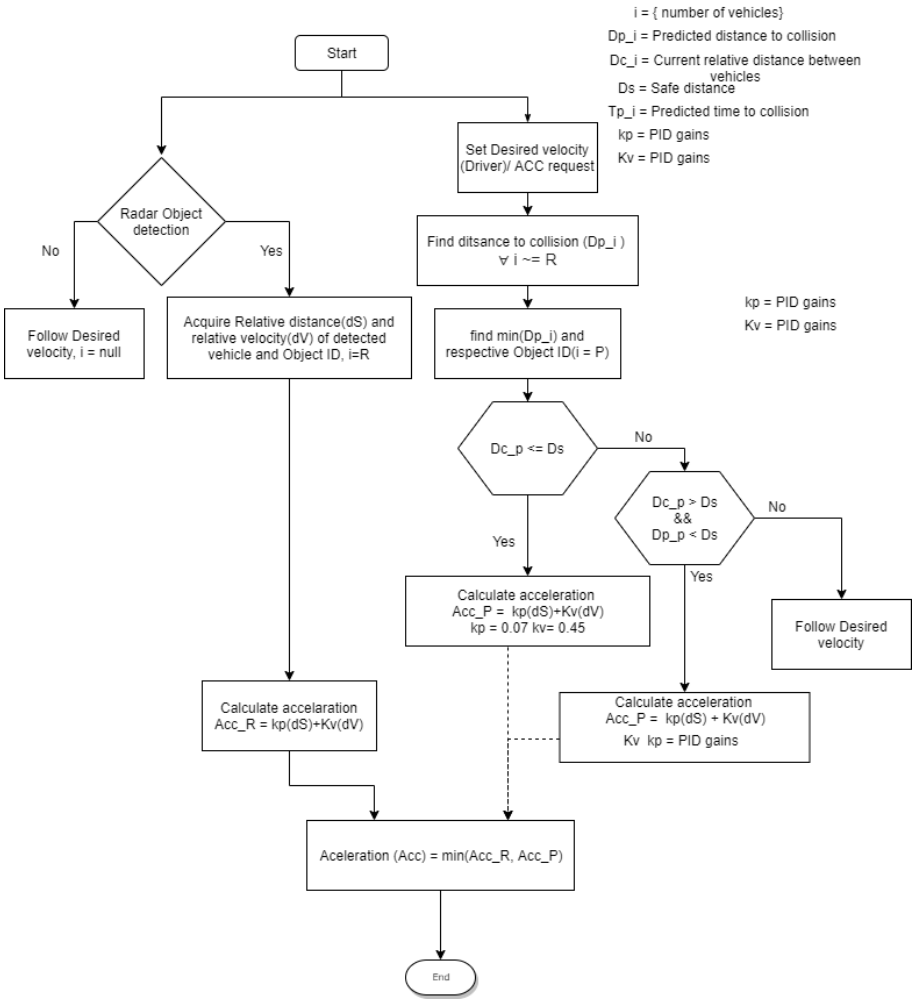

Figure 3.20. Control Algorithm Flow Chart

Chapter 4

Simulation Scenarios and Results

4.1 Simulation Scenarios

The main goal of this framework is to study the impact of the connected vehicles in comfort and safety-related applications. In order to attain this goal, we have tested the framework with a set of scenarios that can occur in the real-world. These tests are performed to evaluate the performance of the control strategies using collision avoidance as one of the key performance indicators. The parameters used by the control algorithm are described in table 3.1, which are extracted from CarMaker and communicated to the infrastructure through the ns3 simulator. The default time headway between Ego Car and Lead Car is fixed at 0.6 seconds and the desired cruise speed is varied between 22m/s and 16 m/s based on the simulation scenario. Furthermore, the vehicle acceleration is limited to $1 m/s^2$ while the maximum deceleration (in case of AEB) is capped at -9.6m/s^2 . The limits are listed are in table 3.4. The topology of the road varies based on the simulation scenario.

We have tested the framework for three different scenarios. They are:

1. Vehicles crossing in a T-Junction

2. Preceding vehicle Cut-out from the lane

3. Slower/Stationery vehicle in the Corner

In order to point out the situations where vehicle sensors are not enough to mitigate the collisions, we have simulated these three scenarios for two different modes.

- Vehicles with sensors only
- Vehicles with Sensors and OBU (Connected Vehicles)

Furthermore, in the case of connected vehicles, we have executed the control algorithm for different forecasting time instances(5, 7, 10 seconds) using a trajectory-based prediction model.

4.1.1 Scenario-1: Vehicles crossing in a T-Junction

In this scenario, we are using two vehicles(Ego car and a Lead Car) which are going to meet at a T-junction. The vehicles and their manoeuvres are pre-defined in CarMaker TestRun. The road topology also created in CarMaker using Scenario Editor. The Ego car is travelling in a constant velocity of 80kmph in a straight route, mentioned as Route 1 in figure 4.1.The lead car is travelling at a lower speed while approaching the T-junction and after crossing the T-junction Lead Car accelerate for 10 seconds with acceleration rate as 1 m/s^2 as shown in figure 4.2. The route of Lead Car is marked as Route 2 in figure 4.1. The scenario is created in a way that the lead car falls under the blind spot of the Ego car RADAR while crossing the T-junction. The Ego Car RADAR range is shown in figure 3.6. With the Lead Car being blind-sided, Ego Car tends to maintain the desired cruise speed. We have analysed the reaction of Ego Car when it detects the presence of Lead Car in both the modes.

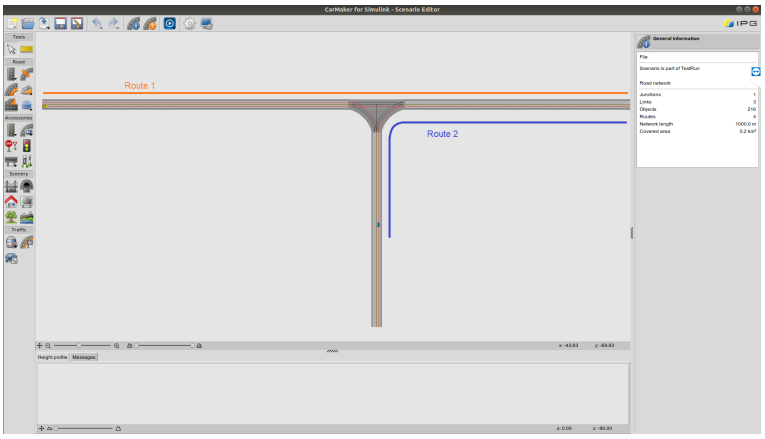

Figure 4.1. CarMaker Scenario Editor

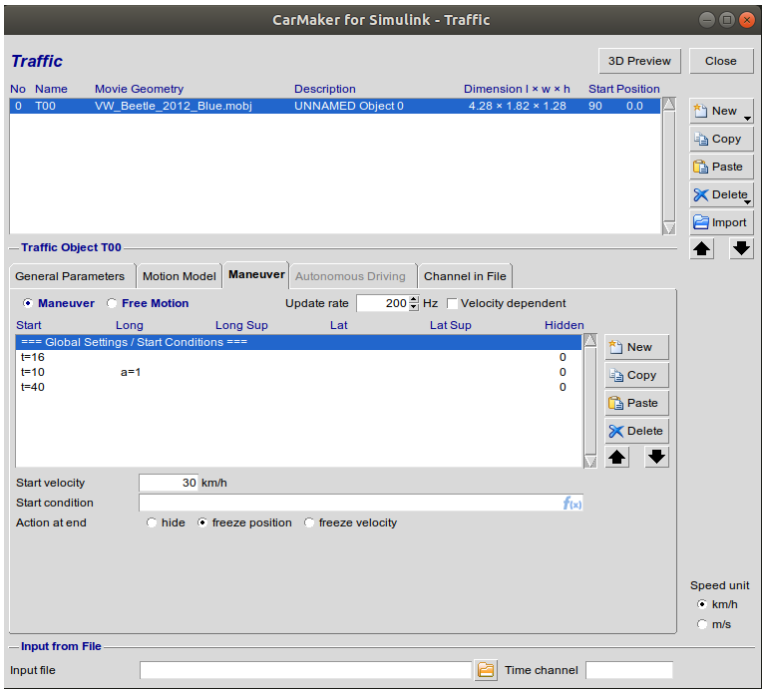

Figure 4.2. Lead Car Manoeuvres

4.1.2 Scenario-2: Preceding vehicle Cut-out from the lane

In the second scenario, we are simulating a cut-out scenario where a Lead Car is changing a lane at a last minute to avoid a collision with another Lead Car in the same lane.Figure 4.3 shows the pictorial representation of this Scenario. In this case, we are using 3 vehicles (Ego car and two Lead cars) in a straight road with two lanes. The Ego car is following a Lead car which is travelling at 80kmph while a Lead Car in front of it is travelling at a speed of 80kmph. Since, Ego car desired speed is 80kmph, it maintains a time-headway of 0.6 seconds with the preceding Lead car. Suddenly the latter changes the lanes to avoid the slower vehicle in front of it,figure 4.4 shows the manoeuvre defined for the LeadCar in CarMaker to avoid the collision. The Ego car radar has less time to change its focus to new target and react accordingly to avoid the collision at higher speed. We have analysed the use of CAM messages in this scenario to identify the presence of slow moving vehicle beforehand and react to it promptly.

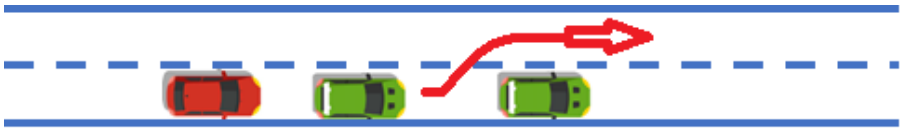

Figure 4.3. Pictorial Representation of Cut-Out Scenario

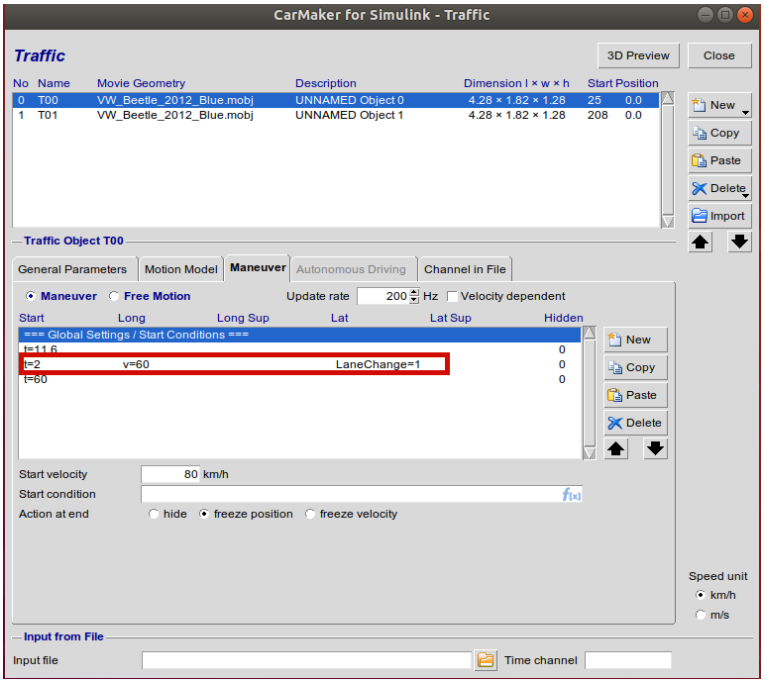

Figure 4.4. Lead Car Cut-Out Manoeuvre in CarMaker

4.1.3 Scenario-3: Slower/Stationary Vehicle in the Corner

In this scenario, we have tested the limitation of RADAR to detect the presence of a vehicles in a corner. Figure 4.5 shows the road topology used for this simulation scenario. To make the scenario more realistic we have added buildings near the corner which could possibly reduce the visibility of the driver. For this simulation, we have used two cars (Ego car and Lead car), where both the vehicles are approaching a corner. The Ego car and Lead car are travelling at 60 kmph and 45 kmph respectively, with a considerably longer distance between them. The desired cruise speed of Ego Car is 60 kmph, so it maintains the current velocity along with the desired time headway. While approaching the corner, Lead car broke down due to some mechanical malfunction and came to a stop. Figure 4.6 shows the Lead Car manoeuvre definition in CarMaker. Given that Ego car radar won't be able to see the static vehicle at the corner, it continues to travel

at higher relative speed. We have analysed the different courses of action performed by the Ego Car to avoid the collision in the two modes.

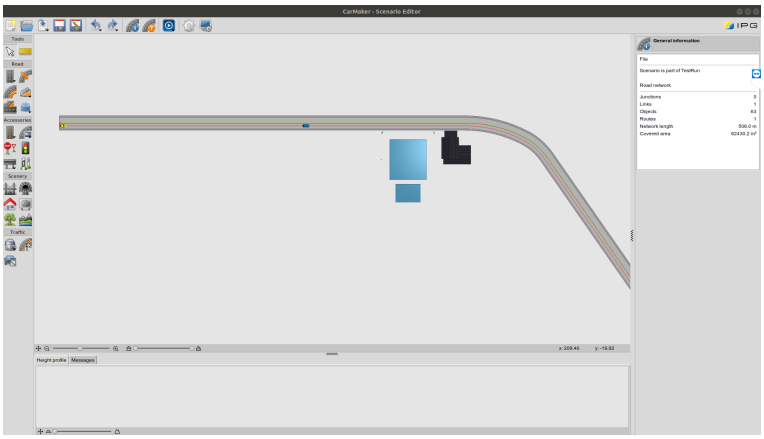

Figure 4.5. Road Topology of Scenario3

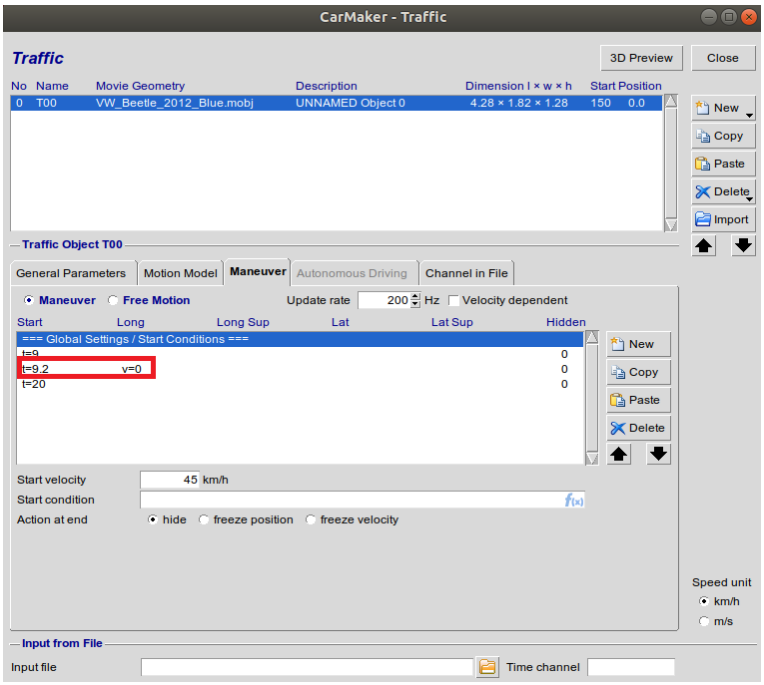

Figure 4.6. Lead Car Manoeuvre in CarMaker

4.2 Results

As discussed earlier, we have simulated all the scenarios for two modes i.e with and without OBU(connected vehicles). In a case of the connected vehicles mode, Ego car should be able to mitigate the collision without activating Automatic Emergency Braking (AEB) mode because the system constantly monitors the movements of the vehicle. The control algorithm uses 10 seconds as its default prediction time frame.

4.2.1 Scenario 1

The goal of this scenario is to identify the potential collision occurrence in the upcoming T-Junction and change the Ego Car velocity to avoid the collision by accelerating/decelerating the vehicle.

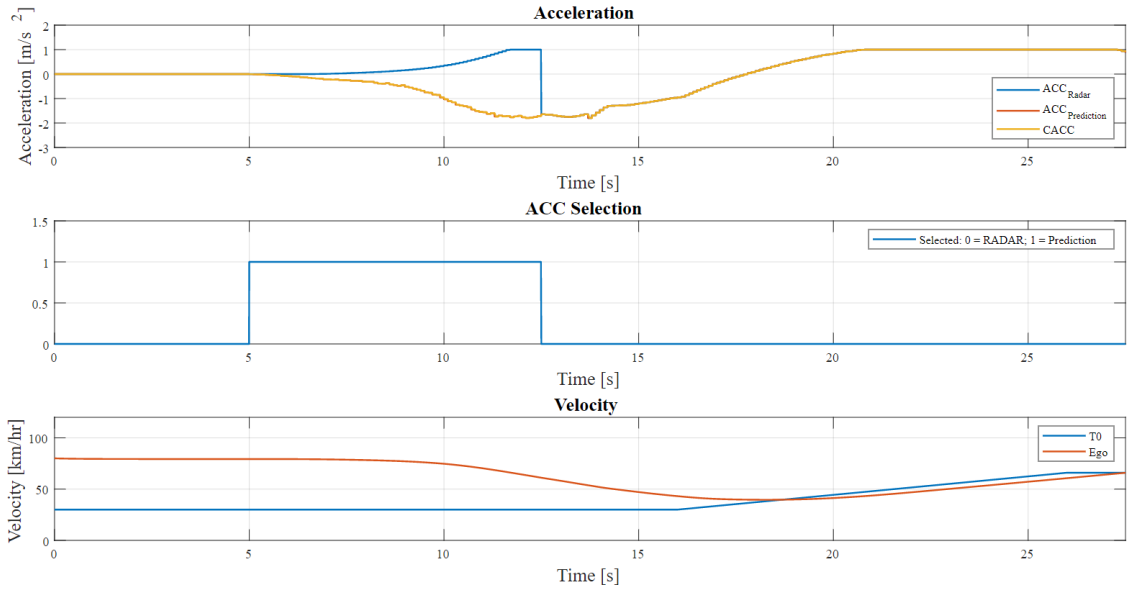

Figure 4.7. Scenario1: Ego Car Acceleration and Velocity variation based on RADAR and CAM Messages

The figure 4.7 show the Acceleration chosen by the control strategy algorithm by considering RADAR and CAM messages as input. It has three sub-plots. The Acceleration sub-plot shows the acceleration output based on RADAR and CAM messages as well as the acceleration used by Ego Car among them. The ACC selection sub-plot indicates whether the control algorithm used ACC or CACC controller to define the desired Ego Car Acceleration. The Velocity sub-plot shows the velocity of the vehicle

involved in the simulation, where T0 indicates the Lead Car. We can see from the acceleration sub-plot, RADAR didn't have any vehicle on its range, so it keeps maintaining the desired cruise speed of the Ego Car as shown in velocity sub-plot. With the help of CAM messages, the control algorithm identifies the presence of other vehicles in the network and it predicted that the slow-moving vehicle is in the collision course with the Ego Car. Once the distance between Ego Car and slow-moving vehicle comes below the safe distance, the control algorithm calculates the desired acceleration value to avoid the collision and communicated it to Ego Car. Since we are foreseeing the collision that can potentially occur in the future, Ego Car reduces the velocity gradually as happened in velocity sub-plot of the figure 4.7. From Acceleration sub-plot of the figure 4.7, we can also notice that Radar-based acceleration is increased to compensate the velocity reduction. The reason behind this behaviour is because there is no vehicle in the RADAR range of the Ego car, so it tries to maintain the desired cruise speed. Once the vehicle falls under the RADAR range, the system also switches its primary focus from Predicted to RADAR based Acceleration as shown in selection sub-plot of the figure 4.7. This can be confirmed from the second sub-plot that shows the switch between predicted to RADAR based acceleration mode. As mentioned in figure 4.2, after crossing T-junction, the Lead Car accelerates to 60 km/h. we can see that the Ego Car also accelerates gradually to maintain the desired distance gap between Ego and Lead Car. The rate of change of Ego Car acceleration is smooth without any abrupt deceleration/acceleration which ensures passenger comfort and safety.

The figure 4.8 shows the simulation with and without OBU. ACC indicates RADAR is the sole source of data for Ego car to decide the acceleration while CACC indicates the usage of CAM messages to calculate the acceleration. In this scenario, Lead Car enter in the RADAR range at the last moment, so the Ego Car activates the pre-brake (-2.6 m/s^2)in AEB mode for a brief time. Given that, the reduction in velocity is not enough to avoid imminent collision, it uses full brake force of -9.6 m/s^2 to avoid the collision. We can notice the steep reduction of velocity in the velocity sub-plot of figure 4.8. On the other hand, with the help of CAM messages, Ego Car detects the collision earlier and applies the brake gradually. This can be confirmed by the smooth reduction of velocity in velocity sub-plot of figure 4.8.

The figure 4.9 shows Ego Car Acceleration trend based on different forecasting periods (10,7,5 seconds). For this scenario, the prediction period of 7 seconds perform better than the others. With forecasting time as 10 seconds, the acceleration is smooth at the initial phase, however it is not sufficient to keep the safe distance between the vehicles. So, at the later stage it chooses higher deceleration value compared to 7 seconds.

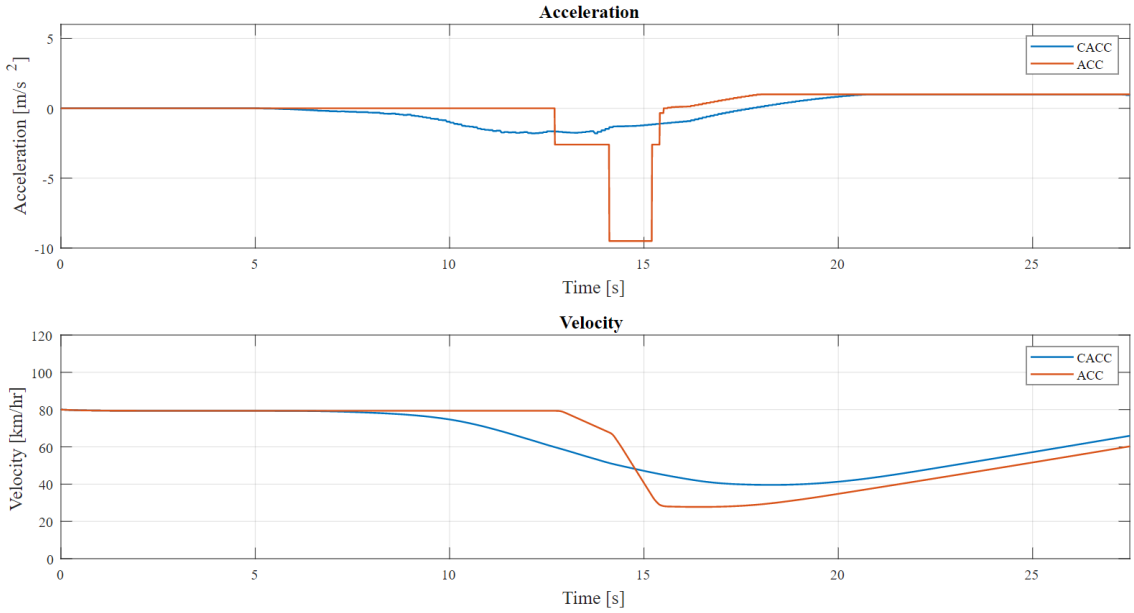

Figure 4.8. Scenario1: ACC vs CACC Comparision

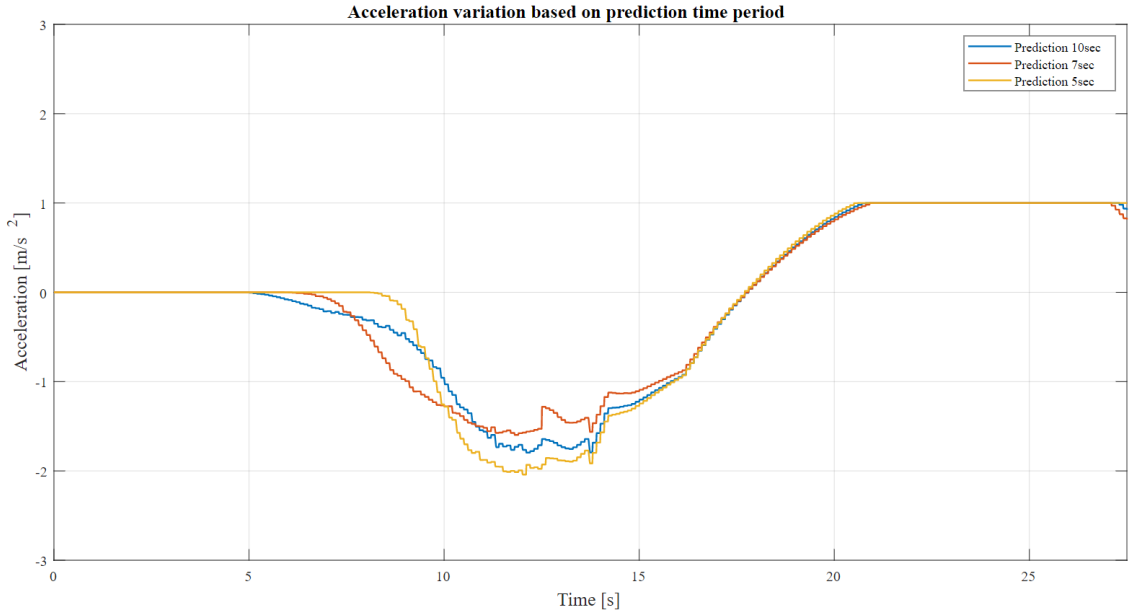

Figure 4.9. Scenario1: Acceleration Variation for different Prediction Time Instances

The plots presented in scenario 2 and scenario 3 follows the same pattern as of scenario 1. To reduce the repetition, we have explained only the differences in the following two scenarios.

4.2.2 Scenario 2

The goal of this scenario is to detect the presence of a slower vehicle in the same lane as of Ego Car and use CACC to maintain the desired time head-way between Ego Car and Lead Car.

In this scenario, we have 3 cars moving in the same lane. In the velocity sub-plot of 4.10 T0 and T1 represents Lead Car(1) and Lead Car(2) respectively. As shown in the velocity sub-plot of figure 4.10 the Lead car(1) which is in front of Ego Car is travelling at the desired cruise speed of Ego Car. Therefore, Ego Car maintains the desired distance gap with zero acceleration as shown in Acceleration plot of 4.10. However, once the control algorithm detects the presence of a slower vehicle in the same lane travelling in the same direction, Ego Car decelerates to maintain the desired time headway with the Lead Car(2) which is in front of the Lead Car(1). As expected, RADAR based acceleration tends to increase to maintain the cruise speed which is based on Lead Car(1) velocity. Once the Lead Car(1) realises the presence of Lead Car(2), it moves to the adjacent lane to avoid the collision. This abrupt action of Lead Car(1) brought Lead Car(2) in the RADAR range of the Ego Car. After the cut-out, Ego Car chose RADAR based acceleration to maintain the desired distance while Prediction based Acceleration chose to accelerate by considering the Lead Car(1). As per the design, the control algorithm chooses the minimum acceleration between Prediction and RADAR as the desired acceleration of Ego Car. The trend of ACC and CACC comparison figure 4.11 is similar to scenario 1. However, since the Lead Car(1) changes the lane at the very last moment, the Ego Car reaction time is lesser than scenario 1. We can notice that AEB pre-brake activation period is considerably less than the Scenario 1 because the deceleration provided by pre-brake is not sufficient to stop the vehicle moving at $80km/h$. When TTC went below 1 second, Ego car activates emergency brake with deceleration value of -9.6m/s^2 to reduce the impact of the collision. This can be also verified by the steep reduction of Ego Car velocity in the velocity sub-plot of the figure 4.11. The prediction time instances figure 4.12 shows that foreseeing the collision before 10 seconds perform better than the others. The acceleration trend of other time instances indicates steep reduction in the acceleration while the 10-second prediction instance shows a gradual reduction in the acceleration.

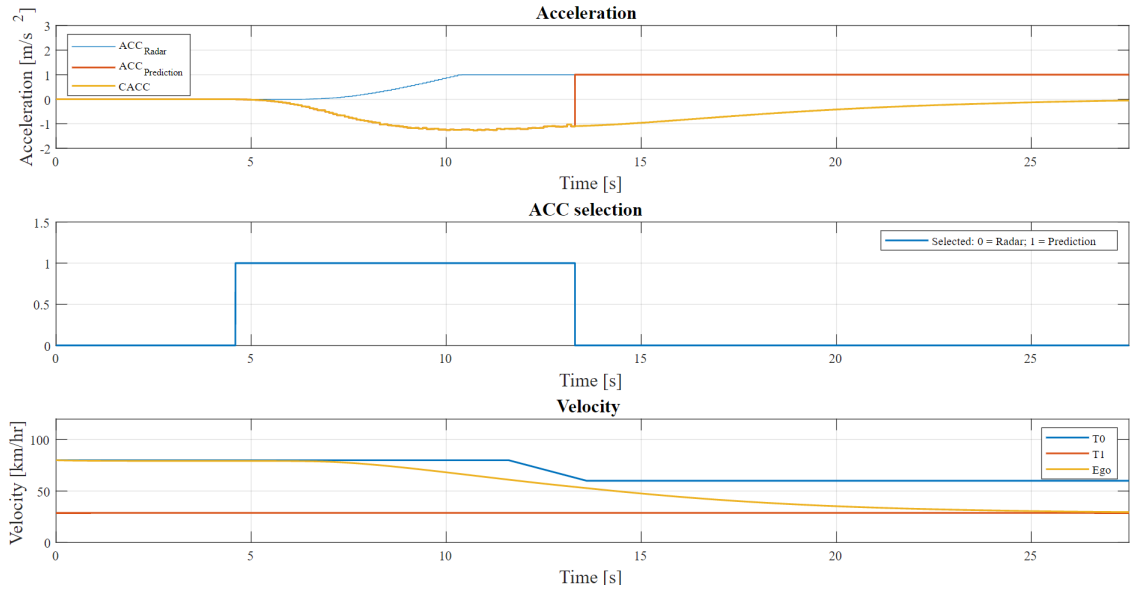

Figure 4.10. Scenario2: Ego Car Acceleration and Velocity variation based on RADAR and CAM Messages

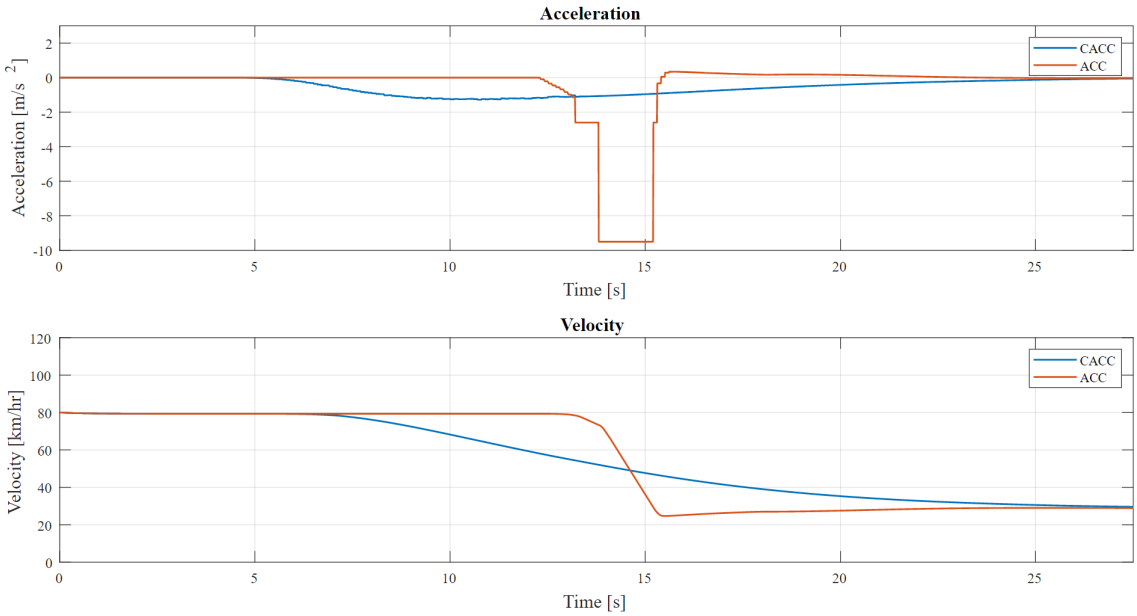

Figure 4.11. Scenario2: ACC vs CACC Comparision

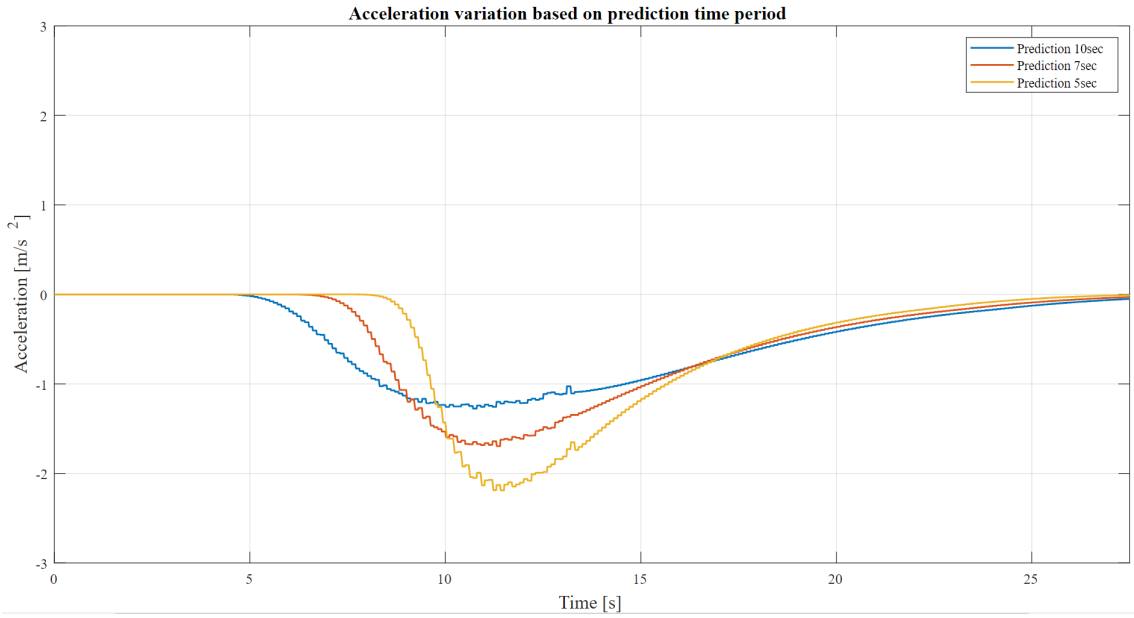

Figure 4.12. Scenario2: Acceleration Variation for different Prediction Time Instances

4.2.3 Scenario 3

This scenario aims to identify the stationary vehicle in the corner of a road with the help of CAM messages and apply the control algorithm to stop the Ego Car.

In this scenario, the Ego Car cruise speed is set at $60km/h$ while the lead car is travelling at $45km/h$. As we can see from the acceleration sub-plot of the figure 4.13, Ego Car acceleration didn't change with the reduction of the Lead Car Velocity(T0) as shown in velocity sub-plot of 4.13. This is because the relative speed between them is very low and the relative distance between them is very high. With the variable proportional gains trend shown in the figure 3.19, the control algorithm chooses the best gain to gradually reduce the acceleration. From 4.14 figure, we can see a lot of variations in ACC acceleration compared to other scenarios because Ego car is reducing its velocity to take a turn around the corner. Once the Ego Car turned in the corner, RADAR identifies the presence of a stationary vehicle and activate AEB with full force to avoid the collision. While the velocity plot of CACC shows gradual reduction in velocity as expected. The prediction time instance figure 4.15 shows the acceleration of Ego Car varies alot before getting stable. We can see that the Ego Car performs better with 10 seconds as the prediction time while other time instances activates AEB pre-brake to avoid the collision.This indicates prediction time frame of 7 or 5 seconds is not sufficient to maintain the TTC above 2.4 seconds, which concerns the safety of the passenger.

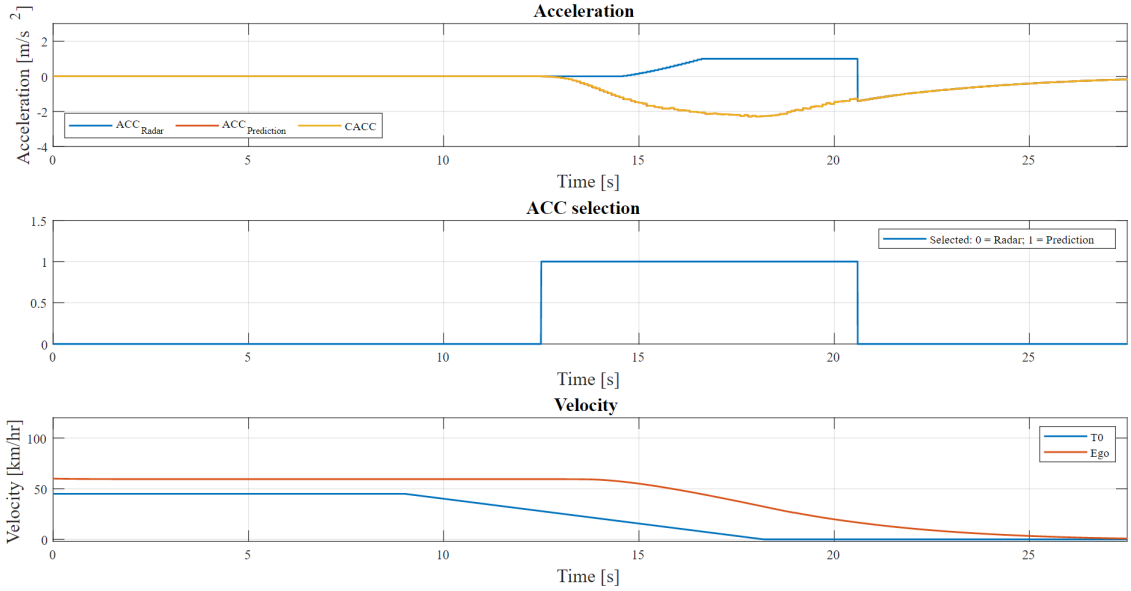

Figure 4.13. Scenario3: Ego Car Acceleration and Velocity variation based on RADAR and CAM Messages

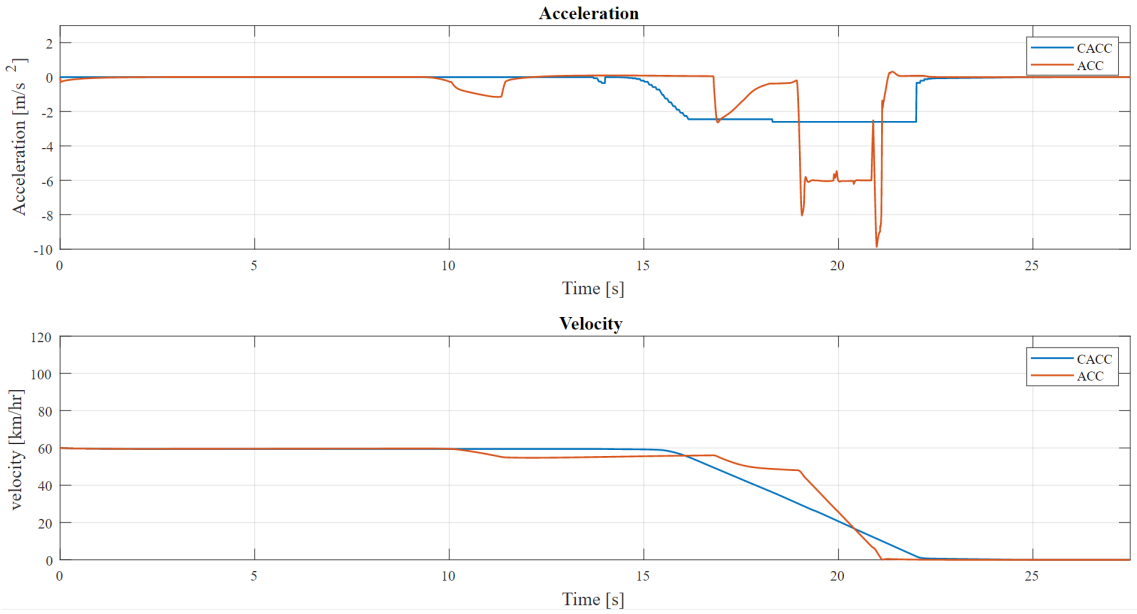

Figure 4.14. Scenario3: ACC vs CACC Comparision

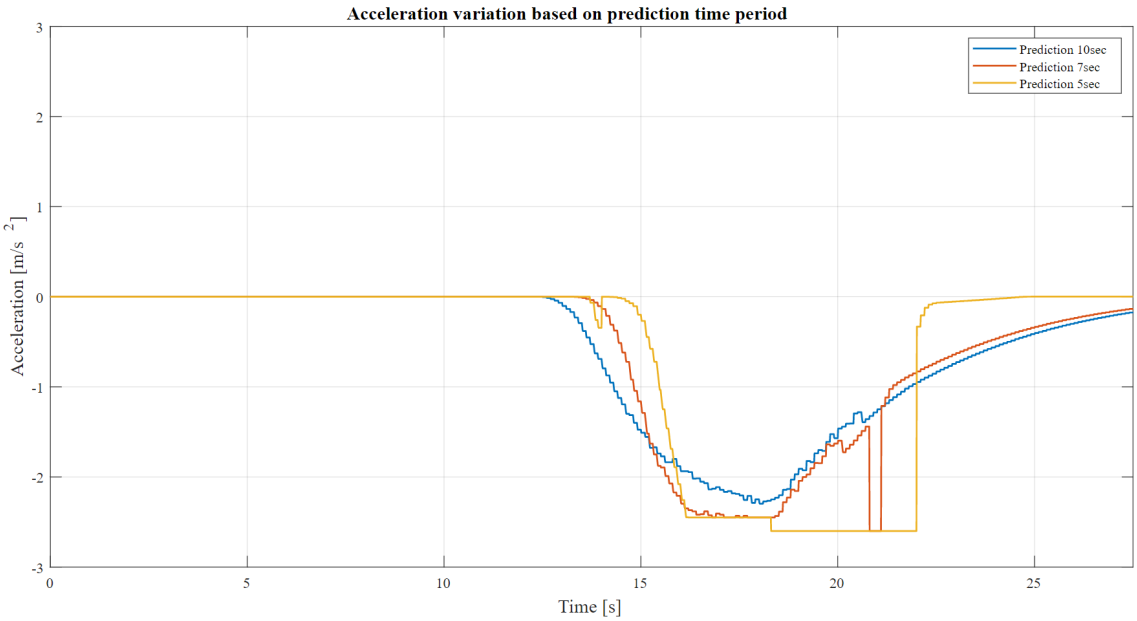

Figure 4.15. Scenario3: Acceleration Variation for different Prediction Time Instances

4.2.4 General Discussion

Scenarios	Quantity	No Prediction (without OBU)	Prediction (with OBU)		
			10 seconds	7 seconds	5 seconds
Scenario 1	Minimum Velocity (km/h)	27.7593	39.5807	40.4	38.6741
	Minimum Acceleration (m/s2)	-9.5	-1.7953	-1.5969	-2.0411
	Acceleration Standard Deviation	1.6733	0.7319	0.7078	0.7858
	Vehicle Detection Distance (m)	30.2753	203.5856	184.929	130.322
Scenario 2	Minimum Velocity (km/h)	24.69	29.0229	28.9579	28.8009
	Minimum Acceleration (m/s2)	-9.5	-1.2755	-1.6956	-2.1896
	Acceleration Standard Deviation	1.9694	0.4452	0.5465	0.55
	Vehicle Detection Distance (m)	20.1614	142.341	122.9626	99.9416
Scenario 3	Minimum Velocity (km/h)	0	0	0	0
	Minimum Acceleration (m/s2)	-9.6	-2.2965	-2.6	-2.6
	Acceleration Standard Deviation	1.636	0.7443	0.8399	1.0187
	Vehicle Detection Distance (m)	15.275	88.555	81.9081	70.6466

Table 4.1. Comparison of basic quantities between with and without OBU modes

Ultimately, we expected the system to avoid all potential collisions that might occur in the future with the help of CAM messages. We have also hoped for gradual acceleration/deceleration of the vehicle to provide a safe and comfortable journey. These expectations are fulfilled by the fact that in the case of OBU (connected vehicles) mode,

we were able to predict the collision in the future and implemented necessary actions to avoid the same. From the table 4.1, we can also notice that Ego Car didn't activate the AEB mode in all three scenarios to avoid the collision while in No Prediction mode, the Ego Car activates AEB with full force for all the three scenarios to avoid/reduce the impact of the collision. The minimum acceleration chosen by Ego Car in the default mode is -2.2965 for scenario3, where braking in the corner also played a major role in the deceleration. The main factor of the ACC/CACC algorithm is the detection distance and velocity, which directly relates to the reaction time of the vehicle. We can see from the table 4.1 that the difference in vehicle detection distance between No Prediction and default Prediction mode is very high. The minimum detection distance for prediction mode is 88.55m whereas for no prediction mode 15.275m. This directly reflects in the acceleration behaviour of the Ego Car. The mean standard deviation of predicted mode is 0.6404 while for no prediction is 1.7595 which shows the major variation in the Ego Car acceleration for no prediction mode. This rapid deceleration behaviour of the vehicle questions the comfort and safety of the passengers. Based on the discussions in the previous section 4.2, we can say that the CACC system can identify the suitable gains to ensure the gradual acceleration/ deceleration of the vehicle. On the other hand, if we compare the different prediction time instances, two out of three scenarios show predicting the collision before 10 seconds helps us to reach our expectations. But in the Scenario 1, prediction time frame of 7 seconds gave us better results compared to others. This can be verified from velocity, acceleration, acceleration standard deviation quantities mentioned in the table 4.1. However, in Scenario 3 it forces the vehicle to activate AEB to avoid the collision. We are developing a vehicle safety-related application where the prediction of the future time instance means finding a potentially critical situation and react to it beforehand. With less forecasting period,we are risking the safety of the passengers by providing less reaction time to the vehicle . From the table 4.1 and plots presented in previous section 4.2, we can safely conclude that CACC with the prediction time frame of 10 seconds is the most suitable parameter selection for the presented scenarios.

Chapter 5

Conclusion

In this thesis, we have created a co-simulation framework that focuses on integrating the functionalities of multiple simulators such as CarMaker, ns3, Simulink to study the impact of connected vehicles in safety-related applications. We have used CarMaker to analyse the dynamics of vehicles, emulate the on-board sensors activities and to create mobility scenarios. ns3 simulator is used to simulate the behaviour of vehicular communications. We have adopted the LTE based V2I system as our communication model where we have exploited ns3 LENA implementation to model the LTE framework. To combine the strengths of these two simulators, we have used Simulink and Python Engine as our interface. Simulink along with MATLAB is used to exchange information between Python Engine and CarMaker whereas ns3 Python bindings are used to perform the ns3 simulation in Python environment. The Python Engine controls the execution of all the simulators. Once a framework is established to combine these simulators, we have developed a control strategy to evaluate the performance of our system. The control strategy uses CAM message that contains the information of all vehicles in the network to predict any possible collision incident in the future. We have applied a trajectory-based collision detection system to calculate the position of the vehicles in various prediction time frames (10,7, 5 seconds). With the foreseen vehicle positions, we were able to identify the vehicles that are in the collision course with the Ego Car. To avoid the collision, the CACC algorithm is used to calculate the desired acceleration of the Ego Car. Through the Simulink interface, we have communicated this desired acceleration data to the CarMaker environment. The control algorithm also considers the Ego Car RADAR information to calculate the desired acceleration. After the creation of the control strategy, we have performed multiple experiments with various traffic scenarios where the mobility pattern of the Lead Cars are defined in a way to test the effect of vehicular communications in safety-related applications. The tested scenarios are Ego and

Lead Car crossing in a T-junction, Lead Car which cut-out at a high speed exposing Ego Car to a slow-moving vehicle, the stationary Lead car at the corner. The importance of the connected vehicles in traffic efficiency and safety is magnified by looking at the results section. The Ego Car which is equipped with OBU was able to predict the collision and avoid them with gradual acceleration/deceleration whereas Ego Car that depends only on-board sensors for avoiding collision ended up activating AEB to avoid/reduce the impact of the collision.

The main achievements of this thesis work are: developing a co-simulation framework that can combine multiple simulators without compromising their functionalities; developing a control strategy that can exploit both ACC and CACC controllers to ensure the passenger comfort and safety; analysing the importance of the connected vehicles in vehicular safety applications.

In the future, we will do extensive experiments that include multiple scenarios with complex traffic conditions where lateral movements are also performed to avoid a collision. The flexibility of the framework allows us to replace the CarMaker module with Simulink and Simulation of Urban Mobility (SUMO). Simulink can be used to model the vehicle dynamics, onboard sensors while the SUMO can represent the vehicle traffic. By using Simulink and SUMO, we can control all the cars involved in the simulation which is not possible in CarMaker. We will exploit the hybrid communication models where V2V with WAVE standard can be used to support platooning related applications while V2I with LTE standard can be used to forecast the collisions that might occur in the future.

Bibliography

[1] United Nations Population Fund, https://www.unfpa.org/sites/default/files/pub-pdf/695_filename_sowp2007_eng.pdf.

[2] Association for Safe International Road Travel, https://www.asirt.org/safe-travel/road-safety-facts

[3] The European Transport Safety Council, https://etsc.eu/euroadsafetydata/

[4] TomTom Traffic Index, https://www.tomtom.com/en_gb/traffic-index/

[5] INRIX Traffic Scorecard, https://inrix.com/press-releases/2019-traffic-scorecard-uk

[6] Y. L. Tseng. LTE-Advanced Enhancement for Vehicular Communication. IEEE Wireless Communications, Dec 2015.

[7] Fabio Arena and Giovanni Pa (2019).An Overview of Vehicular Communications, Future Internet Journal

[8] G. Araniti, C. Campolo, M. Condoluci, A. Iera, A. Molinaro, LTE for Vehicular Networking: A Survey. IEEE Communications Magazine,May 2013

[9] J. Schlienz and A. Roessler. Device to device communication in lte. Technical Report, ROHDE and SCHWARZ, 2016

[10] 3rd Generation Partnership Project. 3GPP technical specification group radio access network study on lte-based v2x services (release 14). Technical Report TR 36.885 V14.0.0, 3GPP, 2016.

[11] Alexandre K. Ligo, Jon M. Peha, João Barros,Throughput and Cost-Effectiveness of Vehicular Mesh Networks for Internet Access, September 2016

[12] Daniel Jiang, Luca Delgrossi,IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments, May 2008

[13] Mila Romana Cécile Tabacoff, Cellular and DSRC approaches for vehicular traffic safety, Politecnico Di Torino Thesis https://webthesis.biblio.polito.it/8233/, 2018

[14] José Santaa, Fernando Pereñígueza, Antonio Moragóna, Antonio F. Skarmeta, Experimental Evaluation of CAM and DENM Messaging Services in Vehicular Communications. Transportation Research Part C Emerging Technologies, September 2014

[15] Ondrej Burkacky, Johannes Deichmann, Georg Doll, and Christian Knochenhauer (2018).*"Rethinking car software and electronics architecture"*. mckinsey.com article

[16] Alice Matthews(2017) *"What is driving the automotive LiDAR and RADAR market?"*.electronicspecifier.com article

[17] National Highway Traffic Safety Administration, Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey, March 2018

[18] Anders Cioran (2015). "System Integration Testing of Advanced Driver Assistance Systems". KTH royal institute of technology school of electrical engineering, Sweden.

[19] Intelligent transport systems — Cooperative adaptive cruise control systems (CACC) — Performance requirements and test procedures,ISO 20035:2019. `https://www.iso.org/obp/ui/#iso:std:iso:20035:ed-1:v1:en`

[20] Dmitry Kachan (2010) Integration of ns3 with MATLAB\Simulink.Lulea University of Technology,Sweden

[21] Jakob Kaths,Sabine Krause (2017) Integrated simulation of microscopic traffic flow and vehicle dynamics.Conference:IPG Apply and InnovateAt,Karlsruhe

[22] Chenxi Lei, Emiel Martijn van Eenennaam, Wouter Klein Wolterink, Jeroen Ploeg, Georgios Karagiannis and Geert Heijenk. Evaluation of CACC string stability using SUMO, Simulink, and OMNeT++. EURASIP Journal on Wireless Communications and Networking 2012, 2012:116

[23] Amr Ibrahim, Chetan Belagal Math, Dip Goswami, Twan Basten, Hong Li. Co-simulation Framework for Control, Communication and Traffic for Vehicle Platoons.21st Euromicro Conference on Digital System Design (DSD),2018 .

[24] Apratim Choudhury, Tomasz Maszczyk, Muhammad Tayyab Asif, Nikola Mitrovic,Chetan B. Math, Hong Li and Justin Dauwels (2016).An integrated V2X simulator with applications in vehicle platooning.IEEE 19th International Conference on Intelligent Transportation Systems.

[25] Network Simulator, ns3 `https://www.nsnam.org`

[26] Network Simulator,ns3 Python bindings. `https://www.nsnam.org/wiki/Python_bindings`

[27] Alberto Arcidiacono, ADAS virtual validation: ACC and AEB case study with IPG CarMaker, Politecnico Di Torino Thesis `https://webthesis.biblio.polito.it/8276`, 2018

[28] ACI (Italian Automobile Club) (1992). "Art. 149. of New road codex - safety distance between vehicles. Decree of the President of the Republic 16 December 1992,n. 495.

[29] IPG CarMaker (2019). "Programmer's Guide Version 8.0". Bannwaldallee 60,76185 Karlsruhe, ipg-automotive.com

[30] IPG CarMaker (2019). "Reference Manual – Version 8.0". Bannwaldallee 60,76185 Karlsruhe, ipg-automotive.com.