CHALMERS
UNIVERSITY OF TECHNOLOGY

Virtual Integration and Simulation of Autonomous Systems

Master of Science Thesis Report

Ashok Krishna
Automotive Engineering

Palaniappa Sambanthan
Systems, Control and Mechatronics

Department of Mechanics and Maritime Sciences
Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2019

Master's thesis 2019

Virtual Integration and Simulation of Autonomous Systems

Ashok Krishna
Palaniappa Sambanthan

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanical and Maritime Sciences
Vehicle Engineering and Autonomous Systems
Vehicle Dynamics Group
and
Department of Electrical Engineering
Mechatronics Research Group
Chalmers University of Technology
Göteborg, Sweden 2019

Virtual Integration and Simulation of Autonomous Systems
Ashok Krishna
Palaniappa Sambanthan

Examiner: Bengt JH Jacobson, Department of Mechanical and Maritime Sciences, Vehicle Engineering and Autonomous Systems, Vehicle Dynamics Group, Chalmers University of Technology.
Examiner: Martin Fabian, Department of Electrical Engineering, Automation Research Group, Chalmers Universityof Technology
Supervisor: Henrik Lönn, Volvo Group Trucks Technology AB, Embedded software, Göteborg, Sweden
Supervisor: Oscar Ljungkrantz, Volvo Group Trucks Technology AB, Vehicle Automation M1, Göteborg, Sweden

Master's Thesis 2019:03
Department of Mechanical and Maritime Sciences
Vehicle Engineering and Autonomous Systems
Vehicle Dynamics Group
Department of Electrical Engineering
Mechatronics Research Group
Chalmers University of Technology
SE-412 96 Göteborg
Telephone +46 31 772 1000

Cover: Simulation of Software and mechatronic systems requires models of relevant aspects

Chalmers Reproservice
Göteborg, Sweden 2019

Virtual Integration and Simulation of Autonomous systems
Ashok Krishna
Master's thesis in Automotive Engineering
Department of Mechanical and Maritime Sciences
Vehicle Engineering and Autonomous systems
Vehicle Dynamics Group
Palaniappa Sambanthan
Master's thesis in Systems, Control and Mechatronics
Department of Electrical Engineering
Mechatronics Research Group,
Chalmers University of Technology

Abstract

Virtual integration and simulation is an important approach and often the most cost effective solution in understanding and verifying systems, as it aids in analyzing a system that is comprised of different software/environments. Early verification and validation through virtual integration helps reduce the lead times in product development.The main aim of this thesis is to investigate the possible approaches to integrate Matlab/Simulink and ROS (in Ubuntu 16.04v in VirtualBox) with the Volvo Trucks' in-house simulator tool ADAPT, thereby creating a hybrid simulation environment for the functionality domains of Vehicle Motion Management (VMM), Motion Support Devices Management (MSDM), Traffic Situation Management (TSM) and the vehicle model. Several integration combinations involving the respective functionality domains in the above mentioned software have been tried in this thesis work. Volvo Transports Model (VTM) in Matlab/Simulink is used used by Volvo GTT for vehicle dynamics modeling and function development. Furthermore, as part of the integration, the existing vehicle dynamics modeling components in VTM, is swapped with a FMU of simpler OpenPBS model which has been modeled in Modelica format using the software Dymola. This task aids in providing a simpler, open source based vehicle model which has several advantages like faster simulation time, ease of model translation into different formats.
The other objectives include, identifying the limits and capabilities of these tool integrations and plant models. Finally, the simulation behavior, execution time, synchronization level for the integrations is compared.

Keywords: Virtual Integration, OpenPBS, Modelica, Matlab/Simulink, ADAPT, ROS, VirtualBox, Ubuntu.

Acknowledgement

This thesis was carried out at Volvo Group Trucks Technology, in the Embedded software development division at Lundby, Göteborg, Sweden. Firstly we would like to thank our supervisor Henrik Lönn at Volvo Group Trucks Technology for providing us an oppurtunity to carry out this thesis, supporting and giving us constant feedback on our work and also for extending his help in linux kernel development.

Secondly we would like to thank our supervisor Oscar Ljungkrantz at Volvo Group Trucks Technology, Automation Group for supporting and felicitating meetings with other development teams who are working with the functionality domains handled in the thesis work. These interactions were quite valuable in providing us with the needed information for our thesis. We would also like to thank Cui Gongpei, Kuang Fangjin at Volvo Group for helping us solve errors and problems faced during Matlab/Simulink interfacing and helping us in understanding the ADAPT platform works to create the simulation environment and giving us the models to test. We would like to thank Josè Vilca and Sachin Janardhanan at Volvo Group for providing us with their valuable insight on the control modules and the plant models.

A special thanks to Martin Fabian at Chalmers, Signals and Systems department, for helping us with the integration of ROS, giving us constant feedback on the python scripts for the synchronization and helping us with the Algebraic loop errors in Matlab. A special thanks to Bengt Jacobson at Chalmers, Vehicle Engineering and Autonomous Systems division for extending his guidance on the plant model and for providing his feedback on model segregation and Dymola software for development and testing purposes.

Palaniappa Sambanthan, Gothenburg, March 2019.
Ashok Krishna, Gothenburg, March 2019.

LIST OF ABBREVIATIONS

1D	One Dimension
ADAPT	Automated integration Data And Products status Tool
API	Application Programming Interface
CAN	Common Area Network
CoG	Center of Gravity
COTS	Commercial Over-the shelf
CPS	Cyber Physical Systems
DCP	Distributed Co-Simulation Protocol
DOF	Degrees of Freedom
FMU	Function Mock-up Unit
LIN	Local Interconnect Network
I/O	Input and Output
IP	Internet Protocol
IRQ	Interrupt Request
MSDM	Motion Support Devices Management
OEM	Original Equipment Manufacturer
PBS	Performance Based Standard
RAM	Random Access Memory
Sync. Manager	Synchronization Manager
TCP	Transmission Control Protocol
TSM	Traffic Situation Management
UDP	User Datagram Protocol
VB	Virtual Box
VeMFRA	Vehicle Motion Functionality Reference Architecture
Volvo GTT	Volvo Group Trucks Technology
VM	Virtual Machine
VMM	Vehicle Motion Management
VTM	Volvo Transports Model

LIST OF Symbols

f	front
r	rear
W	track width - m
L	wheel Base - m
l_f	Longitudinal distance from CoG to front axle - m
l_r	Longitudinal distance from CoG to rear axle - m
m	Mass - kg
J	Mass Moment of Inertia - kgm^2
x,y,z	Vehicle Positions - m
v_x, v_y, v_z	Vehicle velocity, in Inertial System - m/s
a_x, a_y, a_z	Vehicle Acceleration, in Inertial System - m/s2
F_{xv}, F_{yv}	Forces in Vehicle co-ordinate system - N
F_{xw}, F_{yw}	Forces in wheel co-ordinate system - N
C_f, C_r	Cornering Stiffness front and rear wheel - N/rad
S_y	Tyre lateral Slip
ωz	Yaw Angular Velocity - rad/s
δ_f	Steering Angle - rad
β	Vehicle Side Slip Angle - rad

Contents

1
Introduction

Recent advancements in autonomous interventions and controls in vehicles have paved way for further enhancing the level of automation and the development of autonomous vehicles. However, with such advancements, one needs to provide benchmarks and standardization requirements in order to uphold the reliability and usability of the autonomous technology. Failure rate or error rate could be one such statistical criterion wherein the benchmark of mileage without failure demonstrates that the vehicle is reliable. To demonstrate that fully autonomous vehicles or vehicles with higher automation levels have a fatality rate of 1.09 fatalities (at most) per 100 million miles (R=99.99..989%) with a C=95% confidence level, the vehicles would have to be driven 275 million failure-free miles. [10]
These estimates demonstrate that even under regressive testing assumptions, this would be an impossible proposition, if the aim was to demonstrate the vehicle performance prior to be certified for commercial road use. Citing many such studies, Nidhi et al [11] confirms and quantifies that there is the need for alternative methods such as accelerated testing, virtual testing and simulations, mathematical modeling and analysis, scenario and behavior testing as well as extensive focused testing of hardware and software systems to supplement real-world testing in order to assess autonomous vehicle safety and shape appropriate policies and regulations.
Model based design and simulation has become a norm in automotive product development. With the availability of various COTS (Commercial off the Shelf) simulators, modeling and execution environments, such as IPG CarMaker, Dymola, PreScan, ROS, etc. the early verification and validation of automotive systems has been made possible. These software have their own limitations and capabilities of analysing the designed system.
Functional testing or complete vehicle testing requires, several models such as software models, sensor models, vehicle models and external environment models which needs to be provided by the developer and supported by the simulator. Thus there could be instances where the initial modeling of the system is done in one platform, whereas the simulation may have to be carried out on a different platform. There could be situations where, several different platforms have been used to create sub models, but there is a different execution environment. Furthermore, in the above cases the simulation may be handled by software, but when it comes to Hardware in Loop testing, there arises a need for virtual integration and/or co–simulation.

Robot Operating system (ROS) is a commonly used platform by various OEMs to analyse and assess large complex systems which include hardware abstraction and low-level device control. ROS has been forecasted to be employed extensively as a platform to simulate functionalities of vehicle motion control and in real-time applications.

1.1 Background

1.1.1 Functional Architecture View

According to the ISO26262 functional safety standard [12], a functional concept is defined as, "specification of the intended functions and their interactions necessary to achieve the desired behaviour". Thus the functional architecture then refers to logical decomposition of the system into components and sub-components, as well as the data-flows between them. Sagar Behere et al [13] define the typical functional architecture decomposition for autonomous vehicles to be typically comprising of three main categories- Perception, Decision and Control, Vehicle Platform Manipulation.

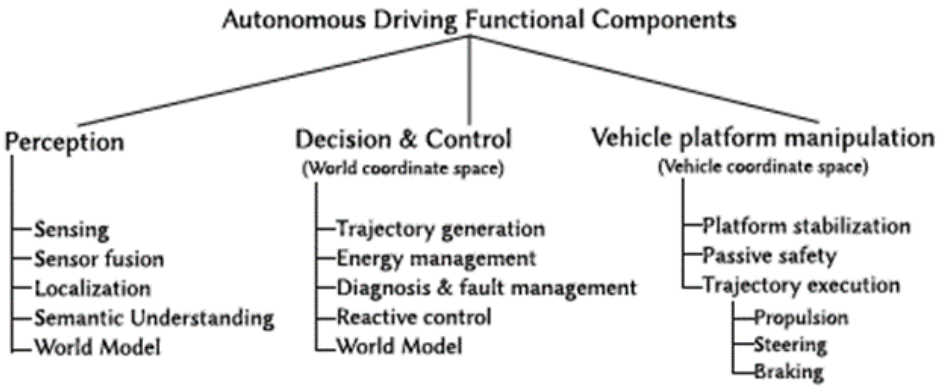

Figure 1.1: *Typical functional components of an autonomous driving system.[13]*

VeMFRA - Vehicle Motion Functionality Reference Architecture
Similarly, Volvo GTT's vehicle motion functionality reference architecture (VeMFRA) defines a reference pattern for decomposition and structuring of functionality with further sub-level definition. One of the cornerstones of VeMFRA is the layers used to organize sets of functionalities. The layer order and functions included in each layer are organized based on their temporal and spatial extension. Upper layers (longer extension) are dependent on the existence of lower layers. Lower layers provide aggregated capabilities to the next upper layer.

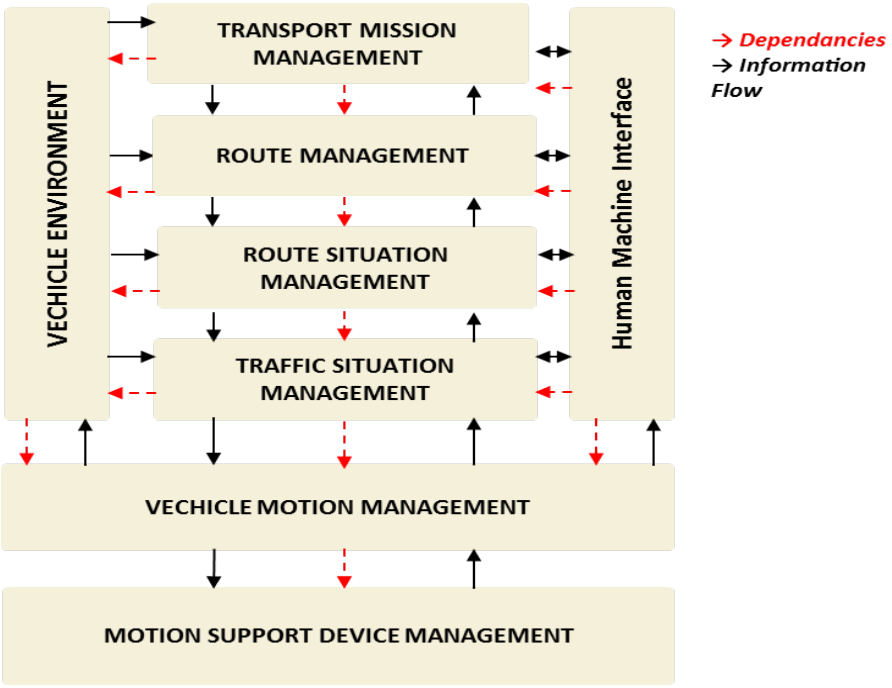

Figure 1.2: *Vehicle Motion Functionality Reference architecture(VeMFRA) [19]*

VeMFRA defines generic elements for hierarchical structuring. The main functionality domains are:

- Transport Mission Management (TMM)
  - Functions in this layer can provide weekly or daily plans for each vehicle managed by a fleet owner
  - Includes functions related to fleet management and route assignment that could be attributed to the purpose of providing information with respect to energy consumption & cost
- Route Management (RM)
  - Handles information on the vehicle performances for an actual designated route
  - Typically the range for this functionality domain is within the spatial horizon of 100km
- Route Situation Management (RSiM)
  - Includes functionality related to road segments, typically 15km, of a planned route.
- **Traffic Situation Management (TSM)**
  - Functionality domain related to functions with a temporal horizon of up to 10 s.
  - The main attributes include: continuous control and decision-making with respect to the subject vehicle behavior in an observed traffic situation, on comparing with the driving task for a human driver
- **Vehicle Motion Management (VMM)**
  - Functionality domain encapsulating knowledge of available actuation and coordinates the use of the actuators within the vehicle
  - Typically, these functions are related to a temporal horizon of up to 1 s

- **Motion Support Device Management (MSDM)**
  - Handles the organization of monitoring, measurement and control of devices,
  - Typically consists of actuators and sensors
  - Typically, these functions are related to a temporal horizon of up to 1 s
- Vehicle Environment Management (VEM)
  - Functionality domain comprising of defining and representing the vehicle surroundings, topology, maps etc.
- Human machine Interface (HMI)
  - Includes detection and response to drivers' intentions

To further understand the architecture view, the function domains of RM & RSiM can be associated with the strategic level of driving; TSM can be associated with tactical maneuvering including aspects as gap acceptance and overtaking; VMM can be associated with control. The operational level is predominantly associated with the motion support devices such as, the engine, gearbox torque management, wheels, brakes etc.

1.1.2 In-House Simulator

COTS are sometimes complex, expensive, inflexible and may have the need for continuous tailoring. To overcome these, a specific simulator platform - ADAPT was developed in context with Volvo Group. ADAPT was developed under the Heavy Road FFI project together with modelling and configuration tooling on the ArEATOP environment. The ADAPT integration Environment is a framework for continuous integration and Delivery and is used for integration and verification in Embedded software department. The simulator platform is represented by figure [1.3]

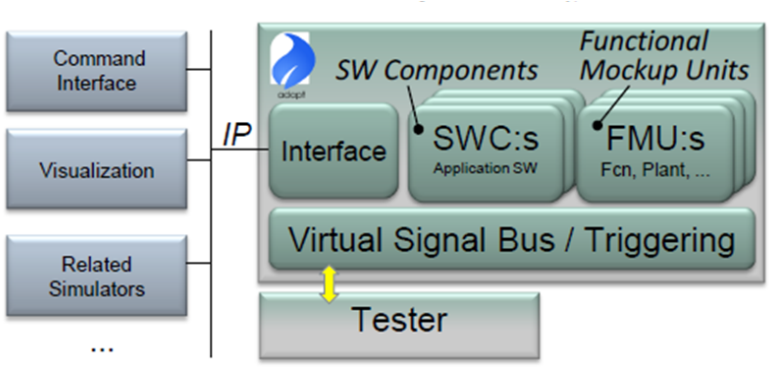

Figure 1.3: *ADAPT Simulation Platform [16]*

Adapt modules represent software components, physical simulation models, logging, along with interfaces to physical buses and I/O. Adapt modules for software components are primarily based on the Autosar standard. A module generator provides the wrapper code that interacts with the software component and with the Adapt simulation core. Adapt modules for Functional Mockup Units are based on binaries and interface descriptions according to the Functional Mockup Interface standard.

An FMU module generator provides wrapper code that interacts with the Functional Mockup Unit's binary according to its specification in the description file. The wrapper code likewise interacts with the Adapt simulation core. The interface module can exchange data with external entities. The communication module reads and writes from CAN and LIN bus as well as handle IP communication like UDP or TCP.

1.1.3 Co-Simulation

Co-Simulation in general includes both co-simulating different parts of the system connected by continuous variable(s) or co-simulating different environments based on timing execution. Several studies have been carried out on Co-simulation and Hybrid simulations such as *Shota et all (2017)* [1] propose an integrated framework to test and simulate autonomous driving system in ROS along with Matlab/Simulink, by using *Runtime manager* tool of Autoware (software stack framework maintined by *Tier IV* firm). Another example of co-simulation is presented by *Sinsa slavnic et all (2013)*[3], where human and robot walking dynamics are modelled using MSC ADAMS and Robot operating systems. There are various other integration done, such as ROS and GAZEBO and several others, but the software involved in all of these integration run on the same platform either on Windows or on Linux. In some applications the software is run as a Functional Mock-Up units *FMU* on the other software. *Fabio cremona* [2] explains why time is an important factor in an hybrid simulation environment, as the hybrid simulators have to interact in predictable and controllable manner and the work also illustrates how different timing model like, Newtonian physical model of time and Cyber approximation affect the Cyber Physical Systems (CPS). A CPS is a combination of computational, networking and physical systems working together.
In this thesis a hybrid simulation environment is created to integrate three different simulation environments, Matlab/Simulink, ROS and ADAPT, to simulate and study different autonomous functions.

1.2 Purpose and Objective

This thesis work has been carried out for the functional architecture based on VeM-FRA with the functionality domains TSM, VMM & MSDM, Plant Model (VTM and OpenPBS FMU later on). As many simulation software are supported by certain application platform only, for example, ROS is supported by Linux, Ubuntu platform and ADAPT is supported by windows platform.
Thus, the primary objective is to achieve virtual integration by creating a synchronization manager which takes care of synchronization based on timing execution between different environments. Also, to study the capabilities of the hybrid simulation environment, i.e., by performing the co-simulation of the functionality domains TSM in ROS/ADAPT/Simulink, VMM&MSDM in ADAPT/Simulink and Plant model in, Simulink or as a FMU. Figure 1.4 illustrates the functionality domains and the respective software platforms for which the integration is carried out for. The integration combinations are presented in detail in section 3.3.2.

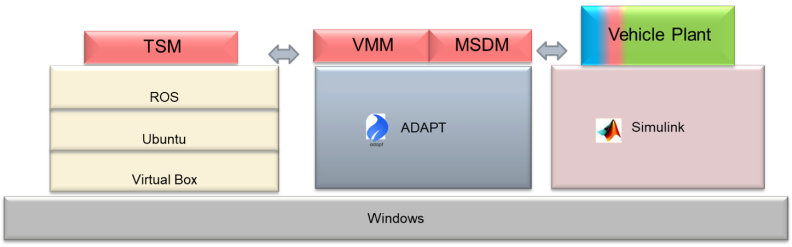

Figure 1.4: *Integration Platform*

1.3 Specifications

The software in study are Matlab/Simulink, ADAPT and ROS. The ROS software is run on Linux on VirtualBox environment on top of Windows platform. The Matlab/simulink has the vehicle model (VTM 8x4 Rigid Truck Model or FMU From Dymola/Open Modelica) for which the autonomous system(Heavy vehicles with automated driving) has to be tested, the ADAPT has VMM, MSDM, whereas ROS runs TSM.

1.4 Limitations

1.4.1 Limitations with Model

Each test scenario presents the requirement for the level of model abstraction i.e., the fidelity of the plant model. Model fidelity simultaneously dictates certain requirements and constraints on the integration while also affecting the synchronization. Thus it is often hard to have a universal simulation environment capable of handling all models and providing the desired simulation capabilities. Certain trade-offs always have to be made either in terms of the time to simulation or model complexity.

1.4.2 Limitations with Experiment Platform

As the Oracle VirtualBox runs on top of the Windows platform, so any operating system executed on a VirtualBox will not be similar to a standalone system. The clock of the virtual Linux system will always be synchronized to the windows system. With the proposed synchronization method the granularity of the system is 1 Second for the Co-Simulation of Matlab/Simulink along with ADAPT and ROS. Whereas for if the Co-Simulation involves only Matlab/Simulink and ADAPT the granularity of the system is 10ms. Since VirtualBox is used, there is limited controllability on the Linux sytsem.

1.4.3 Limitation with Virtual Simulations

Virtual testing and simulations do present the possibility to statistically check for the reliability of the autonomous driving technology. However, they still may be not be exhaustive and quite often not be enough to cover all possible scenarios. Hence,

virtual integration and simulation approaches alone may not be sufficient to address the challenge posed to policymakers, OEMs and developers to put autonomous technology to public use.

1.5 Method

1.5.1 Model Integration

The method followed in the thesis is as follows:

- Existing Setup Study
  - The existing plant Model VTM is studied for its architecture, and the various input and output signals and their respective sample rates need to be studied in order that the interface is maintained when replaced by a different plant model.
- Simulink Integration
  - The plant model VTM is based entirely on Simulink platform. Hence, having VMM, MSDM and TSM in Simulink will serve as a reference for rest of the integrations.
- Replacing the plant model with an open source vehicle dynamics model
  - Split VTM into plant components and actuators
    - The existing VTM plant model is segregated into vehicle plant(wheels and body) and actuators thereby allowing for integrating with the FMU of the single track model.
- Create single track vehicle model
  - In order that the open source model created in Dymola is capable of being integrated with the existing motion devices, certain outputs viz. longitudinal accelerations, velocities, yaw rates, need to be generated and converted suitably.
- Integrate generated FMU nodes with VMM & MSDM

1.5.2 Synchronization Manager

In-order to have a good virtual integration, the run time of each software has to be controlled and synchronized respectively with each software involved. This is done by executing the software only for certain macro time step or one cycle at a time upon issuing a certain command and make the software wait until it receives the next command. The time step is decided according to the fidelity of the system and managed by an external tool. So the steps in achieving the virtual integration are as follows :

- Independent process control
  - Control Matlab/Simulink
  - Control ADAPT
  - Control ROS
- Synchronized process control
  - Create a synchronization manager - to control all the software (Matlab/Simulink + ADAPT + Robot operating system)

1.5.3 Testing

Test the integration with the virtually integrated software for the model integrations.

- Check for the timing granularity

The reference path from the TSM is as shown in the Figure 1.5. The integration setups are simulated for the path follower test case.

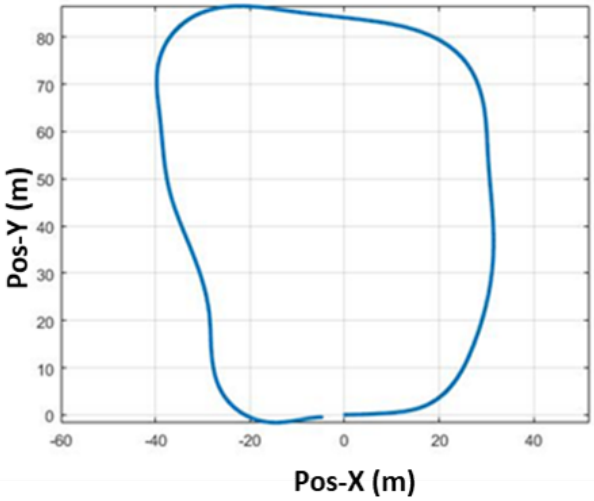

Figure 1.5: *Path follower test refernce path, X-Axis: Position along X, Y-Axis Position along Y*

1.6 Report Structure

This report is structured into different chapters as follows,

- Theory chapter
  - Describes about the vehicle dynamics model in VTM and the open source model in Modelica
  - Explains in detail about the problem and need of synchronization. Also explains in brief how to control each software.
- Method Chapter
  - Details about Plant model
  - Explains on how the Synchronization manager controls and manages the timing between all software.
  - Describes the various integrations carried out
- Results Chapter
  - The results and discussions are presented in this chapter.
- Conclusion Chapter
  - The future work and the conclusion of this thesis is presented in this chapter.
- Finally the whole script along with the other information are presented in the appendices chapter.

2
Theory

2.1 Model Description

2.1.1 VTM

Volvo Transport Model (VTM) is a MATLAB/Simulink based toolbox vehicle model library comprising of detailed longitudinal, lateral and vertical vehicle dynamic models of the complete rigid truck/tractor-trailer combination. The library toolbox covers major components like frame, steering system, suspensions, axles and wheels. VTM utilizes the Simscape Mechanics tool of Simulink. The vehicle model in VTM typically consists of the Vehicle bodies (Truck: Cab, Frame; Trailer; Dollies), Axles blocks, Tyre blocks.
The dynamics of the vehicles in VTM (Library) are modeled by two masses representing the chassis (Front, Rear) and by Pacejka tyre model (PAC2002). A vehicle in VTM is modelled as rigid bodies that interact with other by exchange of forces and moments at the points of connection. The tractor sprung mass consists of the cabin and the chassis frame. The torsional flexibility of the vehicle is modelled by splitting the cab and frame into two bodies and linking them via a 1D spring damper system in roll. The cabin and the front axles are connected to the front body; the rear axles and the coupling are connected to the rear body. The difference in respective roll angles provide the chassis frame torsion. The cabin is suspended on the chassis along the heave, roll and pitch DOF, and the rest of DOF are fixed. The wheels can only heave and roll with respect to the sprung mass and are rigidly included in the axles. The propulsion torque request from the VMM is sent to MSDM. MSDM

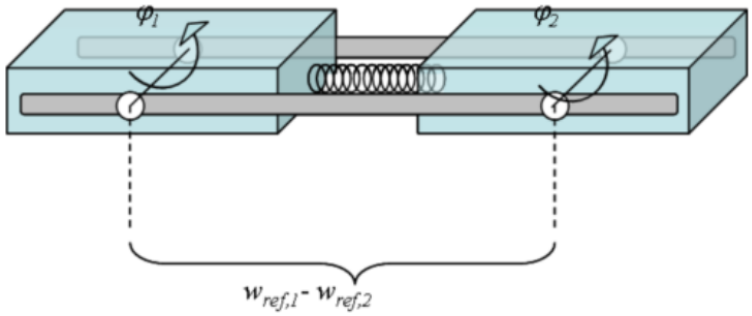

Figure 2.1: *VTM Frame Model [14]*

models the propulsion actuator using a first order transfer function, sends the actuated torque demand to the vehicle sub-plant as a drive torque request. VTM based

vehicle plant has the following inputs:

- Steering Angles (front axle and tag axle),
- Wheel torque

The following outputs are available from the VTM based vehicle plant,

- Cab/Frame position along x,y
- Roll, Pitch, Yaw Rates & Angles,
- Longitudinal acceleration & Velocities
- Lateral Accelerations & Velocities
- Wheel Speeds
- Wheel Forces along x,y,z

The truck model used in this thesis is 8x4 rigid truck with a steered tag axle from the VTM library. Figure 2.2 represents the schematic of the first sub-level of the plant system for a 8x4 Rigid Truck.

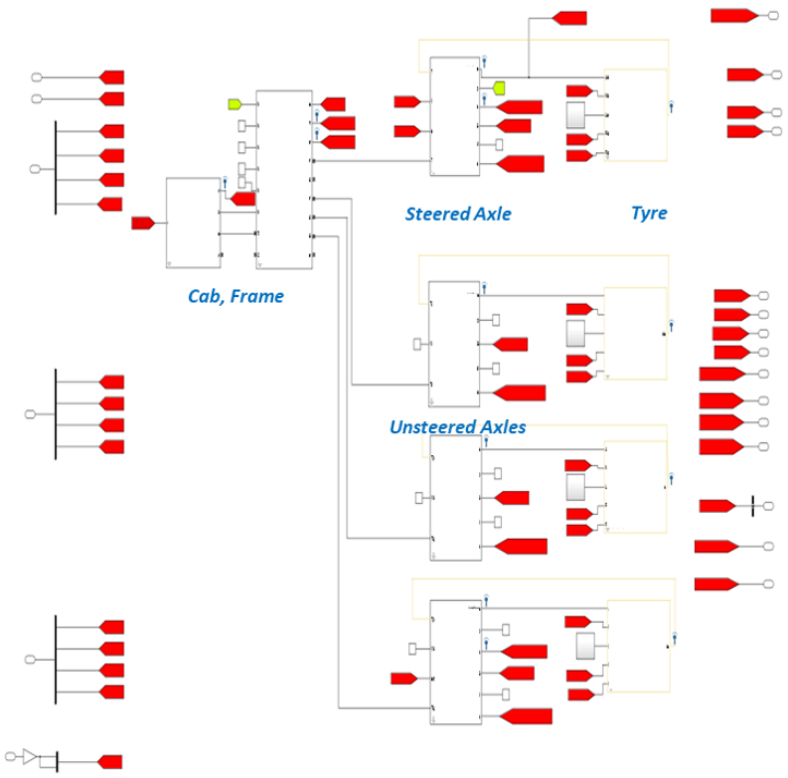

Figure 2.2: *Schematic of Vehicle Plant sub system for 8x4 rigid Truck*

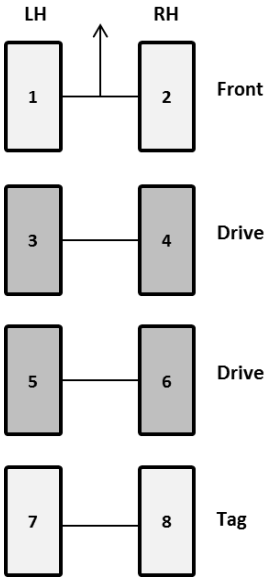

Figure 2.3: *Graphical representation of the truck, Gray wheels denote unsteered axle*

There are several bus signals that are directly fed through between the axle, tyre and steering blocks. Thus, there is an inherent algebraic loop in the system.

2.1.2 OpenPBS Model

The set of performance based standards or PBSes, computational methods for virtual verification and their implementation in a computer tool has been proposed in "Performance Based Standards for High Capacity Transports in Sweden", FFI project, Vinnova (Reference number: 2013-03881)[20].
Bengt et al[23] present a report, developing and describing the first version of the open assessment tool or "OpenPBS". The first version of the tool has published in github, refer [21]. The vehicle dynamics of the open source model is based on a one-track transient model for articulated vehicles with linear tyre equations and is illustrated in Figure 2.4. The model has been created in Dymola software, using the Modelica programming language. Modelica as a useful format for formulating simple vehicle models has been presented in [22].
The subscript w in the figure refers to the wheel coordinate system.

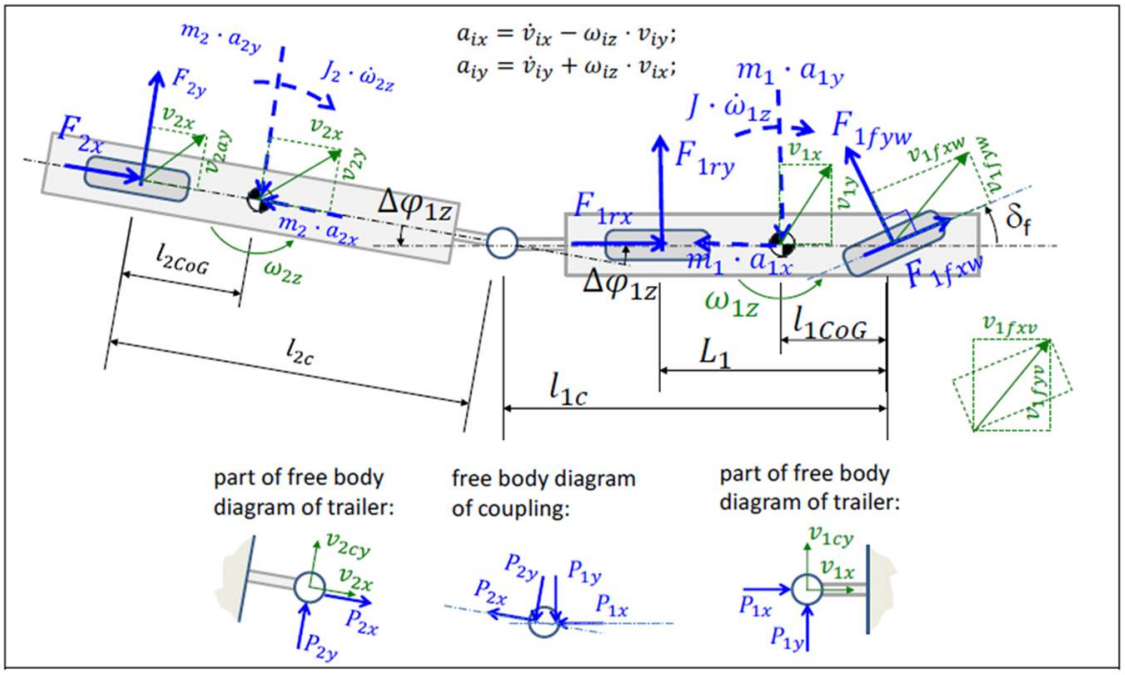

Figure 2.4: *Transient Model for articulated Vehicles [17]*

The single track model for a two axle rigid vehicle is described by the equations 2.1 to 2.4.

Equations of motion :

$$\begin{aligned} m \cdot (a_x - \omega_z \cdot V_y) &= F_{fxw} \cdot cos(\delta_f) + F_{fyw} \cdot sin(\delta_f) + F_{rx} \\ m \cdot (a_y - \omega_z \cdot V_x) &= F_{fxw} \cdot sin(\delta_f) + F_{fyw} \cdot cos(\delta_f) + F_{ry} \\ J \cdot \omega z &= ((F_{fxw} \cdot sin(\delta_f) + (F_{fyw} \cdot cos(\delta_f))l_f - F_{ry} \cdot l_r \end{aligned} \tag{2.1}$$

Constitutive relations :

$$\begin{aligned} F_{fyw} &= -C_f \cdot S_{fy} \\ F_{ry} &= -C_r \cdot S_{ry} \\ S_{fy} &= \frac{V_{fyw}}{V_{fxw}} \\ S_{ry} &= \frac{V_{ry}}{V_{rx}} \end{aligned} \tag{2.2}$$

Compatibility :

$$\begin{aligned} V_{fxv} &= V_x \\ V_{fyv} &= V_y + l_f \cdot \omega_z \\ V_{rx} &= v_x \\ V_{ry} &= V_y + l_r \cdot \omega_z \end{aligned} \tag{2.3}$$

Transformation between vehicle and wheel coordinate systems :

$$V_{fxw} = (V_y + l_f \cdot \omega_z) \cdot sin(\delta_f) + V_x \cdot cos(\delta_f)$$
$$V_{fyw} = (V_y + l_f \cdot \omega_z) \cdot cos(\delta_f) - V_x \cdot sin(\delta_f) \tag{2.4}$$

It is to be noted that the modeling becomes complex for heavy vehicles when more axles are involved and when additional units are connected. Additional terms for force interactions between the axles and coupling interactions are appended to the above equations.
The OpenPBS assessment tool contains vectorized models of articulated vehicles. The paramterization has been extended for creating a rigid 8x4 vehicle and is exported as an FMU. The vehicle sub-plant of VTM is swapped with this FMU and the same has been explained in detail in Section 3.1.

Volvo Groups Trucks uses many simulation software to test and verify the developed models, control systems, for different use cases. When it comes testing all of these systems together, it becomes difficult to run all platforms at the same time, as some platform run at "REAL-TIME", where as others might at different pace. For example, Matlab running on windows platform runs at different pace when compared to the windows' system clock but any application running on Linux platform runs at the real-time, Linux is a *Real Time Operating System - RTOS*, as the system clock and applications' clock run at the same pace. It becomes difficult to test when these two platforms are involved in Co-simulation, as some software are supported by certain platforms only. So it becomes important to create a tool to manage all these platform or the software to run together.

2.2 Simulation Concept

The Co-simulation framework involves three software, Matlab/Simulink, ADAPT and ROS, where Matlab/simulink and ADAPT is running on windows platform where as the ROS is running in VirtualBox on top of Windows platform. Matlab/Simulink has the vehicle model, the vehicle model used is 8x4 truck model. ADAPT has the control systems for the vehicle, which includes steering control module, Braking system module, autonomous vehicle control systems, etc. Where as ROS has the environment model. All these models have been developed by Volvo Group Trucks. In-order to test the behaviour autonomous vehicle in certain scenario, the truck model along with control system have to be tested in the environment, which requires all three software to be executed together.

2.2.1 Synchronization Rule

To have a perfect integrated solution, the software should obey the "synchronization rule". For instance, to simulate an autonomous truck in a traffic situation, The vehicle model in Matlab/Simulink, has to communicate with autonomous control module in ADAPT and steer in a traffic situation environment in ROS and vice-versa. When all these three software programs are executing together, each software has it's own pace to run, as shown in the Figure 2.5. It can been seen that before Matlab/Simulink finishes its first cycle, ADAPT has finished and has started the second cycle whereas ROS has started the third cycle. In general Matlab/simulink always takes a longer time when compared to ADAPT and ADAPT takes longer than ROS. As a result of which the desired result will not achieved due to asynchronous timings.

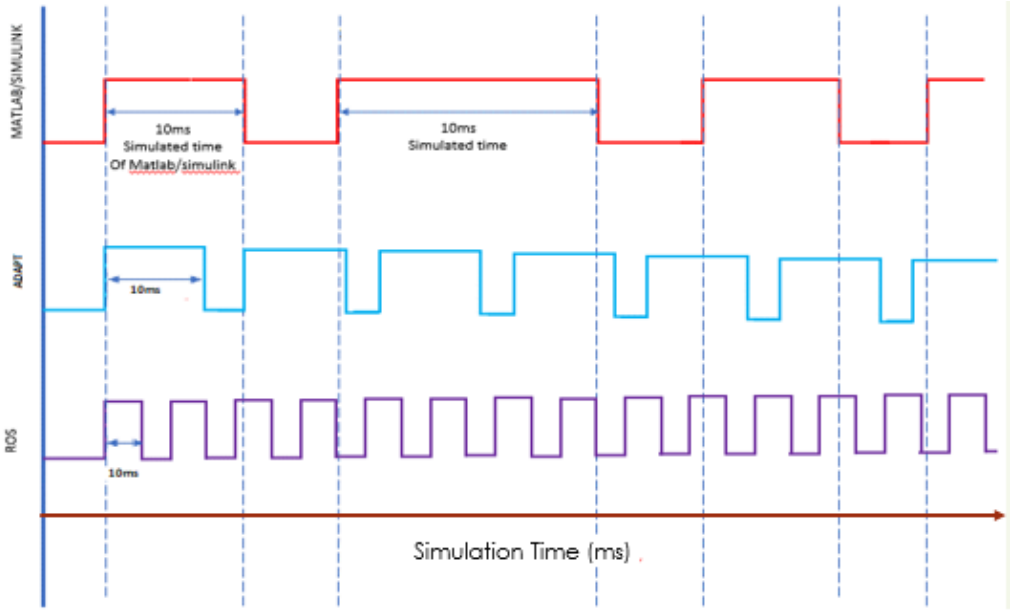

Figure 2.5: *Execution timing example of a simulation "without" synchronized simulation*

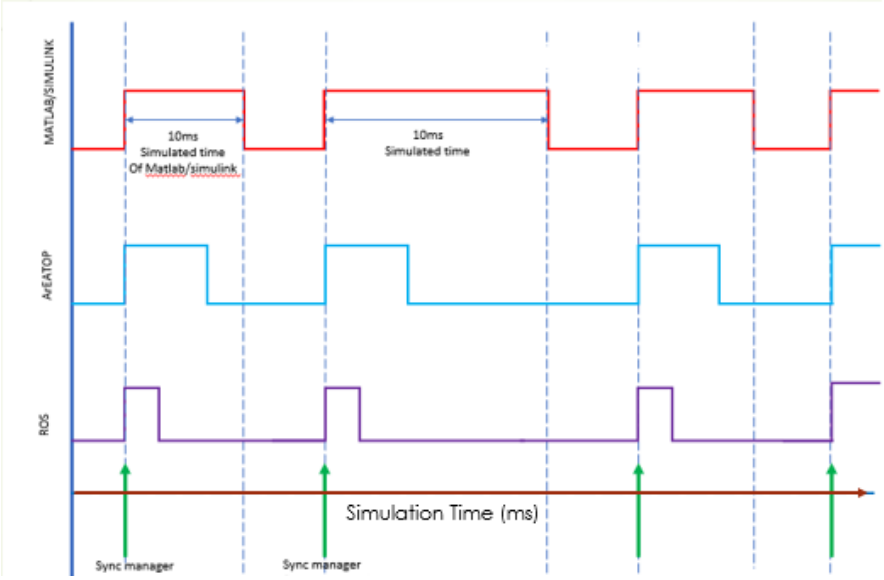

Figure 2.6: *Execution timing example of a simulation for a synchronized simulation*

Figure 2.6, shows that when a synchronization Manager (an external control component) issues a start command or any command to the software programs in the test setup, each software would run for one cycle (one simulated macro time step) and then goes into waiting state. Once all software programs are finished the first cycle simulation, the synchronization manager will send the next command for the next cycle to run. By this it can be ensured that all software programs are made to run at the same time.

2.2.2 Matlab/Simulink Simulation

Matlab is a numerical computing environment, Simulink is graphical programming environment used for modelling, simulating and analyzing dynamical systems. The Matlab/Simulink runs at a pace of 10ms "simulated time" steps, on using the fixed macro time step, *ODE solver*. It is simulated time because, Matlab/Simulink time steps is not synced with windows clock. Certain simulation would take a longer time, depending upon the fidelity of the model involved in simulation as it requires it high computational time for high fidelity model.
In order to obey the synchronization rule, Matlab/simulink has to run only for 10ms(one macro time step value) of simulated time and wait until other software finish their task. Matlab/Simulink has a User Datagaram Protocol (*UDP*) communication block, which provides a communication link to send and receive data between any process and Matlab/Simulink, this block also has a two modes, one is "Blocking" mode and other is "Non-Blocking" mode. If the blocking mode is used, the simulation will be halted or blocked as the name suggest until the message is received by the UDP receive block. The Figure 2.7, shows how the simulation is blocked at time step 2 when the requested data is not received by the block. By this way the Matlab/Simulink can be made to wait until every cycle is finished

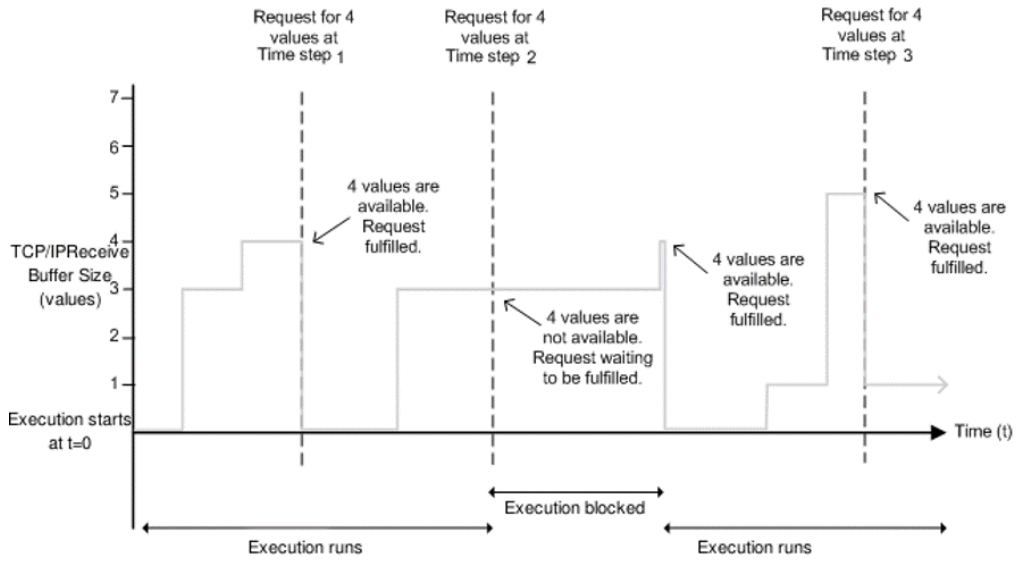

Figure 2.7: *Execution timing example of a simulation for a synchronized simulation[5]*

2.2.3 ADAPT Simulation

Commands in the ADAPT's command line interface can be provided over Transmission Communication Protocol (*TCP*). Similar to Simulink, ADAPT is already programmed to run for certain macro time steps (in milliseconds) according to the user input value, for example if user or test script specifies 10ms it will run only for 10ms and wait until it receives the next command.

2.2.4 Robot Operating System simulation

ROS is executed on Linux operating systems, such as Ubuntu, Xenial or Debian platforms. ROS is also possible to be executed on a Linux system running on a hypervisor such as VirtualBox. So ROS will always run in "Real-time", it is important to make sure that, this real-time operation will coincide with the other software programs for synchronization.

2.2.4.1 Application Level controlling

ROS is running as an application in Linux kernel, so there are API calls in Linux like sleep, wait etc., to pause and resume the process. A parent process is created in Linux and by using the *fork()* call, the child process is created which will run the ROS core. Now the parent process can control the child process (which is running the ROS core) to resume and pause at certain time steps. But if there is an addition of a new ROS process, the parent process code has to be changed. Thus, having a generic process control is more convenient when compared to a specific process control.

2.2.4.2 Linux kernel level controlling

As all process running in the Linux operating system obeys or synchronizes with the clock of the operating systems, so controlling the clock of the system will in turn control the process running time. In a Linux system the system will run the process according to the clock frequency of the system. For example, if a process is run for 10seconds and the frequency of the system is 10Hz that is, 0.1s or 10 clock cycles per second or 10 jiffies per second. Jiffies are the global variable which stores the number of clock cycles that occurred since system start-up.

From the Figure 2.8, when a hardware clock event occurs, the Interrupt request (IRQ) for the system is raised, which in turn will raise the clock event. The clock event will in turn trigger the timer interrupt which will increment the jiffies with the clock ticks of the system. One jiffy increment means one clock cycle increment.

$$Jiffies/second = system\_clock\_frequency \tag{2.5}$$

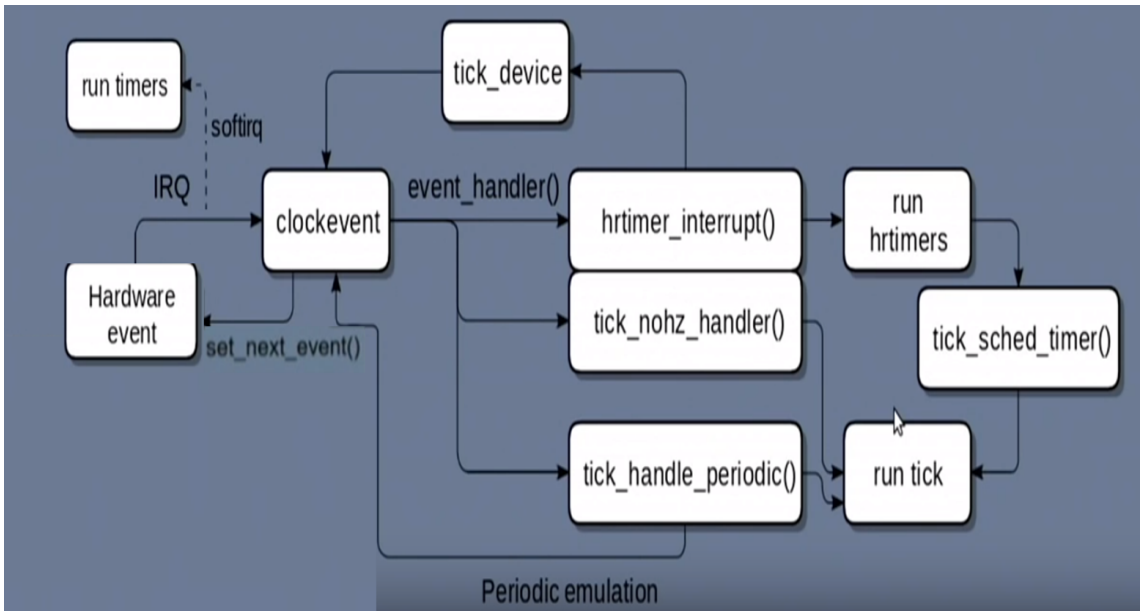

Figure 2.8: *Linux Kernel time keeping on any event [4]*

```
void xtime_update(unsigned long ticks)
{
        write_seqlock(&jiffies_lock);
        do_timer(ticks);
        write_sequnlock(&jiffies_lock);
        update_wall_time();
}
```

```
void do_timer(unsigned long ticks)
{
        jiffies_64 += ticks;
        calc_global_load(ticks);
}
```

Figure 2.9: *Linux Kernel build file timekeeping.c, line : 2375 - 2381,2182-2186*

This Jiffy increment is found in Timekeeping.c, in the kernel build files (kernel source files) illustrated in Figure 2.9. Introducing a control loop or a switch case in this timer update code, will make the jiffy increment wait until a certain update value is specified by the user or the test script. If the timer value is not updated, all running process in the Linux system will wait until the jiffy values are updated. Only hurdle in this process is writing into the kernel address from the host side (Windows operating system in this case) which is explained in section 3.2.3.

2.2.4.3 VirtualBox API calls

The hypervisor are of two types, type-I and type-II. Type-I hypervisor run different operating systems directly on the operating system. Whereas the type-II hypervisor will sit on top of the host operating system and will make use of the hardware clock and RAM of the host system. VirtualBox is a type-II hypervisor, as shown in the Figure 2.10, developed by Oracle Corporation. The VirtualBox has it's own (*API*) calls defined, in which VirtualBox Controlvm API is used to gain control over different operating systems running on Virtual box. This API is used to poweron, poweroff, pause the machine, start the machine etc.

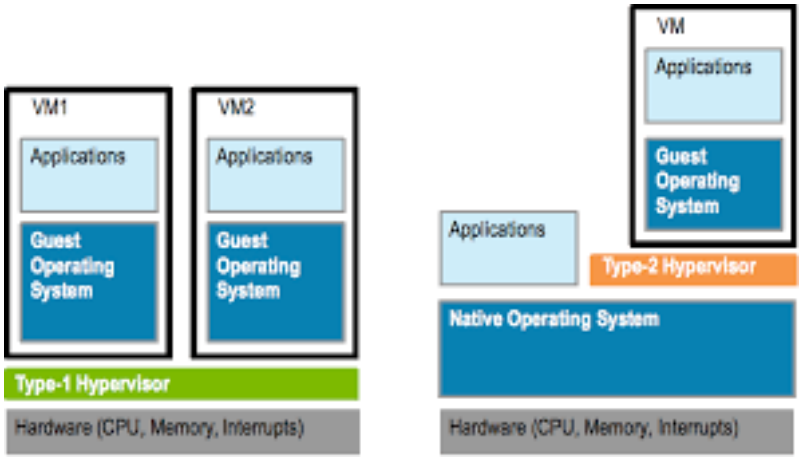

Figure 2.10: *TYPE-I and TYPE-II, hypervisor*

A diagrammatic representation of the synchronization manager controlling the simulation software is shown in Figure 2.11 Having discussed about the theory about the creating the synchronized simulation environment, devising this theory into practice is discussed in the methods chapter.

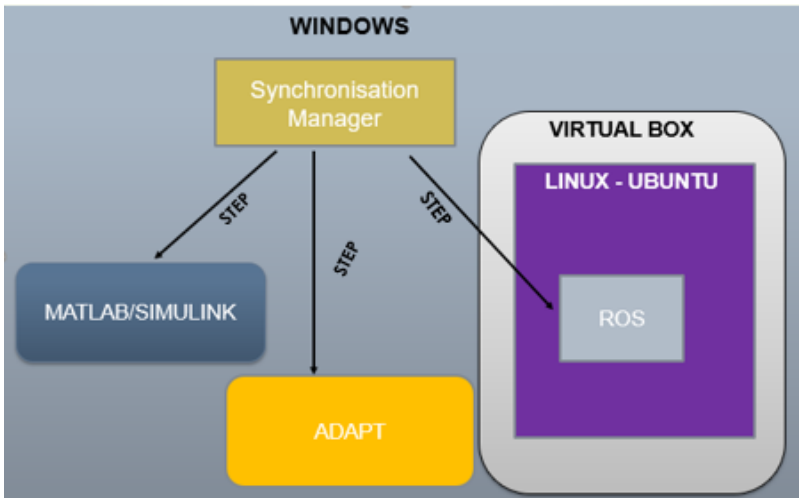

Figure 2.11: *An pictorial representation of how synchronization manager works*

3
Methods

3.1 Plant Model Integration

This section details about how the vehicle sub-plant of VTM is replaced by FMU from the single track open source model.
Recalling about the ADAPT platform, ADAPT has been conceived to integrate and simulate SW components and non-SW components together. The modeling pattern has interfaces at appropriate locations that aid in possible reuse for different verification purposes. [16]

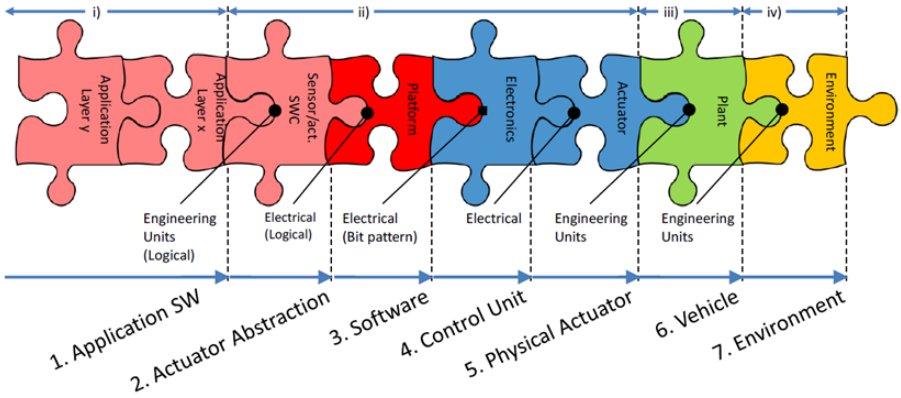

Figure 3.1: *Modeling Pattern for Embedded System proposed by Kaijser et.al[16]*

Hence we see that there is a requirement to have a clear distinction between the plant and actuators.

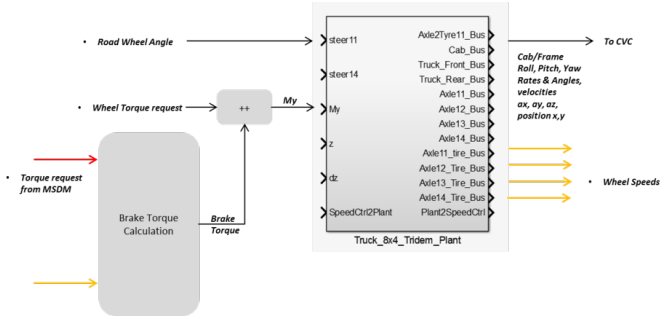

Figure 3.2: *Schematic representation of VTM library consisting of the vehicle sub-plant and actuators*

From the description of VTM, it can be noted that VTM contains the vehicle plant sub-system and models for steering actuators. Thus, there is no clear segregation between the vehicle plant(wheels and body) and sensors/actuating elements(such as Steering). Hence, from the standpoint of being able to provide a clear distinction between physical elements and actuators, the existing VTM system needs to be split into Chassis (Cab, Frame), Tyres, and Axles separating from Steering. However, if one approached this idea of segregation of the vehicle model from the point of being able to simulate for same actuator models of say steering, for different vehicle combinations or product portfolio, it would be beneficial to separate the tyre model from the mix and the remaining set of Chassis (Cab, Frame), and Axles would then be purely representative of the vehicle body, which can be replaced or modified depending on the the required fidelity. This set of Chassis (Cab, Frame) and Axles has been replaced by the open source Model as a FMU.

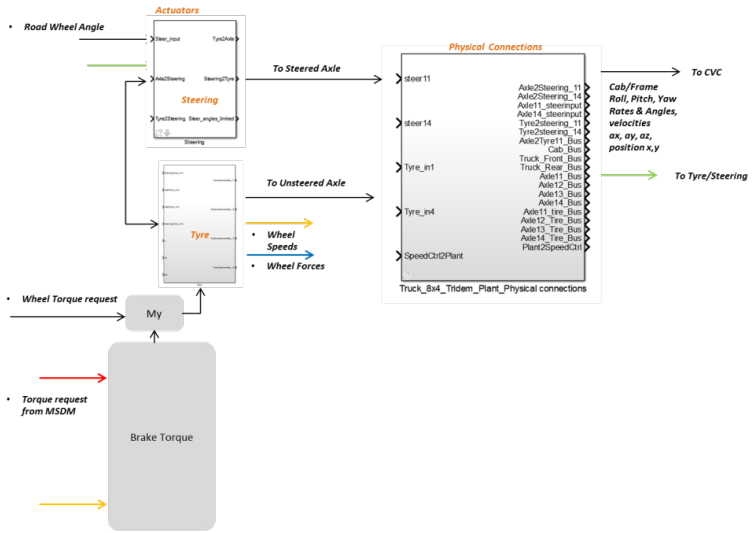

Figure 3.3: *Schematic representation of proposed segregation of VTM*

Figure 3.4 represents how the functionality domains TSM,VMM, MSDM and Plant model (VTM) are connected.

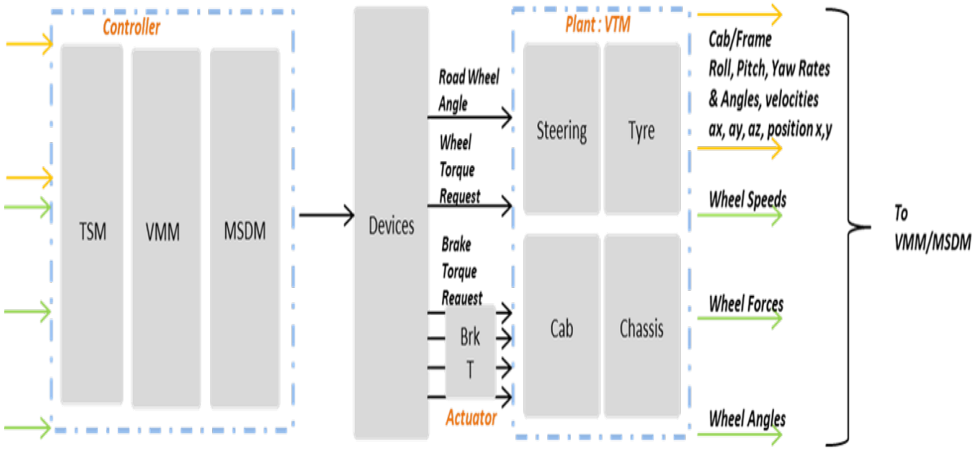

Figure 3.4: *Representation of the simulation architecture when using VTM as plant model*

Figure 3.5 represents how the functionality domains TSM,VMM, MSDM and Plant model (FMU) are connected.

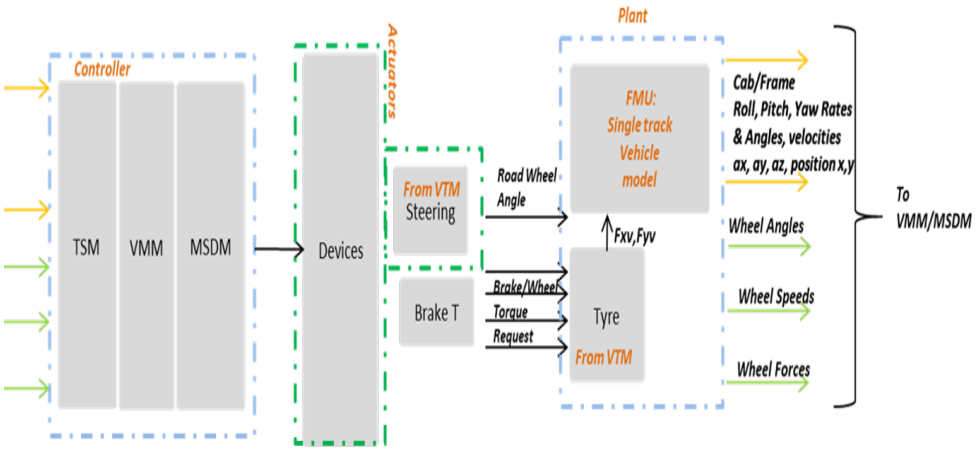

Figure 3.5: *Representation of the simulation architecture when using FMU based plant model*

The signal interfaces are maintained while VTM is segregated and then swapped with the FMU. The inputs to the single track vehicle model FMU are Forces in the vehicle co-ordinate system and the outputs available are the position along x,y, Yaw rates, angles, longitudinal accelerations and velocities, lateral accelerations and velocities.

3.2 Synchronization Control

In the following sections, the synchronization control algorithm is explained. Python programming language is used to device the algorithm for synchronization manager.

3.2.1 Matlab control

As discussed in the section 2.2.2 the UDP send and receive blocks help to make the simulation pause / maintain the blocked state until it receives the message. The UDP blocks in Simulink library is used as shown below, the IP address used is the "local-host" IP address, "127.0.0.1", the port 10004 is defined by the user. The UDP send block, sends a data back to synchronization manager through the same IP address but through a different port 10003, as UDP can not use the same port to send and receive to an external process. When the simulation is completed by this block, it can be concluded that one simulation cycle is completed. Until the data is received by the synchronization manager, the second cycle will not be started.

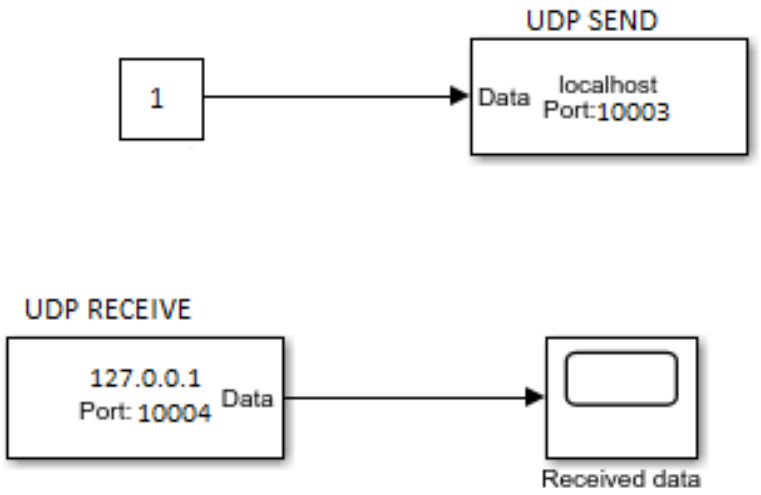

Figure 3.6: *UDP Send and Receive block Simulink Library*

3.2.2 ADAPT Control

The ADAPT works with TCP protocol as discussed in the section 2.2.3, in-order to control ADAPT, the data is sent through TCP ports to start the simulation of one cycle by the synchronization manager, similar to Matlab/Simulink Control. On completion of one simulation cycle the ADAPT sends back an "OK" signal through the same TCP port, TCP ports can be used to send and receive data through the same port at a time. When the "OK" signal is received, the synchronization manager will execute the next cycle. The port used here is assigned by ADAPT and the IP is the local host address. To have a co-simulation setup, "OK" signal(s) should also be received from the other software involved in the simulation.

3.2.3 VirtualBox-Linux-Ubuntu Control

As discussed in the section 2.2.4.2, the Timekeeping.c is modified as shown in the Figure 3.7. The jiffies will be controlled by the synchronisation manager. When the *time_init* is greater than zero the jiffies will be updated which will make the system to run and in-turn the application will run. If the *time_init* will be decremented to zero, when the *time_init* equals zero the system will be halted until the new value is written.

```
Void do_timer(unsigned long ticks) {
        get_user(flag_set_time * timer_adrs1):
                switch(flag_set_time){
                case Regulated_Time :
                        copy_from_user(timer_int, timer_Adrs2)
                        if(timer > 0){
                                jiffies_64 -= ticks;
                                calc_global_load(ticks)
                                timer -= 1
}
}
}
```

Figure 3.7: *Modified example of the Linux kernel Timekeeping.c*

When changing the kernel timer, the *timer_init* should be written from windows side onto the Linux kernel. To do this, the kernel address has to be translated to the corresponding physical address from the guest operating system to the host operating system. The $ReadProcessMemory()$ API call has to be used, to read the memory of any process using the Process ID (*PID*) of the process. This API call can be used if and only if the user/developer of the application has given the access or permission to read the memory of that process, else the function will result in an error $ACCESSDENIED$, which infers that the given memory is protected memory. VirtualBox developer has blocked the access to the memory read and writing by the user. So, this method can not be used to start and stop the system.. So the VirtualBox API call is instead used to control the VirtualBox system. The

VirtualBox documentation has many API call, of which "VBoxManage controlvm" will allow the user to change the state of the running virtual machine. The API, *VBoxManage controlvm <vm> pause* will put the "vm" virtual machine on hold without changing the state of the system, no hardware IRQs or process will occur

when the system is on hold. This API call is issued from the Windows (HOST) end, as soon as the system is ready for simulation. When the first cycle of simulation is started the synchronization manager calls the *VBoxManage controlvm <vm> resume* API and waits for certain time instances before the pause API is called back to put the system to halt. This "WAIT" time is equal to the running time of Matlab/Simulink. For example, the Matlab/Simulink runs for 10ms of simulated time, the wait time is also made to 10ms.

3.2.4 Pesudo CODE

The pesudo code for the above algorithm is given below, which is followed by the section covering the exact code implementation.

- The total simulation time to be obtained
- The ports should be initalized
- The ports should be connected
- At the first simulation time step, @t = 0
  - Send the step command to the UDP port in Matlab/Simulink to start Matlab/Simulink
  - Send step command with the appropriate time stamp through TCP port to ADAPT
  - Start Linux-Ubuntu system, run for 10ms and issue pause command
- Receive the data from UDP send port in Matlab/Simulink
- Receive data from ADAPT
- Ensure Linux-ubuntu system is paused
- Check for the received data.
- If the correct data has been received then run the next cycle until simulation total time.

3.2.5 Implementation of Pseudo-code

```
#-------------main loop start-----
#this condition is set instead of ..
#.."while total_simulation_time > -1" ..
#..because the loop should be in waiting..
#..state - the else condition (#---#),
#the total simulation is run for..
#..total_simulation_time+1 as loop starts from 0
while loop == True:
        if total_simulation_time > -1 :
                if total_simulation_time == time_first :
                #to check all platforms are started and made ready for the run
                        matlabsend(step_time,matlab_port_send)
                        step_next = matlabrecv(matlab_port_recv,init)

                        adapt_step = adapt_ctrl('0',adapt_port,init)
```

```
                    linux_step = controlvb(computer_name,step_time_VB,..
                    ...init,exe_speed)
                    #pausing the linux VB

                    total_simulation_time -= 1
                    #total simulation time is reduced
                    start_system = 1
                    #just to make sure the every..
                    #..thing is started and pause for initial run
                    print("started")

            elif start_system == 1 and ...
            ...adapt_step[:8] == b'Step#OK#'...
            ...and linux_step == 1 and  step_next == b'\x01':
            #check the condition that every..
            #..platform is ready for the nexr run..
            #if yes send data to all platforms
                    matlabsend(step_time,matlab_port_send)
                    step_next = matlabrecv(matlab_port_recv,run)

                    adapt_step = adapt_ctrl('10',adapt_port,run)

                    linux_step = controlvb(computer_name,...
                    ...step_time_VB,run,exe_speed)

                    total_simulation_time -= 1

            else:  #else wait for the system to change
                    step_next = matlabrecv(matlab_port_recv,run)
                    print ("waiting for matlab",step_next)

    elif total_simulation_time == -1:
    #once the total simulation time reaches zero end simualtion
                    loop = False
                    print("simulation complete")
    time.sleep(1)
```

4

Integration Setups

The following chapter illustrates the integration setups. Virtual integration has been tried for the combinations described in the following sections.

4.1 VTM in Simulink, VMM&MSDM in ADAPT, TSM in Simulink

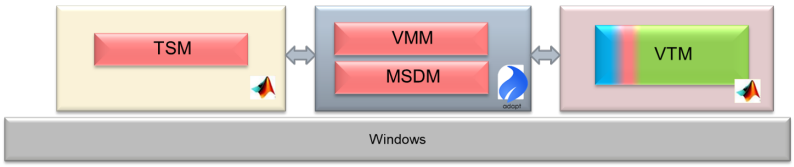

Figure 4.1: *VTM in Simulink, VMM&MSDM in ADAPT, TSM in Simulink*

The integration setup has the TSM as a function block in Simulink and VMM&MSDM in ADAPT and VTM in Simulink. The data exchange between the functionality is through virtual CAN interface.

4.2 VTM in Simulink, VMM&MSDM in Simulink

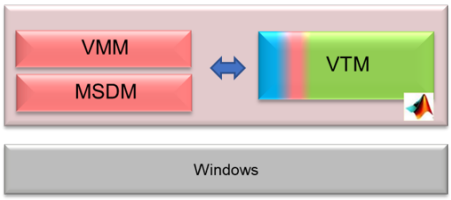

Figure 4.2: *VTM in Simulink, VMM&MSDM in Simulink*

Objective: To test the integration of VTM, VMM&MSDM in Simulink for manual steering inputs. The ADAPT versions of VMM&MSDM are converted into function blocks in Simulink.

4.3 OpenPBS (FMU) in Simulink, VMM&MSDM in Simulink

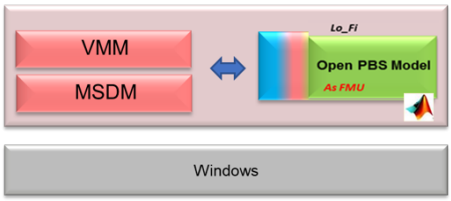

Figure 4.3: *OpenPBS (FMU) in Simulink, VMM&MSDM in Simulink*

Objective: Section 3.1 presented the integration of vehicle sub-plant from OpenPBS as a FMU with the steering actuator and Tyre Model from VTM. This low fidelity plant model and VMM&MSDM in Simulink is simulated for manual steering inputs.

4.4 VTM in Simulink, VMM&MSDM in Simulink, TSM in Simulink

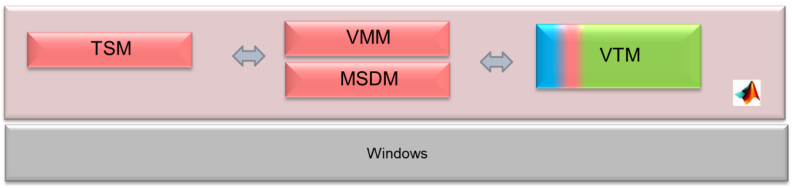

Figure 4.4: *TSM, VMM& MSDM, VTM executed in Simulink*

Objective: To have the setup as a reference as all the components are in a single tool and remove Virtual CAN Interface for Communication.

4.5 OpenPBS (FMU) in Simulink , VMM&MSDM in Simulink, TSM in Simulink

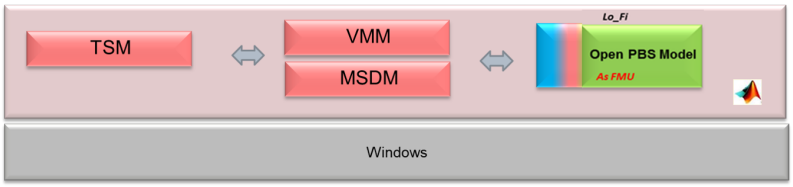

Figure 4.5: *TSM, VMM&MSDM executed in Simulink with single track model*

Objective: To study the simulation of the single tool (Simulink) setup upon swapping vechile sub-plant of VTM with low fidelity OpenPBS FMU.

4.6 VTM in Simulink, VMM&MSDM in ADAPT, TSM in Simulink with sync manager

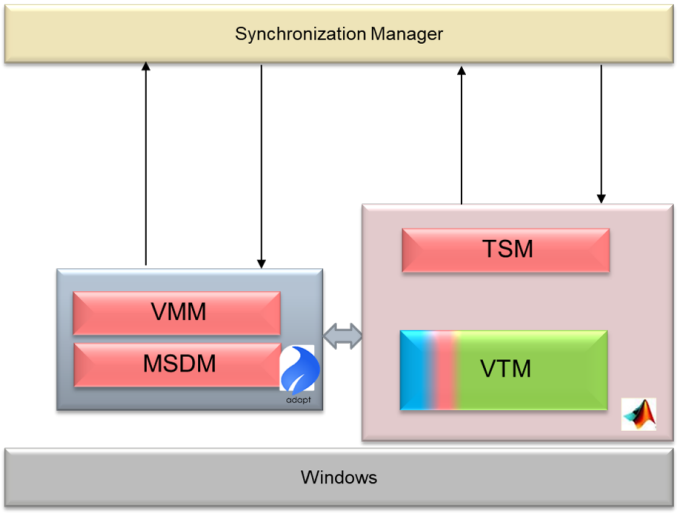

Figure 4.6: *TSM in simulink, VMM&MSDM in ADAPT, VTM in Simulink executed with synchronization manager*

Objective: To check the synchronization manager for the existing setup.

4.7 VTM in Simulink, VMM&MSDM in ADAPT, TSM in ROS with sync manager

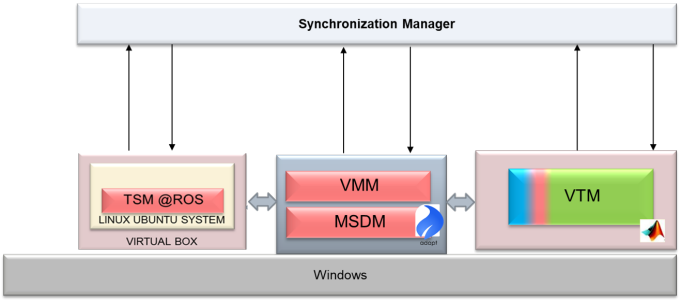

Figure 4.7: *TSM in ROS, VMM&MSDM in ADAPT, VTM in Simulink executed with synchronization manager*

Objective: To check the synchronization manager with ADAPT , ROS and Simulink software.

5
Results

This chapter presents the simulation results for the integration setups.

5.1 VTM in Simulink, VMM&MSDM in ADAPT, TSM in Simulink

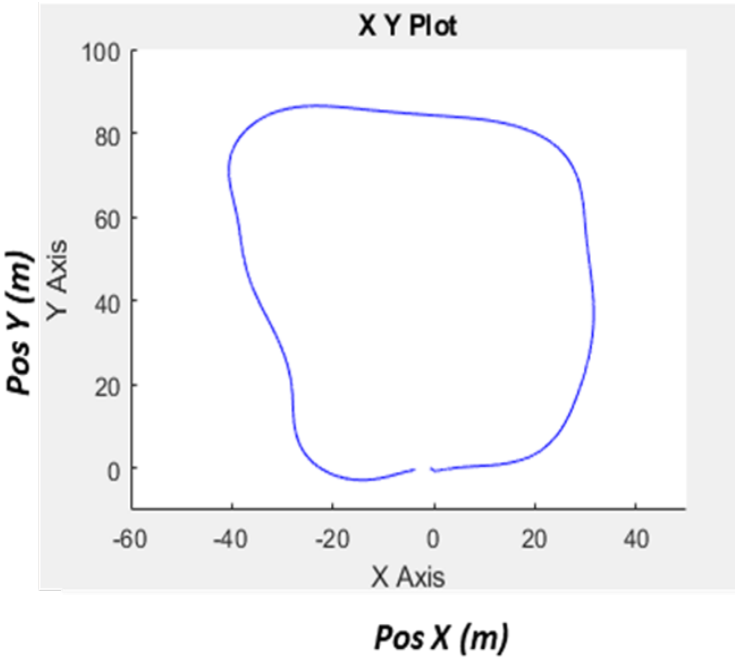

Figure 5.1: *VTM in Simulink, VMM&MSDM in ADAPT, TSM in Simulink*

In this scenario, the controller VMM&MSDM is placed in ADAPT and the plant model VTM and TSM is placed in Simuliunk.

5.2 VTM in Simulink, VMM&MSDM in Simulink

Zero Steering
VTM is simulated for straight line driving scenario with initial velocity v_0= 0m/s and a constant longitudinal request of 10m/s.

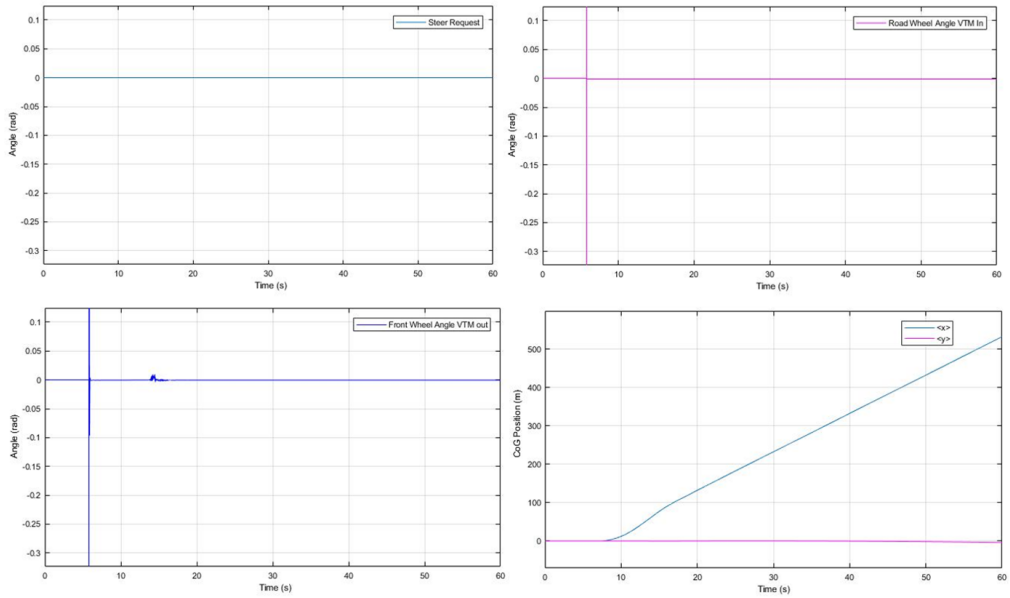

Figure 5.2: *VTM+VMM&MSDM for zero Steering Request, 10m/s velocity*

Sine Steering

VTM is simulated for a sinusoidal steering input of 0.02 rad amplitude and at a frequency of 0.1*π rad/s, for an initial velocity v_0= 0m/s and a constant longitudinal request of 10m/s.

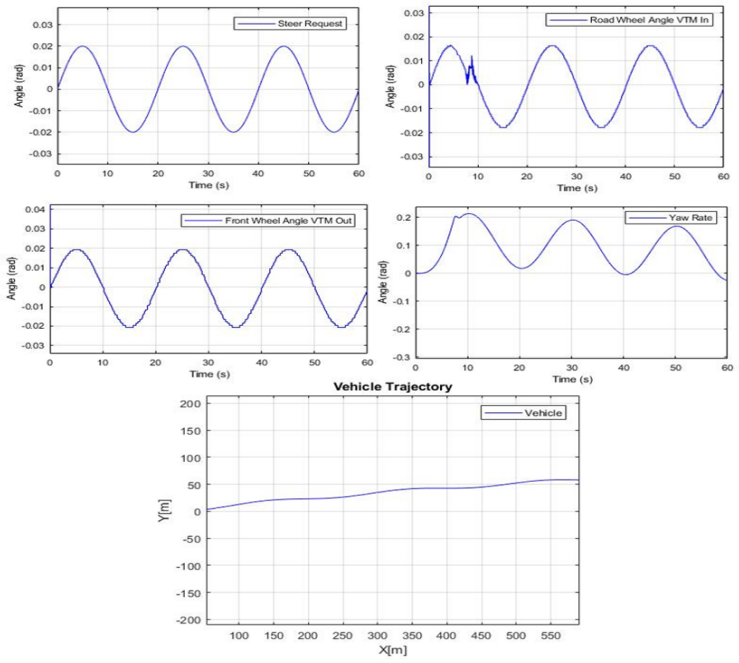

Figure 5.3: *VTM+VMM&MSDM for 0.02 sine Steering Request, 10m/s velocity*

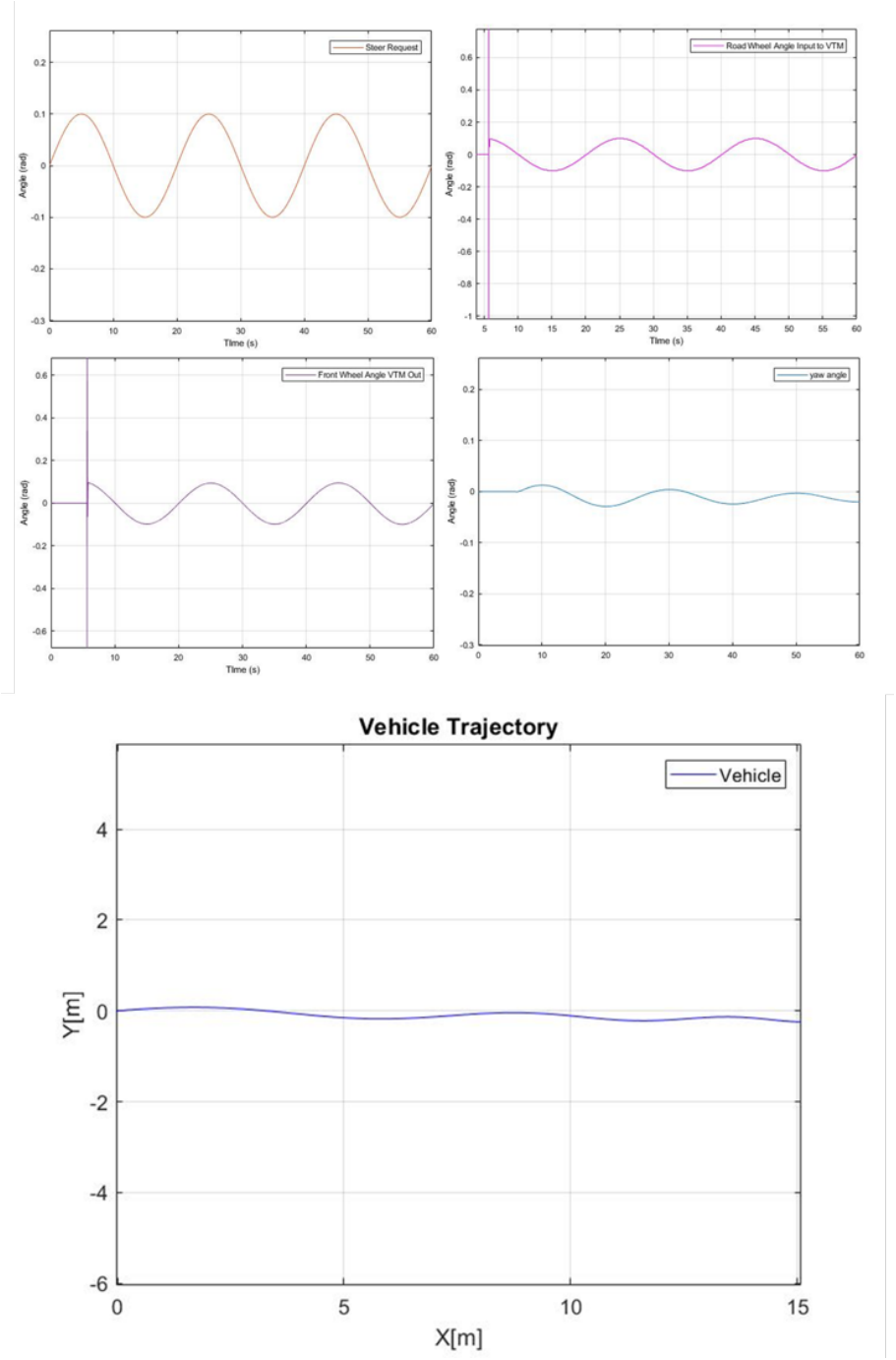

Figure 5.4: *VTM+VMM&MSDM for 0.1 sine Steering Request, 20m/s initial velocity*

Figure 5.4 shows the vehicle motion of VTM with initial velocity v_0= 20m/s, with no further longitudinal request and being steered at 0.1 rad amplitude and at a frequency of 0.1*π rad/s.

Sudden Steering

VTM is simulated for step steering input of 0.02 rad at time t=20s during the simulation and maintaining the steer input constant till the end of simulation, for an

initial velocity v_0= 0m/s and a constant longitudinal request of 10m/s.

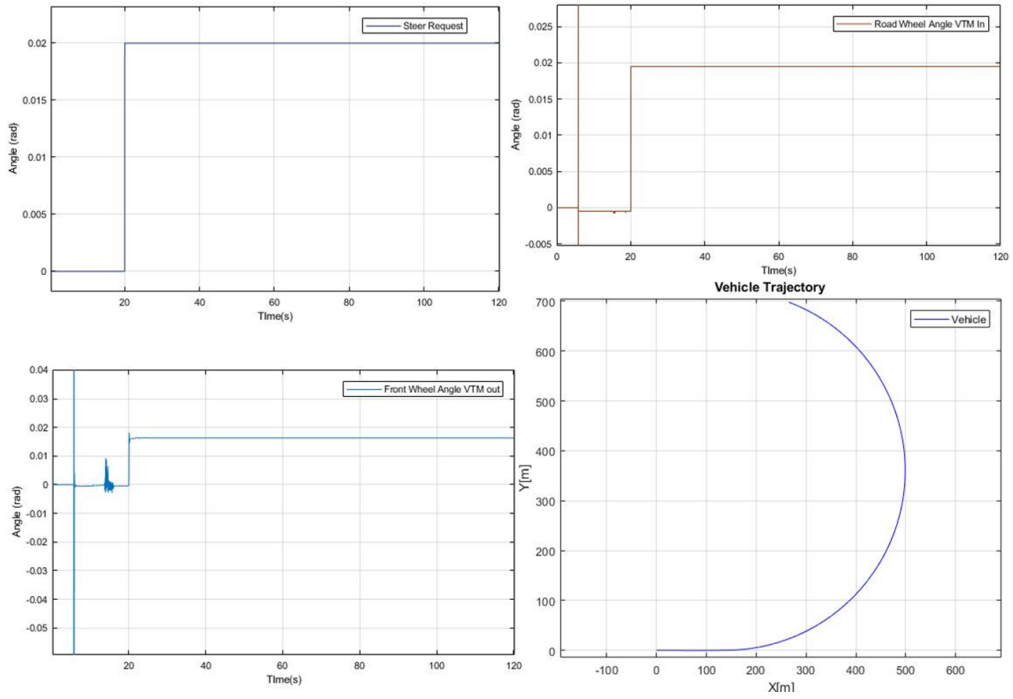

Figure 5.5: *VTM+VMM&MSDM for 0.02 step Steering Request, 10m/s velocity*

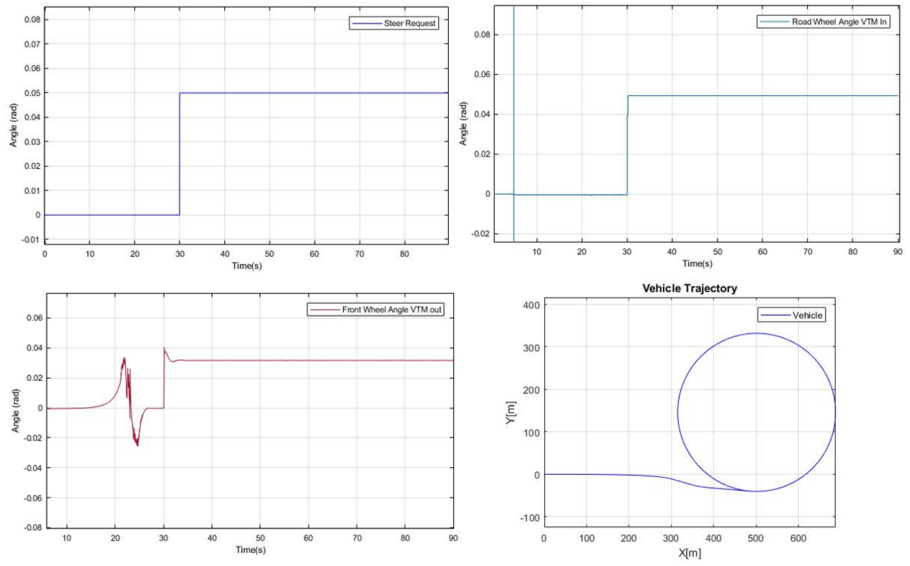

Figure 5.6: *VTM+VMM&MSDM for 0.05 sudden Steering Request, 20m/s velocity*

Figure 5.6 shows the vehicle motion of VTM for a sudden steer input 0.05 rad at time t=30s during the simulation and maintaining the steer input constant till the end of simulation, for an initial velocity v_0= 0m/s and a constant longitudinal request of 20m/s.

5.3 OpenPBS (FMU) in Simulink, VMM&MSDM in Simulink

Zero Steering

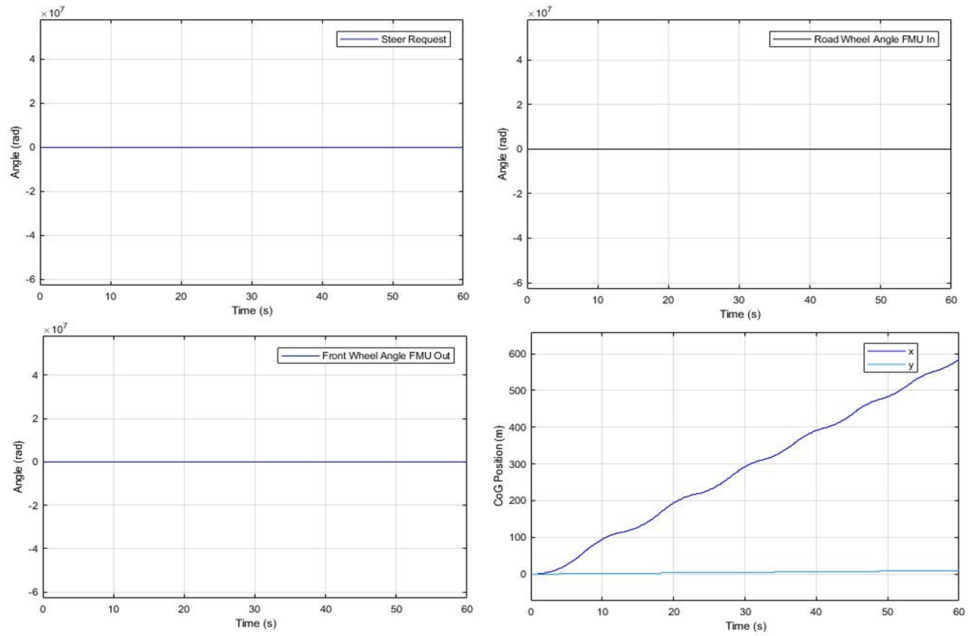

Figure 5.7: *Open source single track plant model+VMM&MSDM for zero Steering Request, 10m/s velocity*

The OpenPBS based plant model is simulated for straight line driving scenario with initial velocity v_0= 0m/s and a constant longitudinal request of 10m/s.

Sine Steering
The OpenPBS based plant model is simulated for a sinusoidal steering input of 0.02 rad amplitude and at a frequency of 0.1*π rad/s, for an initial velocity v_0= 0m/s and a constant longitudinal request of 10m/s.

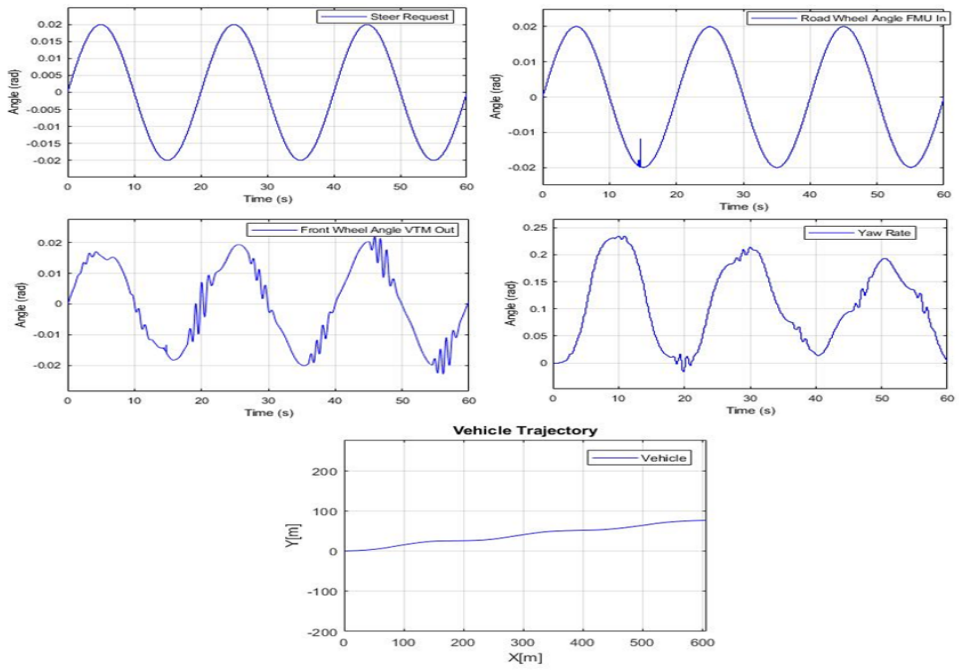

Figure 5.8: *Open source single track plant model+VMM&MSDM for 0.02 sine Steering Request, 10m/s velocity*

Sudden Steering

The OpenPBS based plant model is simulated for step steering input of 0.02 rad at time t=20s during the simulation and maintaining the steer input constant till the end of simulation, for an initial velocity v_0= 0m/s and a constant longitudinal request of 10m/s.

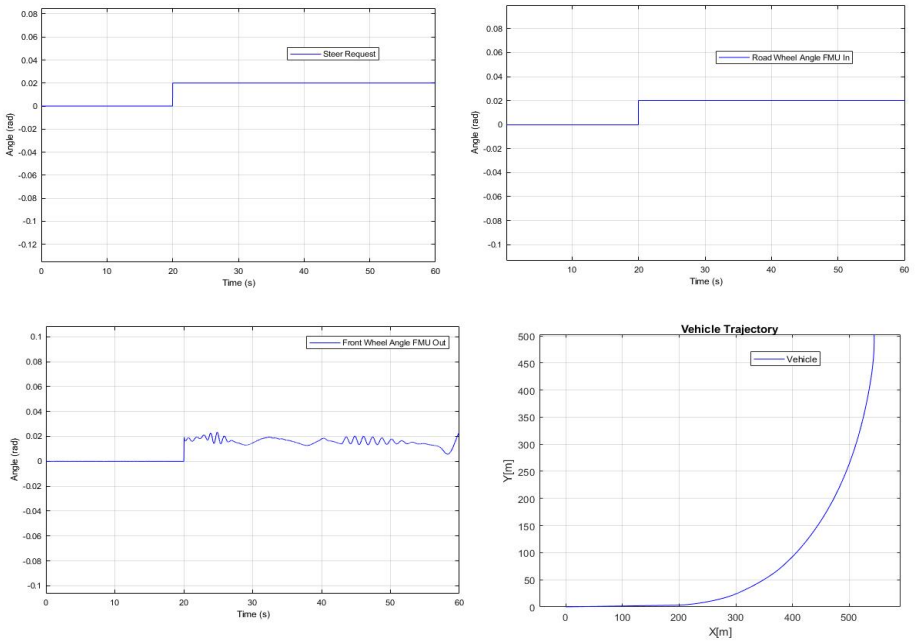

Figure 5.9: *Open source single track plant model+VMM&MSDM for 0.02 step Steering Request, 10m/s velocity*

Figure 5.10 shows the vehicle motion of OpenPBS based plant model for a sudden steer input 0.05 rad at time t=30s during the simulation and maintaining the steer input constant till the end of simulation, for an initial velocity v_0= 0m/s and a constant longitudinal request of 20m/s.

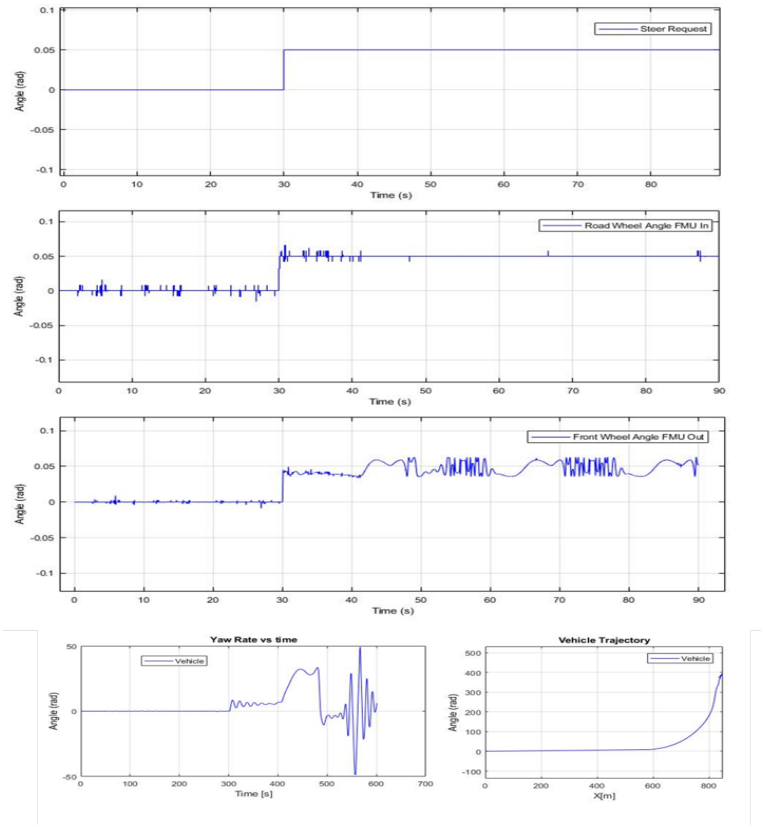

Figure 5.10: *Open source single track plant model+VMM&MSDM for 0.05 sudden Steering Request, 20m/s velocity*

Upon simulating the vehicle models for simple maneuvers, it was observed that at low speeds the integration setup of VTM + VMM& MSDM and Plant model based on FMU from OpenPBS + VMM& MSDM behaved as expected. Typical longitudinal effects such as load transfer is not taken into consideration. The dynamic load transfer on tires is also absent. The two tires on an axle are lumped into a single tire in the middle of the axle. Hence, the input force from the tyre block is an average value. The other assumption is that the coupling points have no torque transfer between units. Furthermore, the Cab and frame have been lumped into one body and the units are not suspended on the tires. Hence, the inertial sensor units in the controller receive the same signal. Finally, the controller gains that have been set for VTM would need to be modified accordingly for simulating the open source model. Owing to these factors, there are slight variations in the simulations result.

5.4 VTM in Simulink, VMM&MSDM in Simulink, TSM in Simulink

The goal of the integration was to have all the functionality domains in Simulink. Hence, the ADAPT versions of VMM&MSDM was converted into function blocks in Simulink. VMM&MSDM by itself is composed of several sub-function blocks. The respective signals were directly connected without using the Virtual CAN interface. The plant model (VTM) executes at a continuous rate, but the controller executes at 10ms sample rate. Hence, there necessitates a requirement to add rate transitions and zero order hold blocks at both input and output signal interfaces from the plant model to the controller in order convert signals between continuous and discrete time. These rate transitions and delay blocks also help in breaking the algebraic loop in these signals.
When a model contains an algebraic loop, Simulink uses a nonlinear solver at each macro time step to solve the algebraic loop. The solver performs iterations to determine the solution to the algebraic constraint, if there is one. Simulink needs the value of the block's input signal to compute its output at the current time step. As a result, models with algebraic loops can run more slowly than models without algebraic loops. [15] On simulation, the vehicle was observed to follow the path only partially. TSM provides the heading and the position to the vehicle. This is done by calculating the trajectory of the vehicle based on the difference between the current position and the set of reference path positions. It is possible that TSM has received wrong data at a certain time step due to the introduction of the rate transitions or due to any errors in the VMM&MSDM integration and the same accumulated error over a long period has caused a large deviation and the vehicle, unable to steer to the reference trajectory.

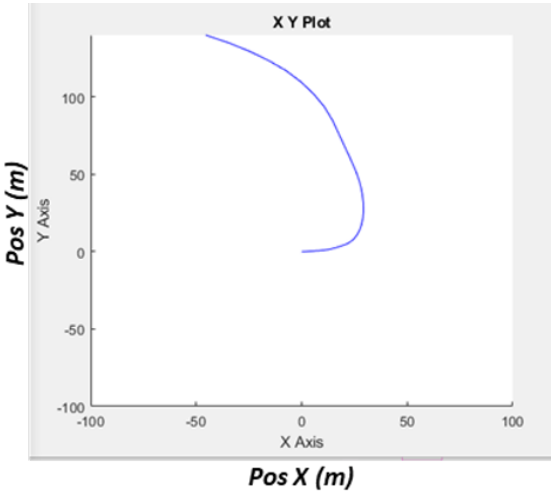

Figure 5.11: *Simulation result for VTM in Simulink, VMM&MSDM in Simulink, TSM in Simulink*

5.4.1 VTM in Simulink, VMM&MSDM(only required sub function blocks) in Simulink, TSM in Simulink

Integration setup in Section 5.4 is retried with using only those function blocks within VMM& MSDM viz., Inertial Motion Unit, Motion Estimation, Motion Co-Ordination, Motion Stability, Target Generation that interacted with TSM and the required vehicle parameters were fed through as constants referring back to the original VTM library. This prevented the occurrence of algebraic loops as a result of the above blocks having a direct feed through since they became sub systems under VMM&MSDM in the integration setup in Section 5.4 . However, the required rate transitions and zero order hold are still needed as the sample rates for these blocks are different from VTM. It was observed that the vehicle followed the path generated by the TSM path follower and the execution time was faster than the previous integration setup.

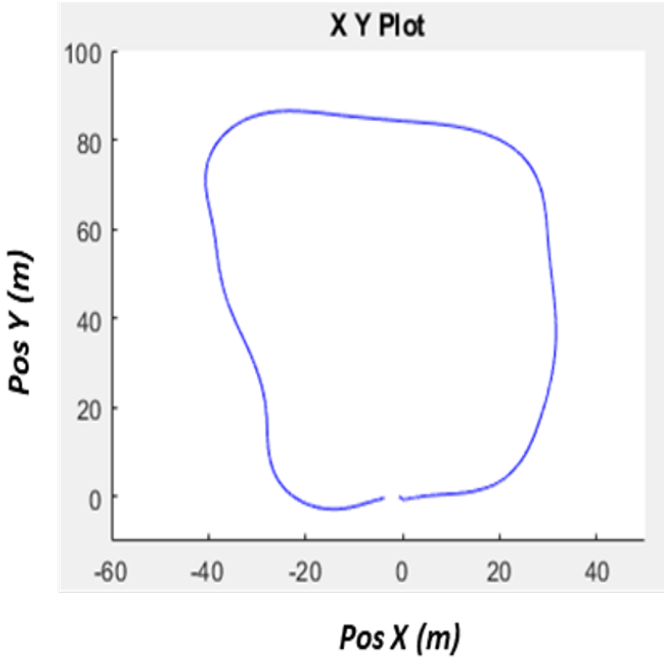

Figure 5.12: *Simulation result for VTM in Simulink, VMM&MSDM(only required sub function blocks) in Simulink, TSM in Simulink*

5.5 OpenPBS (FMU) in Simulink , VMM&MSDM in Simulink, TSM in Simulink

VTM based vehicle plant in Integration setup in Section 5.4 was swapped with a FMU node generated for a single track vehicle dynamics model as described in Section 2.1.2. As explained in 5.3, in the single track model we have lumped the masses, tyres, cab and frame. The inertial units in the controller(VMM&MSDM) have been modelled to recive separate signals from cab and frame. These now receive the same values of yaw rate due to lumping of cab and frame into one body. Since, in the single track model, the vehicle is not on a suspended frame and does not have detailed modeling for vertical dynamics, the signals for pitch rate, roll rate,vertical acceleration a_z were provided with zero input. These signals are inputs to several blocks in VMM&MSDM. When this integration was simulated for the TSM path follower, it was observed that the execution time was faster than the integration setup in 5.4 and 5.4.1 vehicle followed the path to almost half of the generated path.

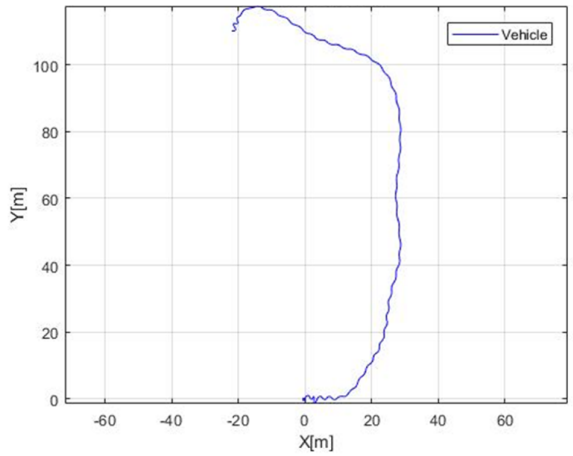

Figure 5.13: *Simulation result for Open source based plant model in Simulink, VMM&MSDM in Simulink, TSM in Simulink*

5.5.1 OpenPBS (FMU) in Simulink , VMM&MSDM(only required sub function blocks) in Simulink, TSM in Simulink

In this scenario, instead of using the complete VMM& MSDM controller, only the required function blocks within VMM& MSDM was integrated with the FMU based vehicle plant. It was still observed that the vehicle was able to follow the TSM path follower partially.

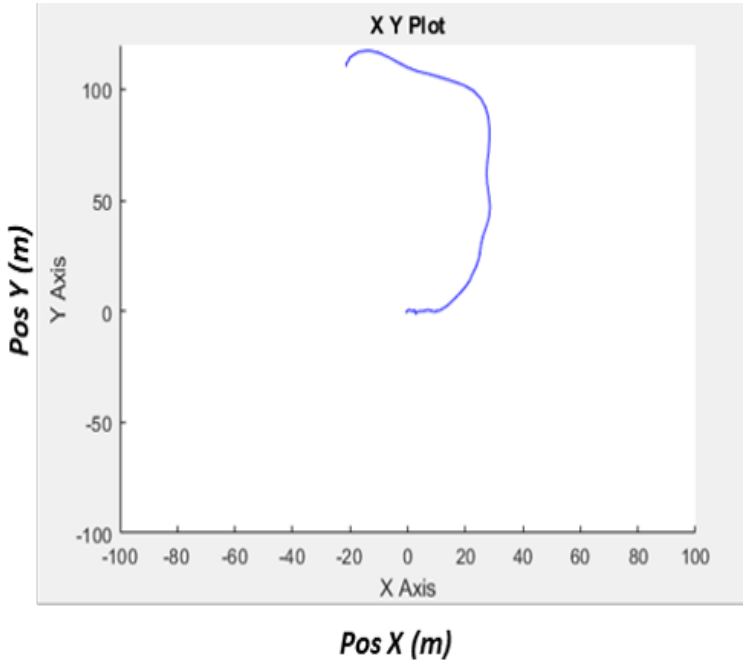

Figure 5.14: *Simulation result for Open source based plant model in Simulink, VMM&MSDM(only required sub function blocks) in Simulink, TSM in Simulink*

As previously mentioned, TSM provides the heading and position to the vehicle. TSM has the reference trajectory points for the path for reference, checks the vehicle's current position and then provides next heading and position. In the above integration setups where the path was followed partially, the vehicle has deviated largely from the reference trajectory points that it is unable to return to the intended path, as the vehicle has a defined maximum and minimum steering rate. The simulation in all these scenarios is halted midway as the TSM is unable to calculate the trajectory to the next point.
This is due to the complexity of the controller that has been designed for a complex system. When using the same controller for a simpler system the output will not be the same as the output of the complex system, as some assumptions made to derive the simple system from the complex system would affect the controller's performance due to the Gain and eigen values set for the controller. In our case the simpler system is the OpenPBS vehicle model in the FMU and the complex system is the VTM 8x4 model. Another fact is that the delay compensation might affect the system output. As we can see from [25] and [24], in co-simulated systems, the errors in simulation are dominated by the delay effects and coupling errors. These errors could be solved by appropriate delay compensation with coupling error correction.

Synchronization
The synchronization manager was tested with two different test setups. The first setup is Simulink along with ADAPT. The other setup is Matlab/Simulink, ADAPT and ROS.

5.6 VTM in Simulink, VMM&MSDM in ADAPT, TSM in Simulink with sync manager

ADAPT and SIMULINK Setup
The Plant model(VTM) and the TSM is placed in the Simulink whereas the controller is placed in ADAPT. The synchronization manager is used to run these two software in synchronization. The synchronization manager will control the ADAPT and Simulink, to obey the synchronization rule as stated in the section 2.2.1 and also to test the real time scenario. The plot generated from the simulation as shown in the Figure 5.15, shows that the truck is able to follow the path generated by the TSM. The total execution step for the synchronization manager to run the set-up was about 12000 steps, where each step was 10ms (macro time step).

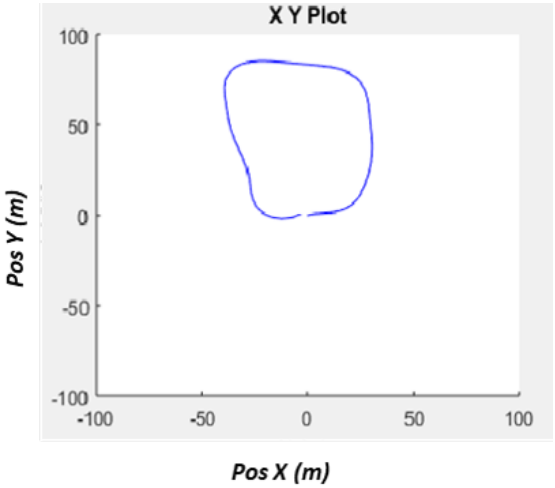

Figure 5.15: *Simulation result for VTM in Simulink, VMM&MSDM in ADAPT, TSM in Simulink with sync manager*

5.7 VTM in Simulink, VMM&MSDM in ADAPT, TSM in ROS with sync manager

SIMULINK, ADAPT and ROS Setup
In this case the plant model(VTM) and TSM is placed in simulink where as the controller is placed in ADAPT. A simple send and receive module between two process is configured in ROS along with the Matlab/Simulink and ADAPT. On testing ROS along with the other platforms with synchronization manager, it was observed that the Linux system updates every second on the real-time clock, MATLAB and

adapt has to run every second to obey the synchronization. On increasing the synchronization time from 10ms to 1s, it was observed that the Simulink has it's own tick counter, which counts the missed ticks, if this number exceeds a certain value then Simulink will crash. So there is a maximum granularity of setting the synchronization cycle time for the synchronizaiton.
Figure 5.16, shows the paused state of all three platforms and waiting for the next step command from the synchronization manager.

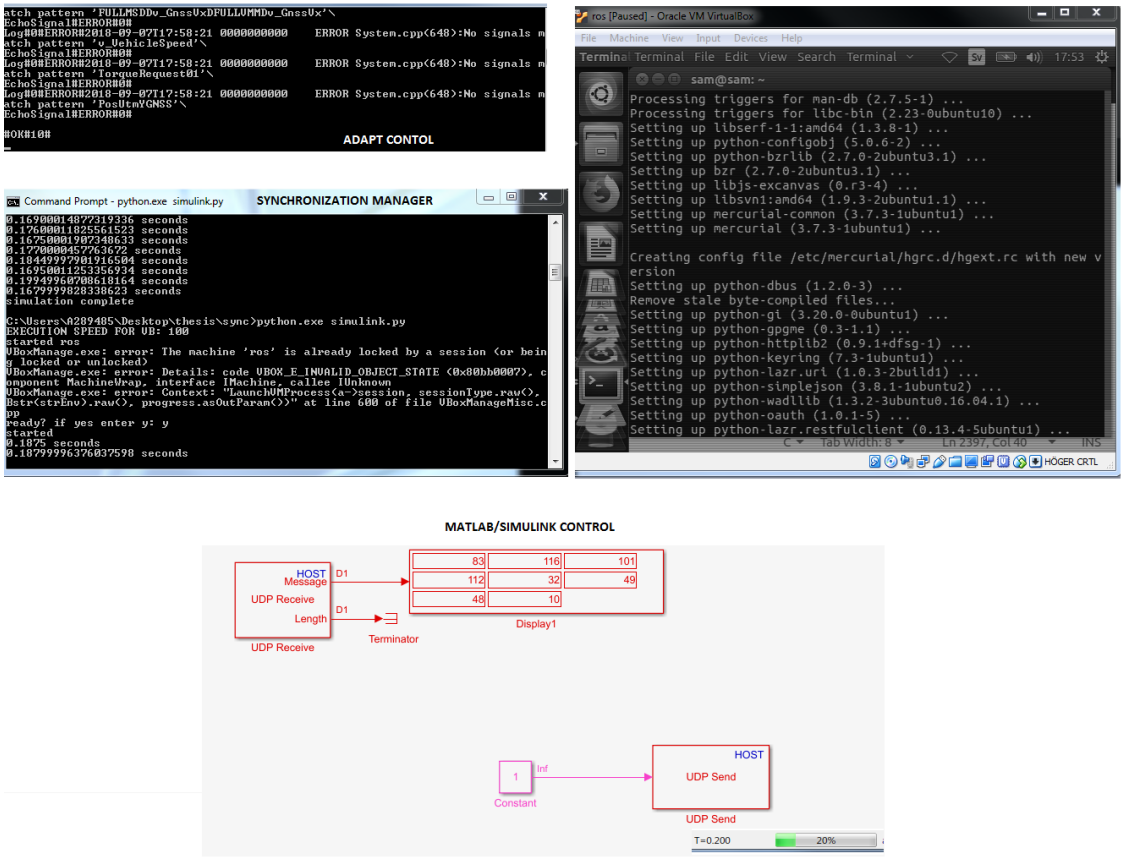

Figure 5.16: *At T=2 Of simulation time, paused state of all software*

The control of the Linux system is running in "while" loop, the time taken for the execution will be different every time so, the exact time at which the system is stopped will be little more than 10ms, that is, it might 10.18ms or 10.19ms or even 11ms sometimes. This would be a problem in long simulation runs. Inorder to avoid the excess running of the simulation, the execution speed of the Linux system can be modified with the Virtuabox API call, *VBoxManage modifyvm <vm> cpuexecutioncap <range>*, to make the system run slower than the normal system clock frequency. The "*<range>*" is between 1% and 100%. The 100% system frequency is the actual or the normal system clock frequency. On reducing the system frequency lesser than 100% would affect the execution time of process running in the system. Using this api the VirtualBox can be controlled to synchronize with other software. It would also be good to see how the fmu works with the synchronization manager, due to time constraints this scenario was not tested during this thesis.

6
Conclusion and Future Work

6.1 Conclusion

Several integrations with and without a synchronization manager were tried and tested for their simulation capability. VTM was successfully segregated into vehicle sub-plant (body components viz, cab, frame and axle) and actuators. OpenPBS vehicle model for a rigid vehicle was created, and was converted into a *Co-simulation* type FMU. This FMU was successfully integrated with the steering actuators and Tyre model from VTM.
At low speeds and simple steering conditions, the vehicle plant integration based on OpenPBS and VTM behaved similarly but with small variations and fluctuations in the plots. The reason for such fluctuations are possibly due to lumping of vehicle bodies (i.e, cab and frame into a single rigid body) and the modeling assumptions made in the OpenPBS. Typical longitudinal effects such as load transfer is not taken into consideration in the model. The dynamic load transfer on tires is also absent. The two tires on an axle are lumped into a single tire in the middle of the axle. The modeling assumptions are further compounded by the simplifications made while integrating the FMU to the steering actuator, Tyre models and the controllers i.e, VMM&MSDM, TSM, in form of averaging the signals for vehicle yaw rates to the inertial sensor units and the input tyre forces. The controllers have been designed for a complex system like VTM. Hence, the controller gains and eigen values that have been set for working with a higher order plant model such as VTM , when executed with a simpler plant model like OpenPBS, could contribute to the fluctuations.
Furthermore, vehicle models in VTM have been validated only for few driving scenarios and the modelling does not consider the effects of variation in ambient temperature and friction of road surface. The limitations for any model based virtual integration is that a single integration may not handle all scenarios. On the other hand, for low speeds, one can use the simple single track plant model after accordingly checking for the error tolerances by tuning accordingly.

6.2 Future Work

6.2.1 Model & Integration

The integration method presented in this work could be further tuned to reduce the errors that are observed while integrating the OpenPBS vehicle sub-plant FMU with actuators and tyre model from VTM. Frequency domain analysis can be employed to check the error tolerance level of OpenPBS FMU and VTM. The delay compensation and the coupling errors can be investigated and the energy-preserving concept, presented by W.Chen et al [25] can be tried to have a strongly coupled co-simulation system.
One of the reasons for using a open source vehicle model is to reduce the dependency of requiring SimScape license to simulate the model in Matlab. This could also be achieved by converting the respective blocks of VTM into S-functions or generating FMUs. However, minor corrections need to be carried out with the controller and/or the model in order to successfully add to the existing integration tool chain.
Furthermore, the investigation can be extended to check the controller for OpenPBS FMU plant system integration and check for redesign if necessary. Additionally, articulated vehicles can be tested with the path follower and for more critical maneuvers such as braking on a icy roads, emergency braking or Start/Stop in Uphill.

6.2.2 Synchronization Manager

Even though the execution cap of the VirtualBox is changed there will be a little delay still added to the execution. This can be reduced by writing on to the kernel, as VirtualBox is a hypervisor, it uses the RAM of the HOST system, the guest Virtual address on the Linux system will converted to guest physical address through address mapping and the guest physical address will be mapped to the host physical address through host virtual address with the help "shadow table mappings", as shown in Figure 6.1. Finding this shadow table is hard process, as it requires an external tool, like ArchC tool chain to determine the memory mapping between the guest physical and host virtual address, or the VmWare tool instead virtual box, or using a Linux as host, to have a PCI pass through method. PCI pass through methods is a support to the hypervisors where it behaves like the guest is attached directly to the host hardware [18]. The ROS core was extended to Windows operating system, which can be used to replace the ROS core running on Virtual machines to eliminate the Kernel level controlling problems and tested along with the synchronization manager.

It would also be interesting to implement and test the standardized protocol for co-simulation of the environments using the Distributed Co-Simulation Protocol(DCP) developed by Modelica association to integrate real-time systems, as suggested by *Martin Krammer et all* 26.

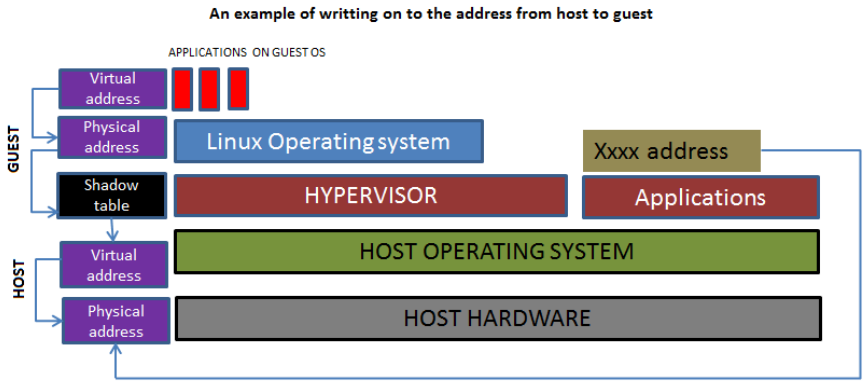

Figure 6.1: *Hypervisor Memory mapping route*

Bibliography

[1] Shota Tokunanga, Takuya Azumi, Co-Simulation Framework for Autonomous Driving Systems with MATLAB/SIMULINK.IEEE Real-time and Embedded technology and applications symposium. Pittsburgh,PA,USA.2017

[2] Fabio Cremona, Marten Lohstroh, David Broman, Edward A.Lee, Michael Masin, Stavros Tripakis, Hybrid co-simulation: it's about time, Software & Systems Modeling, Springer Berlin Heidelberg, page 1-25 ,1619-1366

[3] Siniša Slavnić, Adrian Leu, Danijela Ristić-Durrant and Axel Gräser, Modelling and Simulation of Human Walking With Wearable Powered Assisting Device, ASME VEHICLES AND HUMAND ROBOTICS PROCEEDINGS, Paper No. DSCC2013-4049.2103

[4] Stephen Boyd, Timekeeping in the linux kernel, Qualcomm Innovation Centre,Inc. Presentation On Open IoT and ELC, Embedded Linux Conference February 2017, Portland, USA.2017

[5] Mathworks Website, Real-Time UDP receive block Help., https://www.mathworks.com/help/instrument/udpreceive.html

[6] Peter Norton and Arthur Griffith, Peter Norton's Complete Guide to Linux, Page 243-283, ISBN : 0-672-31573-4, 2000.

[7] Evin Nemeth et all, Unix and linux system administration hanbook, 4th edition, page 120-140,283-292, ISBN : 013148005-7, 2010.

[8] Robert Love, Linux Kernel Development, A practical guide to the design and implementation of linux kernel, page 23-40,41-67,113-131,207-230 ,ISBN : 067232512-8,2004.

[9] Andy Oram and Ellen Siever, Linux Device Drivers, First Edition, page 131-150 and 178-212.ISBN: 156592292-1,1998.

[10] O'Connor, Patrick, and Andre Kleyner, Practical Reliability Engineering, 5th ed., Hoboken, N.J.: John Wiley & Sons, 2012.

[11] Kalra, Nidhi and Susan M. Paddock, Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability. Santa Monica,CA:RAND Corporation, 2016. https://www.rand.org/pubs/research_reports/RR1478.html.

[12] ISO Standards, ISO:26262:2011 Road vehicles - Functional safety, 2011.

[13] Sagar Behere , Martin Törngren, A Functional Architecture for Autonomous Driving, Proceedings of the First International Workshop on Automotive Software Architecture, May 04-04, 2015, Montréal, QC, Canada

[14] John Aurell, The influence of warp compliance on the handling and stability of heavy commercial vehicles, International Symposium on Advanced vehicle control, 2002, Tokyo, (Society of Automotive Engineers, Japan).

[15] Mathworks Webiste, Algebraic Loops Help, https://www.mathworks.com/help/simulink/ug/algebraic-loops.html

[16] Henrik Kaijser, Henrik Lönn, Peter Thorngren, Johan Ekberg, Maria Henningsson, Mats Larsson (2018). Towards Simulation-Based Verification for Continuous Integration and Delivery. ERTS 2018 - 9th European Congress on Embedded Real Time Software and Systems, Toulouse

[17] Bengt Jacobson (2012), Vehicle Dynamics – Compendium for Course MMF062, Chalmers University of Technology, Göteborg, Sweden.

[18] Vmware ESXi and vCenter Server 5.1 Documentation,

[19] Peter Nilsson. Traffic situation management for driving automation of articulated heavy road transports. Thesis for the degree of doctor of philosophy in machine and vehicle systems, 2017.

[20] Vinnova, Performance Based Standards for High Capacity Transports in Sweden, 2013-2017.

[21] https://github.com/performance-based-standards/OpenPBS.

[22] P. Sundström, B. Jacobson and L. Laine, “Vectorized single-track model in Modelica for articulated vehicles with arbitrary number of units and axles,” in Modelica conference 2014, March 10-12, 2014, Lund, Sweden, 2014.

[23] Bengt Jacobson et al., An Open Assessment Tool for Performance Based Standards of Long Combination Vehicles, Research report, Gothenburg, Sweden: Chalmers University of Technology, 2017.

[24] Canhui Wu, Co-simulation Methods for EPAS and Chassis Systems Development.(2018) Göteborg : Chalmers University of Technology

[25] W.Chen et al., Real-time Co-simulation Method Study for Vehicle Steering and Chassis System, IFAC PapersOnLine, Volume 51, Issue 9, 2018, Pages 273-278

[26] Martin Krammer, Klaus Schuch, Christian Kater, Khaled Alekeish, Torsten Blochwitz, Stefan Materne, Andreas Soppa, Martin Benedikt., Standardized Integration of Real-Time and Non-Real-Time Systems: The Distributed Co-Simulation Protocol, Proceedings of the 13th International Modelica Conference, March 4-6, 2019, Regensburg, Germany 10.3384/ecp1915787.

A

Appendix

The detailed script of the synchronization manager is as follows

```
import socket
import time
import subprocess
import win32api
import os
import time

#--------------TCP and UDP socket config-----
 #UDP SOCKET
matlab_sock_send = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,0)
matlab_sock_recv = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,0) #UDP SOCKET
adapt_sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM,0)#TCP SOCKET
#-------------function start ------

def conv (data) :#convert string to 8 bit unsigned integer.
        converted_data = bytes(data, 'utf-8')
        return converted_data

#matalb recieving function
def matlabrecv (port_recv,run) :
#port_recv is the matalab configure port as in *1*
        port = port_recv
        #"run" states wether the system is at start or running
        if run == 0 :
        #to connect the socket for the first time alone/ during start -----
                matlab_sock_recv.bind((IP, port))
                #Ip is the local host
        else :
                pass
        data,addr = matlab_sock_recv.recvfrom(buffer)
        #recieve the okay signal or any data that matlab is completed
        return data

#matalb sending function
def matlabsend(step_time,port_send) :
        data ='Step '+step_time+'\n'
```

```
        message = conv(data)
        matlab_sock_send.sendto(message,(IP,port_send))
        return 1

#adapt sending and recieving function
def adapt_ctrl (step_time,port,run) :
#port_recv is the matalab configure port as in *2*
        data ='Step '+step_time+'\n'
        message = conv(data)
        if run == 0 :
        #to connect the socket for the first time alone/ during start
                adapt_sock.connect ((IP,port))
        else :
                pass
        adapt_sock.sendall(message)
        data = adapt_sock.recv(buffer)
        return data

#linux execution control
#NOTE : COMPUTER NAME AND EXE_SPEED SHOULD BE GIVEN IN STRING FORMAT, '
ROS','50'

def exe_cap (computer_name,exe_speed) :
#linux execution speed changed EXE_SPEED CAN BE ISSUED FROM 1-100
        file_name = "C:\Program Files\Oracle\Virtualbox\VBoxManage.exe modifyvm"
        #this is an API call which can be called in in command
        #shell likewise VBoxManage.exe
        # --executioncap calls the cpu execution speed command in VB
        exe_cmd = file_name+"\t"+computer_name+"\t"+"..
        ..--cpuexecutioncap"+ "\t"+exe_speed
        print("EXECUTION SPEED is set to :",exe_speed)
        subprocess.call(exe_cmd)
        cpu = win32api.GetLastError ()
        if cpu == 0 :
                return 0
        else :
                return 1
#time value calculation for set execution cap
                #--------------------#
                #| EXE. CAP|  TIME  |
                #--------------------#
                #|100      | 1*TIME  |
                #--------------------#
                #|75       | 1.5*TIME|
                #--------------------#
                #|50       | 2*TIME  |
                #--------------------#
```

```
                #|25        | 2.5*TIME|
                #--------------------#

def run_time_calc(time_value,exe_speed):
        if exe_speed == 100:
                time = time_value*1
        elif exe_speed == 75:
                time = time_value*1.5
        elif exe_speed == 50:
                time = time_value*2
        elif exe_speed == 25:
                time = time_value*2.5
        else:
                print("enter values either 100,75,50,25")
        return time

# execution speed confirmation
def enter_exe_input() :
        Y = [100,75,50,25]
        while True:
                try:
                        exe = int(input('EXECUTION SPEED FOR VB: '))
                        assert exe in Y
                        return exe
                except ValueError:
                        print("Not a valid number! Please enter an number.")
                except AssertionError:
                        print("Please enter any value in [100,75,50,25]")
                else:
                        break

def start_vb(computer_name) :
        file_name = "C:\Program Files\Oracle\Virtualbox\VBoxManage.exe startvm"
        #this is an API call which can be called in in command shell
        #likewise VBoxManage.exe
        exe_cmd = file_name+"\t"+computer_name
        # startvm is to issue power on command on to VB
        print("started",computer_name)
        subprocess.call(exe_cmd)
        while True:
                try:
                        exe = input('ready? if yes enter y: ')
                        assert exe == 'y'
                        return 1
                except AssertionError:
                        print("waiting state,when ready enter 'y'")
```

```
                else:
                        break

#linux control function
#NOTE : COMPUTER NAME SHOULD BE GIVEN IN STRING FORMAT
def controlvb(computer_name,time_value,run,exe_speed) :
#linux/virtualbox controlling function
        file_name = "C:\Program Files\Oracle\Virtualbox\VBoxManage.exe controlvm"
        #this is an API call which can be called..
        #...in command shell likewise VBoxManage.exe
        pause_cmd = file_name+"\t"+computer_name+"\t"+"pause"
        #controlvm controls the vm for power..
        #..off/pause/resume/sleep/save state etc.,
        resume_cmd = file_name+"\t"+computer_name+"\t"+"resume"
        if run == 0:
        #this run is similar to cases used in previous..
        #..function -- TO PAUSE FOR THE INITAL RUN
                subprocess.call(pause_cmd)
                #pause the virtualbox
                return 1
                #return a value to make sure the system is..
                #...paused,this can also be obtained..
                #..from the windows error message by issuing
                #..another pause command
        elif run == 1 :
        #start the virtualbox by resume command
                start_time = time.time()
                subprocess.call(resume_cmd)
                #this is made to sleep for 0.001(1ms)..
                #..times the time values,obtained from..
                #..run_time_calc,time.sleep(0.01*time_value)..
                #... inorder to make the system run for only 10ms..
                #..or certain millisecond value...
                subprocess.call(pause_cmd)
                #... so the loop is configured to run for ..
                #..10m, once started it will run
                end_time = time.time()
                print(start_time)
                print(end_time)
                print("%s seconds" % (time.time() - start_time))
                return 1        #.. and sleep after 10ms.


#--------decelaration-------------------------------
```

```
IP = "127.0.0.1"        #local host ip same for all system
buffer = 1024         #buffer rate to send and ..
#..recive data for tcp and udp protocols
matlab_port_send = 10002
#matlab send port
matlab_port_recv = 10003        #matlab recieve port
adapt_port = 11200        #adapt sending and recieving port
step_time = '10'        #counts to 0.01s to matlab...
#..NOTE ALWAYS IN STRING FORMAT AS TCP AND UDP .
step_time_int = 10        #set to run the linux control
#..loop for 10ms - so if the exe speed is set to 100% then..
#..ACCEPTS STRING DATA VB runs for 10ms,
#else for more time according to the table above
total_simulation_time = 10
#total sim time,1 loop counts for 0.1 steps in matlab so,
#to run matlab for 1 matlab..
#..simulation time, total_sim_time will be 10
exe_speed = enter_exe_input()
#user input for exe speed
computer_name = 'ros'
#computer name to be controlled in VB
start_virtualbox = start_vb(computer_name)
#Start the vb as execution ..
#..speed is changed when machine is in power off state
step_time_VB = run_time_calc(step_time_int,exe_speed)
#to calculate the running time of vb after the exe is changed.
#-------loop variables------------

loop = True
X = 0
time_first = total_simulation_time
init = 0
run = 1
```