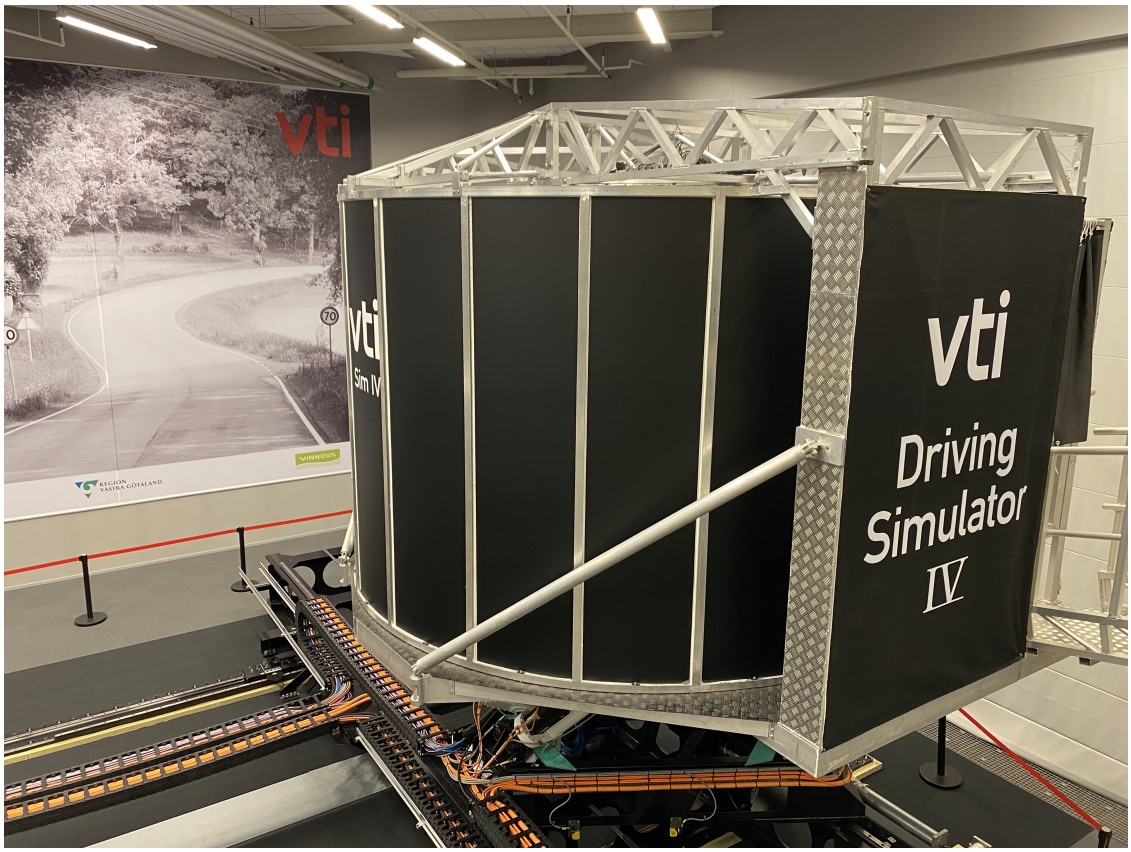
CHALMERS
UNIVERSITY OF TECHNOLOGY



Implementation of a latency controller in an 8-DOF driving simulator.

A Latency controller based on the Otto-Smith predictor, with a model developed using System Identification.

Master's thesis in Master Programme Systems Control and Mechatronics

Alexander Hägglund

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Implementation of a latency controller in an 8-DOF driving simulator.

Alexander Hägglund

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2023

Implementation of a latency controller in an 8-DOF driving simulator.
A Latency controller based on the Otto-Smith predictor, with a model developed using System Identification.
Alexander Hägglund

Supervisor: Smit Saparia & Sogol Kharrazi, the Swedish National Road and Transport Research Institute
Examiner: Fredrik Bruzelius, Department of Mechanics and Maritime Sciences

Master's Thesis 2023
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Overview of SimIV at the Swedish National Road and Transport Research Institute, picture taken from the control room.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Implementation of a latency controller in an 8-DOF driving simulator.
A Latency controller based on the Otto-Smith predictor, with a model developed using System Identification.
Alexander Hägglund
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

In all driving simulators delays will be present. These delays are in all parts of the system. Delays in the driving simulator have according to previous studies both effects on the driving performance and the risk of experience *simulator sickness*. This thesis focused on the acceleration in the Motion System, and no other delays were investigated. The aim was to investigate the accelerations in Sim IV with an IMU and use the IMU-data to develop a dynamic model of the longitudinal dynamics of the XY-table. This model should then be used to better control the acceleration against the Motion Cueing reference. The controller is partially an Otto-Smith predictor. The result of this thesis is a lowered acceleration latency with both synthetic and driving data as input. More tests with actual driving need to be done for evaluating if the controller has an impact on the driver's experience. This controller was implemented for longitudinal dynamics with the XY-table, in the future it could be implemented for the other seven degrees of freedom.

Keywords: Driving Simulator, Latency, Control system, Smith-Predictor, System Identification, IMU, Accelerometer, Raspberry Pi

Acknowledgements

I would like to thank my supervisors at the Swedish National Road and Transport Research Institute(VTI) Smit Saparia and Sogol Kharrazi. At VTI I would also like to thank all the other people that have helped me with my thesis, for both technical advice and for welcoming me to the office. Special thanks to Isabella Sanchez and Isabella Nilsen that also wrote their thesis at VTI for all the interesting discussions and advice.
In addition to the people at VTI, I would also like to thank my examiner at Chalmers Fredrik Bruzelius for all the advice. In addition to my examiner, I would also like to thank Associate Prof. Jonas Bärgman and Dr. Tina Mayberry for a good course in academic writing. Lastly, I want to thank my friend Melker Wikström for helping me with the language in the report.

Alexander Hägglund, Gothenburg, 011-2023

Contents

List of Figures

List of Tables

1

Introduction

A driving simulator is a tool to make it possible for a driver to operate a virtual car. For example, this tool can be used in car development, driver training, or in studies of driver behaviors. At the Swedish National Road and Transport Research Institute(VTI) the simulator in Figure 1.1 is named Sim IV, and is used for road safety research, vehicle dynamics, and external projects.

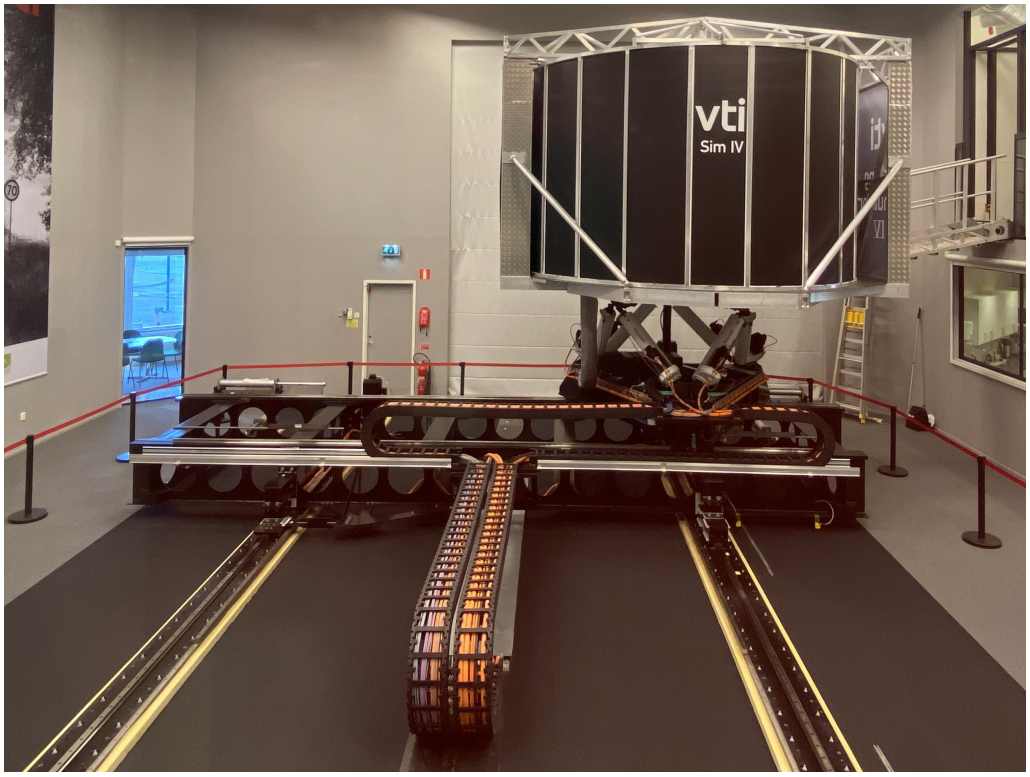


Figure 1.1: Sim IV at the Swedish National Road and Transport Research Institute

The basic principle for a driver in the loop simulator is to give a human the possibility to operate a virtual vehicle. This is facilitated by the combination of a virtual car, often called *vehicle model* (VM), a *visual system* (VS), and some sort of driving controls for the driver. At Sim IV, the front half of a Volvo XC60 is mounted, and the driver control inputs are connected to the corresponding ones of the Volvo XC60. In addition to the Volvo XC60, a Volvo FH truck cabin, and a pedaling bike can also be used. For all of this project, the Volvo XC60 was mounted. The VS of Sim

IV consists of a curved screen of 210 degrees with nine projectors. The car and VS are shown in Figure 1.2.

Figure 1.2: Inside the Sim IV "dome"

1.1 Background

The background for this thesis is the knowledge that all mechatronic systems have dynamic behavior that will result in some delay for the requested input. In addition to the dynamics, the system will also have a fixed time delay due to communication.

All delays in a driving simulator will make the driving experience worse, impact the performance of the driver, and could also result in a higher likelihood of simulator sickness, which is a variant of motion sickness. In addition to the potential gains of reducing delays, no analyses of the Sim IV feedback signal have been performed.

1.2 Purpose

Given the information presented previously in Chapter 1, the purpose of this thesis is to enhance the driving experience by reducing the response delay. Reducing the delays is done by implementing a *prediction algorithm* (PA) based on a dynamic model, developed using system identification. Reducing the response delay will hopefully give the driver a better "feel" of the car, and could also have the benefit of reducing the risk of "simulator-sickness".

1.3 Limitations

This project will focus on the longitudinal dynamics using the **X-Y**-table of Sim IV. Starting with longitudinal dynamics makes for a good evaluation of the method, and if it works. However, if the method turns out successful applying the same principle to lateral dynamics could be possible.

The dynamical model of the motion-platform of *Sim IV* should be developed using System Identification. This thesis will only focus on delays in the Motion System (MS) and not investigate the delays in the VS, even if it is acknowledged that delays in VS have an impact on driving performance and simulator sickness.

1.4 Specifications of the issue under investigation

This project aims to answer three main questions about the driving simulator Sim IV at VTI

- Is the existing feedback a good data source when doing experiments for System Identification or is better to use an IMU?
- How well can a 1st-order model represent the dynamics of the Sim IV XY-table in longitudinal dynamics?
- Could this 1st-order model be used to lower the acceleration latency?

1.5 Ehical aspects - Ecological and social impact

During the execution of the thesis, one ethical risk is identified, the risk is if a simulator study is used as a validation test. The issue is that the test could have the risk of exposing people to motion-sickness. It should not be a problem, since the exposure is no more than in other simulator studies done at VTI. To make sure this is the case, the experiment will be checked with the *ethical commite* at VTI if driving tests are to be performed. In the end, no driving was performed, and this ethical issue was not investigated deeper.

This thesis aims to achieve an overall better experience in a driving simulator. This could result in a larger proportion of studies being performed in a driving simulator, and therefore reduce the number of kilometers driven with test cars. This will have a positive ecological impact.

The social impact could come from a reduction of simulator sickness, and therefore the possibility for more people to participate in simulator studies and give a more representative result.

2

Theory

This theory section aims to provide the reader with the necessary background for understanding and review of this thesis. It will explain the background of how the used methods work, and different products work. The different subjects covered will be driving simulators, IMU sensors, Data logging, System Identification, and control design. The covered subjects are explained making the reading easier.

2.1 Driving simulators

As described in Chapter 1 a DIL simulator is a tool for driving a virtual vehicle. In this project, the vehicle will be a road car, and the DIL will be a *driving simulator*. A motion driving simulator is usually classified by the number of degrees of freedom (*DOF*) they can represent. From a simple stationary simulator *0-DOF* up to a simulator capable of representing motion in *6-DOF*. A *6-DOF* simulator can represent both translations and rotation around/along **X, Y, Z**-Axis. This is usually done by using a hexapod also called a *Stewart platform*, with six linear actuators moving a platform on which the driver of the vehicle sits [1].

To further enhance the virtual driving experience or for specific types of tests, some driving simulators have different types of actuators. This could be vibration or a longer linear stroke. Examples of simulators with a linear motion envelope of $\pm 17.5m$ X-direction and $\pm 10m$ Y-direction and vibration in *3-DOF's* exist. [1].

2.1.1 Sim IV

Sim IV at VTI has a conventional hexapod that gives *6-DOF*. In addition to the hexapod in Figure 2.1, Sim IV also has a **X-Y**-table on which the hexapod is mounted.

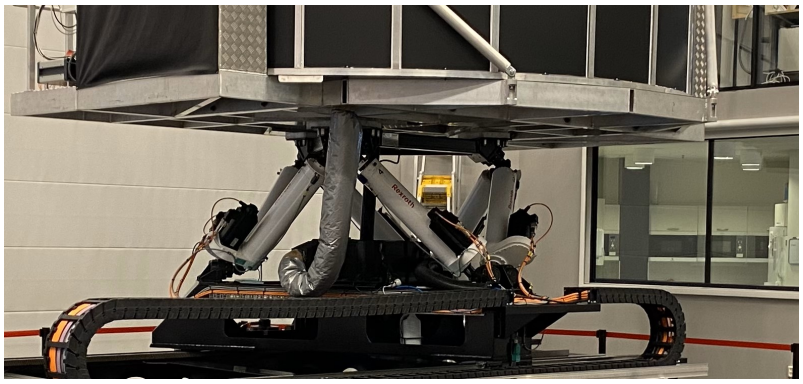
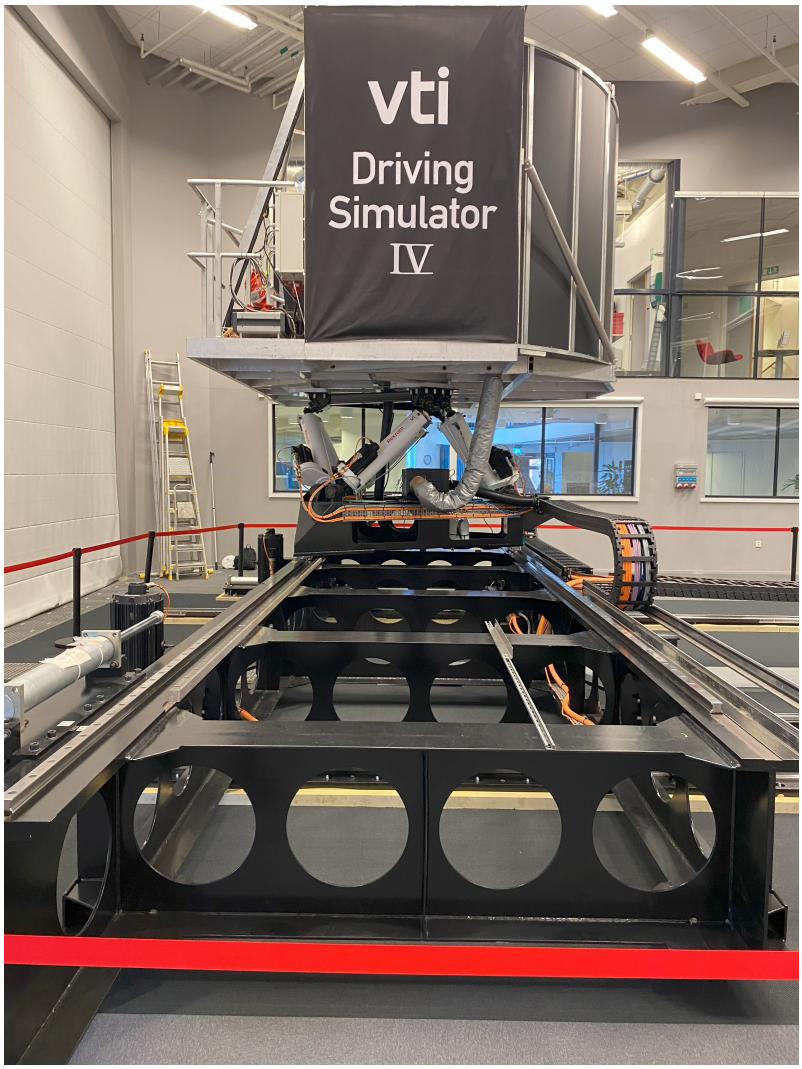


Figure 2.1: Hexapod of Sim IV.

In Figure 2.2(a)-2.2(b) the **X-Y**-table is shown, this is linear rails, on which the simulator can move. This makes it possible for the simulator to represent larger strokes along the **X-Y**-axis than possible only with the hexapod. To state that Sim IV has more actuators than a normal *6-DOF* simulator, it is called a *8-DOF* simulator, even if the maximum combined degree of freedom is six [2]. The two extra *DOF's* make the planar linear motion have two redundant actuators, and can represent different frequencies with different actuators. The MC algorithm in SimIV is based on the classical *washout principle* [2]. The basic specifications of Sim IV are specified in Table 2.1.

Table 2.1: The Specifications of the Sim IV Motion System.

Motion	Acceleration	Velocity	Position
	$([g])[m/s^2]$	$[m/s]$	$[m]$
Surge	±(0.65 g)/6.374	±0.8	+0.408 -0.307
Sway	±(0.60 g)/5.884	±0.8	+0.318 -0.318
Heave	±(0.60 g)/5.884	±0.6	+0.261 -0.240
	$[deg/s^2]$	$[deg/s]$	$[deg]$
Roll	±275	±40.0	+16.5 -16.5
Pitch	±275	±40.0	+15.5 -16.0
Yaw	±350	±50.0	+20.5 -20.5
	$([g])[m/s^2]$	$[m/s]$	$[m]$
X	±(0.50 g) 4.903	±2.0	±2.5
Y	±(0.50 g) 4.903	±3.0	±2.3



(a) The Y rails of Sim IV

(b) The X rails of Sim IV

Figure 2.2: The XY-Sleed of Sim IV.

2.1.1.1 Motion Cueing

Motion Cueing (MC) is the algorithm that converts the accelerations in the Vehicle model (VM) to inputs for the simulator. This is done by MC splitting the acceleration of the VM to the different actuators for letting to driver experience a goal acceleration. The acceleration could be split based on for example frequency of what type of acceleration it is. Different types of acceleration could be lateral acceleration due to cornering or changing lanes on a highway. The MC in Sim IV is based on the classic washout algorithm, adjusted to an 8 DOF driving simulator. [2]

The MC is a Simulink file that contains the filters and controller that split the accelerations. The MC-file is compiled to *C++* -code for use in the simulator.

2.1.1.2 Main loop in Sim IV

In Figure 2.3 the main simulation loop is visualized in a simplified block diagram. The simulation is controlled on the Sim IV kernel computer, all the signals from the VM are sent to the Kernel for processing. As seen in Figure 2.3, some blocks are in blue and some are in white. This is to clearly visualize what blocks are included in this project. The boxes in blue are not part of this project, the boxes in white are included.

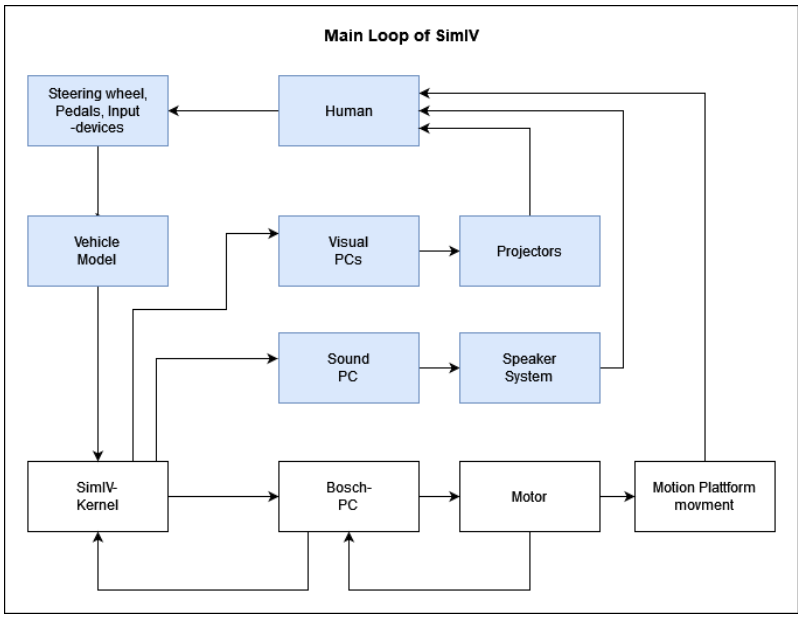


Figure 2.3: Main Loop of Sim IV.

The Sim IV-kernel is processing the signals and sends the right command to the VS-PC and the sound system PC. These separate systems are then computing the corresponding output to the projectors and speakers. The Sim IV-kernel is running the MC algorithm, which computes what motion command should be sent to the MS. The motion commands are *acceleration, velocity, and position* for all $8-DOF's$. These signals are sent to the *Bosch motion PC*, and then to the low-level actuator control, that moves the simulator.

After the signals are sent from the Sim IV-kernel to the Bosch-PC, the operator has insight into what signals are sent. All low-level control is handled by Bosch software that is not accessible by the user.

As shown in Figure 2.3, the human driver is included. The reason to include the driver is that the audio, visual, and motion feedback is used by driver to make the driving decisions in closed-loop together with the simulator.

Then using Sim IV in this thesis the signals sent to the Bosch PC will be referred to as *command* (CMD), and the signals the Bosch Pc returns to the Sim IV kernel will be named *feedback* (fdbk).

2.1.2 Latencies in driving simulators

A delay in a mechatronics system is defined as the time it takes between a requested input, to the system response with that input. This is a phenomenon that is impossible to fully eliminate, but it is possible to reduce the delay by different types of control algorithms. The delay for a motion simulator is often classified into different types according to Fang [3]. The first delay is caused by the sampling time of the discrete system, and inputs can occur at any continuous time. This delay has a mean value of $\frac{t_s}{2}$ with a max delay of t_s stated by Guo [4]. Next delay is a constant overhead that is caused by the communication, computation, and other fixed times that occur every time step [3].

The last part of delays is the actuation of the *motion-system* (MS), the "dynamic response delay" depends on the performance of the MS according to Lee [5]. In other driving simulators with a similar MS, this delay has been measured to be up to 200 ms at some frequencies. This delay also has the characteristic of a "phase shift/delay" of the input signal [3].

Then using a driving simulator, it is common to experience a symptom called "simulator-sickness". This symptom often results in the following effects, blech-ing, flatulence, and cold-sweating. According to Oman [6] delay time could increase the risk of "simulator-sickness" for the driver.

A large delay in either MS or VS reduces the accuracy of the driver's car control, in the study by Hogema this is exemplified with a *double lane change* (DLC). The result clearly shows that delay results in larger steering angles, and an overshoot in the heading. The paper also demonstrates that a prediction could have positive impacts on the driver's inputs [7].

2.2 Inertial Measurement Unit - IMU

In this project, one possibility is to use an IMU for collecting the data for System Identification. An Inertial Measurement Unit (IMU) is an electronic device that contains several sensors for measuring the physical state of the object it is attached to. Typical sensors that are included in an IMU are a three-axis gyroscope and a three-axis accelerometer. Additional sensors that could be included are a three-axis

magnetometer and a barometer. If all these sensors are included, it is called a 10-DOF IMU. In this project, a 9-DOF IMU from Adafruit is used. This IMU has an accelerometer, gyroscope, and magnetometer, but lacks a barometer [8].

This project will only make use of the three-axis accelerometer in the IMU, and therefore is this the only sensor described in depth below.

2.2.1 Accelerometer

An accelerometer is one of the sensors usually included in an IMU, often three accelerometers are placed orthogonally. The three accelerometers are necessary for measuring the acceleration in a $3D$-space. The type of accelerometer that is implemented in this thesis is of the *Microelectromechanical systems* (MEMS) type. That is a small sensor type that is often used in inexpensive measurement devices. The principle of a MEMS-accelerometer is shown in Figure 2.4.

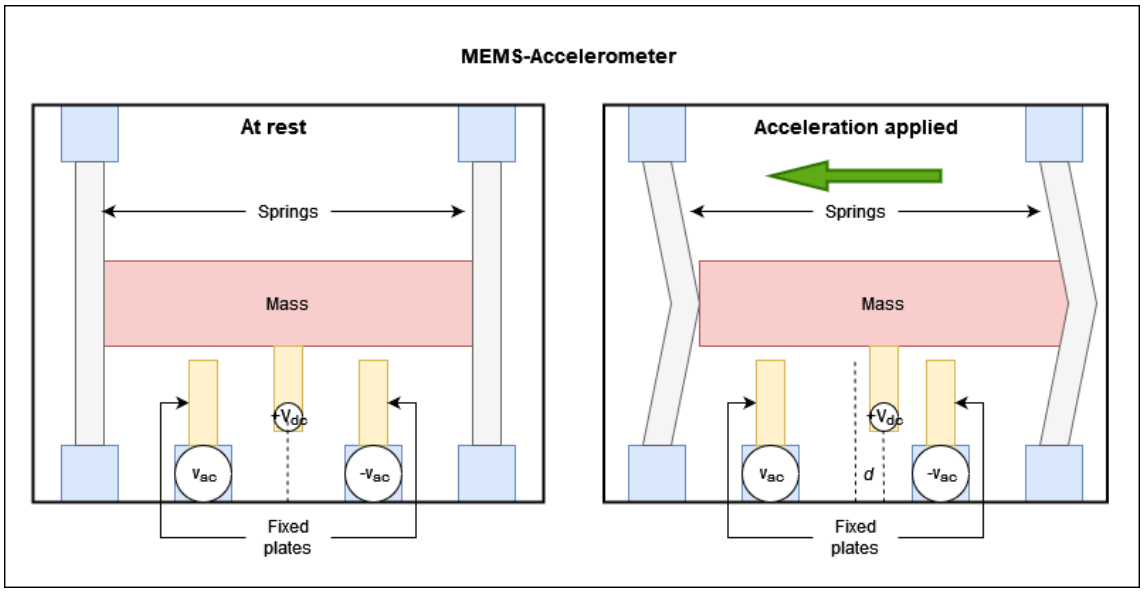


Figure 2.4: schematic explanation of a MEMS accelerometer, with one set of *fixed plates* visualized.

As seen, the accelerometer has a small mass that is suspended with springs, and fixed plates and plates connected to the mass. In Figure 2.4 the mass is shown both at rest and then an acceleration is applied. As stated by Sinha an *Alternating current*(AC) is applied, and to the plate connected to the mass an *Direct Current*(DC) is applied [9].

Then acceleration is applied, and the mass is moving in the opposite direction. For calculating the acceleration, the different displacement d is changing the capacitances between each of the fixed plates and the moving plate. This change in capacitances is then calibrated to a specific acceleration [9].

2.2.2 Calibration of accelerometer

For calibration of an IMU using gravity, the method is based on equation Equation 2.1. This is that then the IMU is stationary the sum of all three accelerometers is $1g$ [10].

$$1g = \sqrt{(a_x^2 + a_y^2 + a_z^2)} \tag{2.1}$$

Using this equation, and multiple measurements, it is possible to find calibration values for both the bias and misalignment of the accelerometers. According to Hassan, the error model for the sensors is presented in Equation 2.4. In this paper, the two compensation arrays are presented in Equation 2.2 and Equation 2.3. In the equations, A is the rotation matrix that compensates for the misalignment and b is the vector compensating for the bias in the measurements [11].

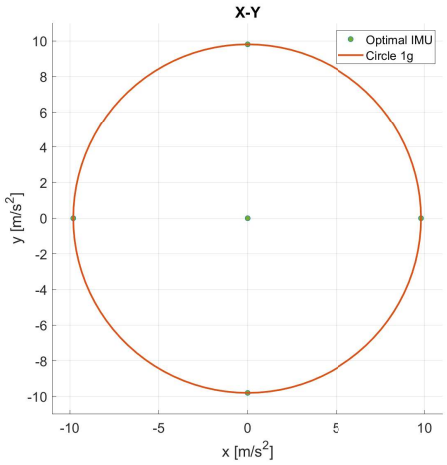
$$A = \begin{bmatrix} s_x & s_{xy} & s_{xz} \\ s_{yx} & s_y & s_{yz} \\ s_{zx} & s_{zy} & s_z \end{bmatrix} \tag{2.2}$$

$$B = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \tag{2.3}$$

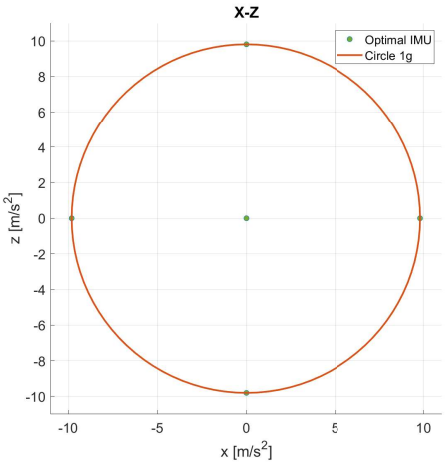
Then the symmetry condition $s_{xy} = s_{yx}\ s_{xz} = s_{zx}\ s_{yz} = s_{zy}$ explained by Hassan is applied, and the full equation is the following [11].

$$\begin{bmatrix} ax_{cal} \\ ay_{cal} \\ az_{cal} \end{bmatrix} = \begin{bmatrix} s_x & s_{xy} & s_{xz} \\ s_{yx} & s_y & s_{yz} \\ s_{zx} & s_{zy} & s_z \end{bmatrix} \left(\begin{bmatrix} ax \\ ay \\ az \end{bmatrix} - \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \right) \tag{2.4}$$

After the calibration, the IMU readings should, be at $\pm 1g$ and located along the three axes, shown in 2.5(a)-2.5(b). This and the method for determining the A, B arrays are presented in Subsection 4.1.4.



(a) Showing the $X - Y$ readings of an optimal IMU.



(b) Showing the $X - Z$ readings of an optimal IMU.

Figure 2.5: The readings of a theoretical optimal IMU.

2.3 Network communication and data logging

All the communication between different computers in this project is done through the *simulator subnet* for Sim IV. The subnet is handling all communication between computers when running simulations in Sim IV. Then running simulations, the network is disconnected from the internet and the rest of the network at the VTI office. The two protocols used for communication are *Secure Copy Protocol* (SCP) and *Secure Shell* (SSH).

To remotely control other computers during experiments SSH is used. SSH is a protocol for secure remote control of other computers over the network. The SSH protocol is as described by Ylonen, and can provide an encrypted link with user authentication. When connecting to a remote computer the following command is used, *ssh user@<ip-adress>*. This command will take the user to an authentication page for the remote computer. This experiment was run with Linux computers and then using SSH, the user gets access to the command line on the remote Computer. This makes it possible to execute all commands possible from the command line [12].

In addition to SSH, the file transfer protocol SCP is used. SCP is a protocol for transferring files between computers over the network. SCP is based on the *file transfer protocol* (FTP) according to Wilson. The difference between FTP and SCP is that SCP adds security through an authentication layer similar to the SSH protocol. Using SCP makes it possible to transfer files and directories both to and from a remote computer [13].

Both the SSH and SCP protocol in this project is run from Python to automate all the commands needed for running the tests. The SCP implementation in Python is done by the *Paramiko package*. The Paramiko package is a package that implements the SSH protocol as a Python API. All the SSH functionality is available from the Python script to be used for automating the experiments. The *SCP package* is the SCP implementation in Python that relies on the *Paramiko package* for authentication and acts as a submodule. Similar to the SSH implementation, the SCP provides the option to use Python for automating the file transfers in scripts [14] [15].

2.4 System Identification

A mathematical model over a dynamic system is a model meant to represent the relations between different variables and states of a system. Mathematical models are often used for calculation, control design, or other cases when a user might want to study a system or a procces. For developing a mathematical model, two common methods are the following. The first of these is to analyze the actual system and manually write the equations for the relations. The second method is called *System-identification* (SI), and it is a method for developing a model from known data for input and output [16].

The goal of SI is to get a mathematical model that is a good representation of the observed system, even in cases where the system is initiated with different initial

conditions or excited by different inputs [17].

2.4.1 Procces for system identification

When planning to develop a model by system identification, the process can be called the ”System Identification Loop” as it is called in by Ljung. This name is highlighting to that system identification is an iterative process. The main step in SI is described in Subsection 2.4.1.1 and Subsection 2.4.1.3 [16].

2.4.1.1 Experiment Design

The first step in the SI-loop according to Ljung is to design and perform experiments on the system to collect data. The first step when designing the experiment is to figure out what part of the system should be identified. This decision will impact what sensors that should be used, and where these sensors should be placed. It will also impact what input that should be used to excite the system to expose the dynamics [16].

To design the input, one needs to check the constraints of what is allowed or possible to send as inputs to the system. Ljung describes that some common input signals could be, Filtered Gaussian White Noise(FGWN), Random Binary Signal (RBS), Sinusoids, or Swept-Sinusoids(often called Chirp) [16].

2.4.1.2 Model development

The next step according to Ljung is to process the collected data so that it can be used for model development and select the section(s) of the experiment that should be used. When preprocessing the data it may also need ”repairing” if parts are missing or are obviously wrong [16].

After preprocessing the data, the model development can start. For this part, Matlab System Identification Toolbox(MSIT) is often used. MSIT is according to Ljung capable of identifying many different model structures. In this project, MSIT will be used for estimating a first 1^{st}-order model [18].

As described by Ljung, for estimating a State-Space model and Transfer-Function of 1^{st}-order, the functions *ssest* and *tfest* is used. These functions will identify a model with specified order with a black box method. To solve the parameter estimations, MSIT solves the problem as a *Least-square-problem*(LSP) for coming up with the estimated model with the best fit compared to the I/O data. To evaluate the LSP, MSIT is using the mean squared error(MSE) from the simulated and actual output as the cost to minimize [18].

2.4.1.3 Model validation

The last step in the SI-loop is to validate how accurate the model is. According to Ljung this can be done in many ways. Examples of this are to compare the precited output to other I/O-tests of the system (called *cross-validation*). Other tests are to plot the poles and zeros of the model or analyze the model's frequency, impulse, or

step response. Another useful tool that Ljung points out is to compare the bode diagrams for the different models [18] [16].

To cross-validate the SI-model in MSIT the function *Compare* is used. This function uses the MSE, in a similar way to the identification functions. Cross-validation is a good method for comparing different models and testing which of the models is best representing the system. according to Ljung, the only drawback with using cross-validation is that data collected in Subsection 2.4.1.1 has to be put aside for validation and can not be used for estimating the model [18].

After validating the model, the results need to be compared to what the performance targets are set for the model. These performance targets could vary depending on the usage of the model. If the model performs in an acceptable way, the SI loop is finished. If the model is not meeting the performance targets, then the SI process is continuing. According to Ljung the user has the flexibility to choose from which step the next iteration should start, it could be to start collecting more data for validation or further development, designing a new experiment, choosing another model structure, or modifying the performance criteria [16].

2.5 Open Loop Controller

Open Loop Contol(OLC) is one of the simplest controllers, the structure is shown in Figure 2.6.

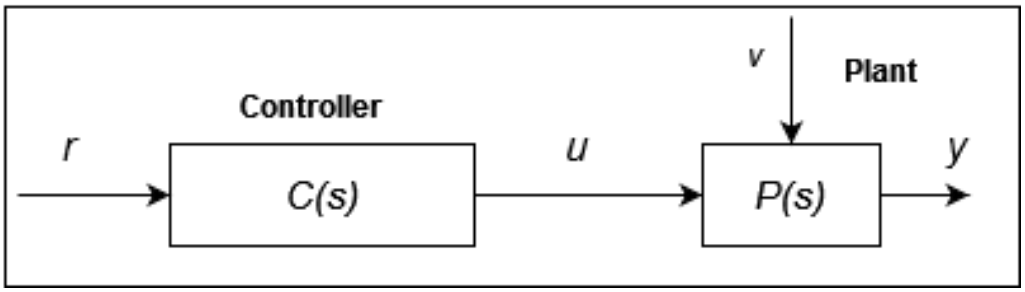


Figure 2.6: The structure open loop controller.

Some common principles for OLC are according to Lennartson time control, manual control, or feedforward control. Time control and manual control are very simple In time control the system is run according to a pre-determined schedule and for manual control, the user is controlling the system [19].

For feedforward control the existing knowledge for mapping a reference r to a control signal u is used. This is without taking the state of the plant P into consideration. The drawback with feedforward control is that a potential disturbance v will affect the system without the possibility for the controller to change the input to track the reference value.

2.6 Closed Loop Controller

A Closed Loop Controller(CLC) (shown in Figure 2.7 is a controller that has a feedback loop from the output state of the plant to calculate a control error $e = r(t) - y(t)$. The feedback loop is used for compensating disturbances v. By incorporating and compensating for the disturbances v, the CLC is solving the big problem with the OLC. [19].

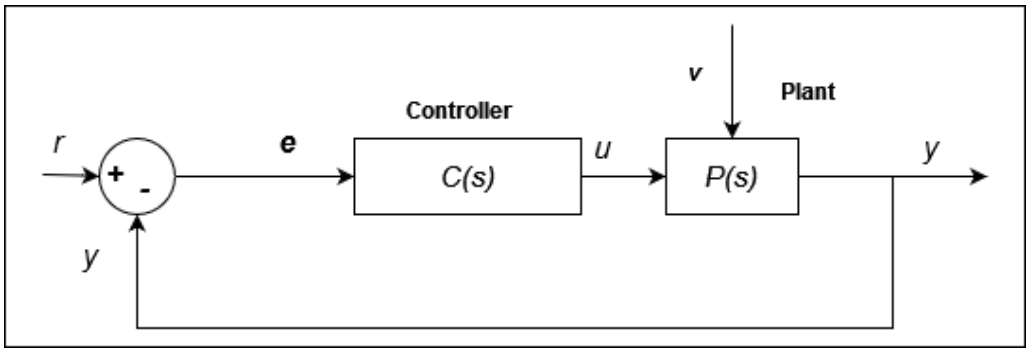


Figure 2.7: The structure Closed loop controller.

As explained by Lennartson, feedback has the drawback of introducing the risk of instability in the system. A well know example of instability by feedback is *acustic feedback* when a microphone and speaker are placed too close to each other [19].

2.6.0.1 PID - Controller

The *PID-conteller* is a common controller to use in a CLC. The PID is made up of three parts that are affecting the control signal in different ways, the parts are **P**roportional, **I**ntegral, and **D**erivative. The transfer function $C(s)$ for the PID-controller is derived to Equation 2.5 [19].

$$G(s) = K_p(1 + \frac{1}{T_i(s)} + T_d(s)) \tag{2.5}$$

Higher **proportional** action is usually affecting the system by increasing the response speed, lowering the stability margins, increasing the control signals, and better resistance against process disturbances. Lowering the proportional action will affect the system in the opposite direction [19].

Higher **integral** actions are increasing the compensation for low-frequency disturbances, and static accuracy, and are necessary for completely eliminating the control error e when tracking a fixed target. The integral part is negatively affecting the phase margin for the system [19].

Opposite to integral action, the **derivative** part is positively affecting the phase margin of the system. A higher phase margin is often resulting in better stability and lower damping. The other consequence of adding a derivative part is higher

gain at high-frequency disturbances. This higher gain could have the benefit of counteracting the lowered stability margins from an integral part [19].

It is essential to keep in mind that adjusting the proportional part also affects the other constants as described in Equation 2.5 [19].

2.6.1 Smith Predictor

A Smith-Predictor(SP) is a control method developed for controlling systems that has feedback with a long dead time delay as described by Recio in [20]. The structure of an SP is shown in Figure 2.8. In the figure it is shown that the SP uses both the delayed feedback and the estimated state from the model to create the closed loop.

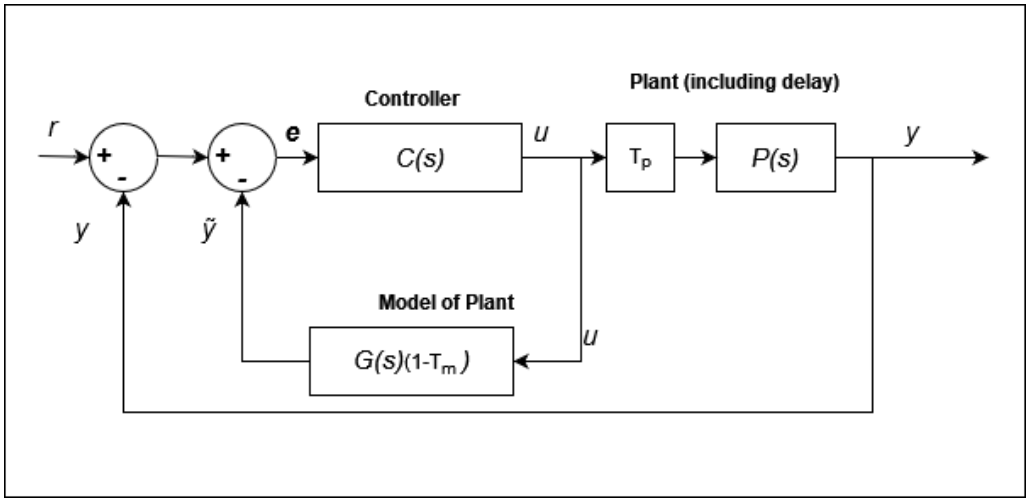


Figure 2.8: The structure of a Smith Predictor.

From the structure of the Smith-predictor in Figure 2.8, the transfer function from *reference* to *output* $\frac{Y(s)}{R(s)}$ is derived. The total transfer function is shown in Equation 2.6 [21].

In Equation 2.6, the variables are derived from Figure 2.8. That means $C(s)$ is the controller, $G(s)$ is the model, $P(s)$ is the plant, $T_(s)$ is the time dealy for the plant feedback and $T_m(s)$ is the time delay implemented in the model.

$$\frac{Y(s)}{R(s)} = \frac{C(s)P(s)Tp(s)}{1 + C(s)G(s) - C(s)G(s)T_m(s) + C(s)P(s)T_p(s)} \tag{2.6}$$

In the ideal case, with a perfect model and a delay estimation, *i.e* $G(s) = P(s)$ and $T_m = T_p$ the transfer function would be simplified to the equation shown in Equation 2.7.

$$\frac{Y(s)}{R(s)} = \frac{C(s)P(s)Tp(s)}{1 + C(s)G(s} \tag{2.7}$$

3

Pre-study, An evaluation of the accuracy of Sim IV feedback

To answer the first question in Section 1.4, a prestudy is done. This pre-study is to analyze existing data for tests done in Sim IV using an external IMU. The data used in this part of the project was collected by Arun Kumar, an Industrial Ph.D. student at Volvo Cars [22]. The experiment was carried out with a similar method as this project would use. By exciting the MS of Sim IV with acceleration inputs, and measuring the resulting accelerations of Sim IV with an accelerometer. By the end of this pre-study in Section 3.3, a decision is taken to either use the feedback data or an IMU for the SI experiments. The feedback data is the reported states by the bosch MS of SimIV.

3.1 Method

In Section 3.1 the method for the experiment done by Kumar and the tools used for evaluating the difference between the reported feedback and the external IMU are explained.

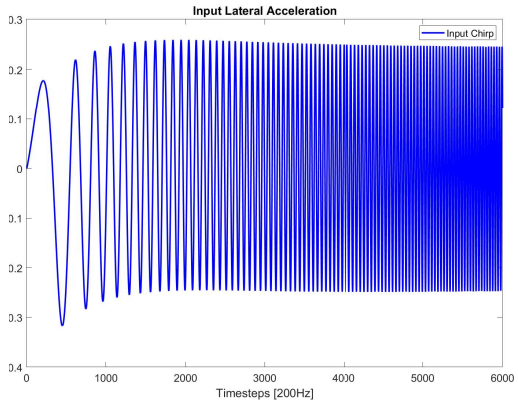
3.1.1 Method for collecting data

The setup used by Kumar during the experiment was a Dewesoft DS-IMU2, that was mounted to Sim IV shown in picture Figure 3.1. The Dewesoft IMU is a commercial IMU that has an accelerometer with a bandwidth of 400 Hz and a range of up to $\pm 16\, g$, these are specifications that are higher than the capabilities of Sim IV. This method makes it possible to measure all the acceleration Sim IV can produce. The DS-IMU2 also has many modes for synchronizing the time to external sources, in the case of the experiment the time was synced with the Sim IV kernel [23].

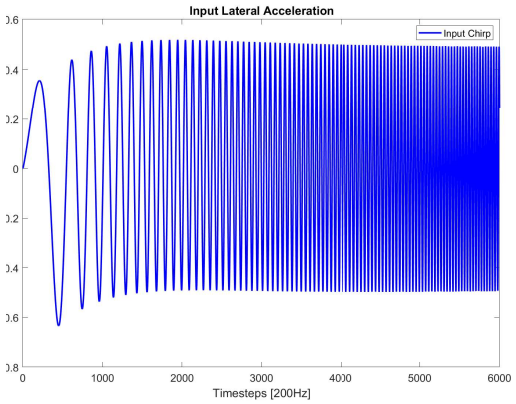
Figure 3.1: IMU placement used by Kumar [22].

The data were analyzed with the frequency of 200 Hz, this is to match the sample rate of the Sim IV kernel, the IMU data was recorded with a sampling rate of 500Hz.

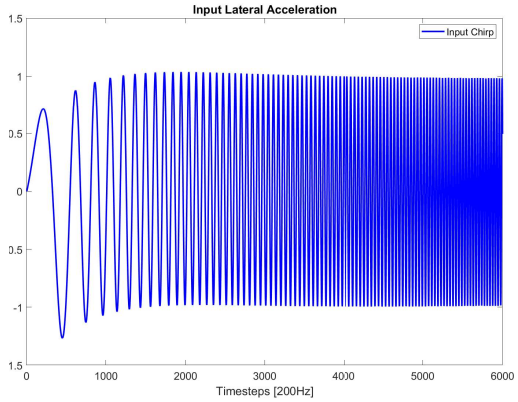
As input for the tests, Kumar used *Chirp-signals*. A chirp signal is a sine-wave with slowly increasing frequency over time. The input signals used by Kumar are shown in 3.2(a) to 3.2(d), with a frequency between $0-10\,Hz$. The amplitude are between $\pm\,0.25\,m/s^2\ -\ \pm\,1.5\,m/s^2$.



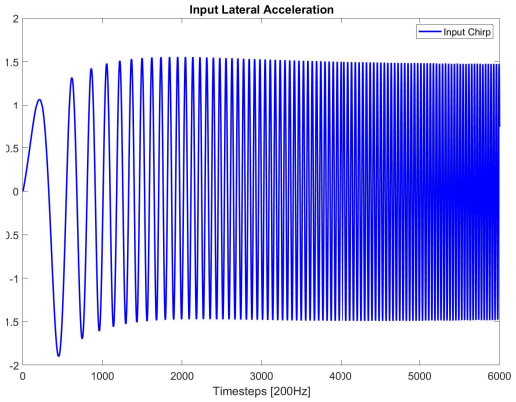
(a) Fq: $0 - 10Hz$, Amp: ± 0.25.



(b) Fq: $0 - 10Hz$, Amp: ± 0.5.



(c) Fq: $0 - 10Hz$, Amp: ± 1.



(d) Fq: $0 - 10Hz$, Amp: ± 1.5.

Figure 3.2: The four lateral acceleration chirp inputs used by Kumar [22].

The inputs to the motion system were sent in before the MC algorithm splits the acceleration to different DOF's. This resulted in the acceleration being split between the hexapod and the XY-sled in combination by the frequency split in the MC. This makes it impossible to identify the feedback for the hexapod and sled separately, it needs to be evaluated together.

3.1.2 Evaluation of data

The test data were analyzed first by calculating *RMSE-values* for the error between the feedbacks and IMU, and also by visualization using plots to see. For *RMSE*, four values were calculated using Equation 3.1.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \mid A - B \mid^2} \tag{3.1}$$

The four *RMSE* Calculated was with A and B equal to the vectors in Table 3.1.

Table 3.1: The different RMSE values.

A-variable	**B-variable**
Command Lateral Acceleration	Feedback Lateral Acceleration
Command Lateral Acceleration	IMU Lateral Acceleration
Feedback Lateral Acceleration	IMU Lateral Acceleration
IMU Lateral Acceleration	Feedback Lateral Acceleration

3.2 Results of data analyze

The result of analyzing the data collected by [22], is presented in Section 3.2. This result is only a pre-study and is not used in further in the system identification performed in this work.

3.2.1 RMSE Values

Table 3.2: RMSE values for the pre-study. Cmd is the acceleration command, Fdbk is the reported feedback from Sim IV and IMU is the measured acceleration from the accelerometer.

Test:	Amp	*Rmse Cmd-Fdbk, (a)*	*Rmse Cmd-Imu, (b)*	*Rmse Fdbk-Imu, (c)*
1	±0.25	0.2445	0.2365	0.2517
2	±0.5	0.3642	0.4751	0.3861
3	±1	0.6917	0.9416	0.7306
4	±1.5	1.0355	1.4035	1.0677

3.2.2 Figures

In addition to the *Rmse*-value, the data was also analyzed visually to check where the difference was, the graphs are shown in Figure 3.3-Figure 3.6. As seen in the plots, there is a clear difference between the reported feedback and what the DEWESoft IMU reads. This difference is also not consistent, depending on which amplitude (*i.e* test number), and frequency (*i.e* test time) the difference is changing.

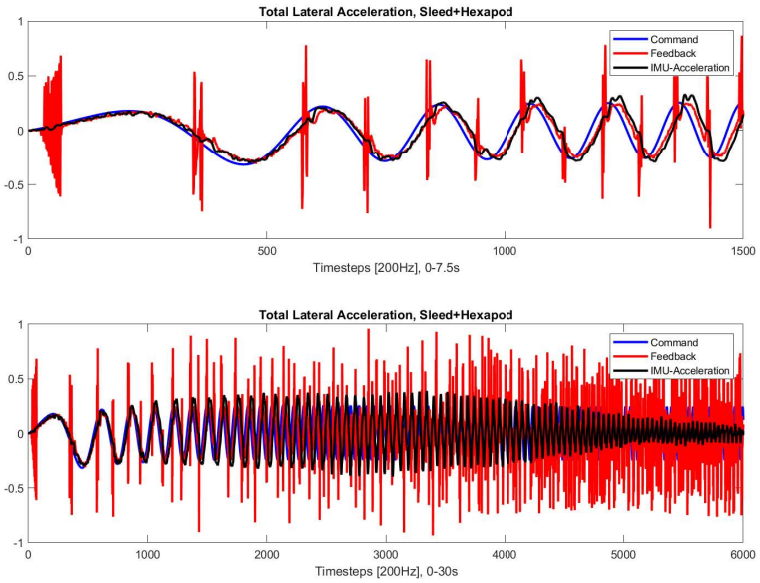


Figure 3.3: Test 1: Acceleration command, feedback, and IMU measurements, the first subplot is $0 - 7.5\,s$ and the lower the complete test of $30\,s$.

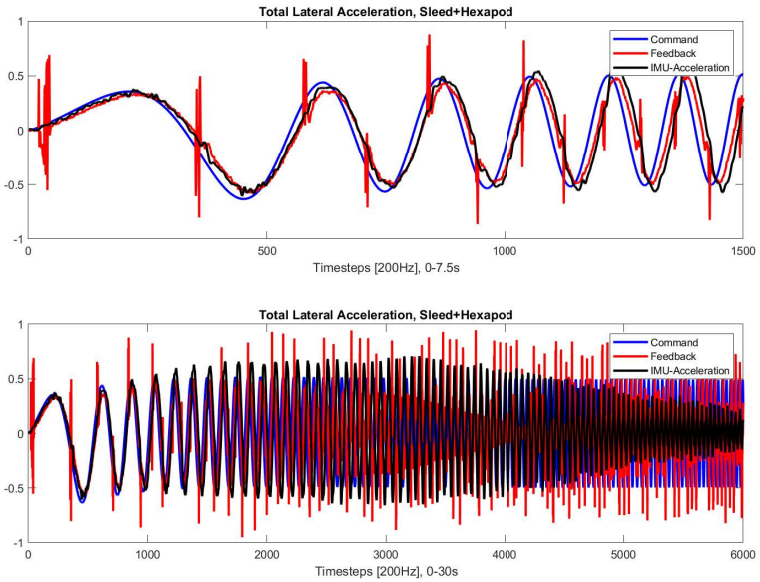


Figure 3.4: Test 2: Acceleration command, feedback, and IMU measurements, the first subplot is $0 - 7.5\,s$ and the lower the complete test of $30\,s$.

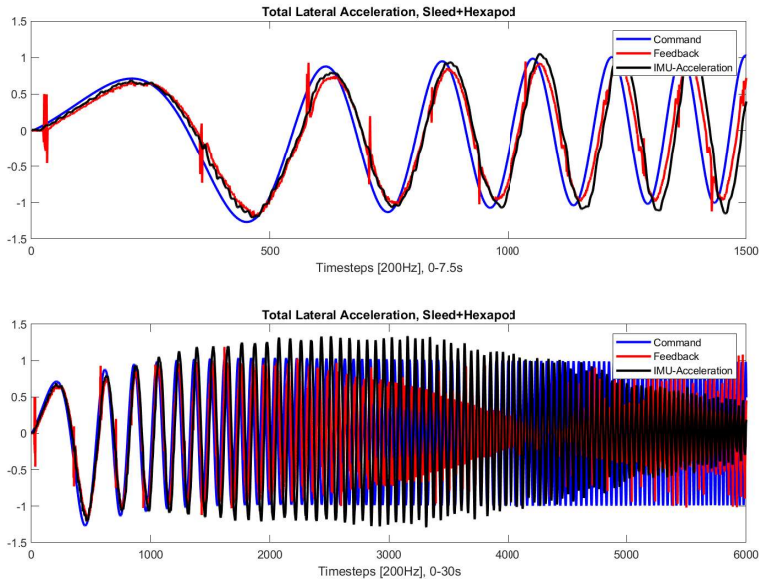


Figure 3.5: Test 3: Acceleration command, feedback, and IMU measurements, the first subplot is $0 - 7.5\,s$ and the lower the complete test of $30\,s$.

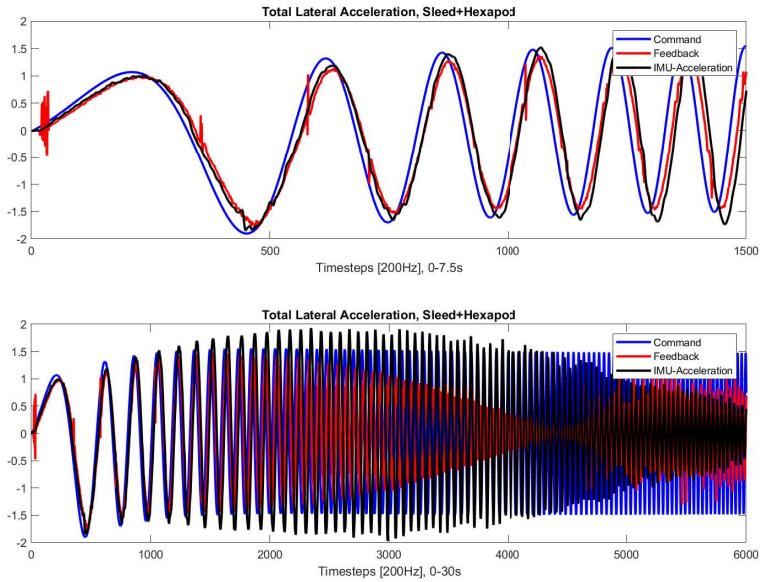


Figure 3.6: Test 4: Acceleration command, feedback, and IMU measurements, the first subplot is $0 - 7.5\,s$ and the lower the complete test of $30\,s$.

3.3 Conclusion

The results in Section 3.2 show that the difference between the acceleration feedback and IMU measurements is large. In the *RMSE*-values shown in Table 3.2 the values have similar for test number $2-4$. Test number 1 is not following this trend, this could be due to the amplitude being too low to properly excite the MS of Sim IV. This is taken into account when designing tests for system identification.

From the figures 3.2(a)-3.2(d), it is clear that the difference between the feedback and the IMU could be important when trying to lower the acceleration latency for the driver. Since the difference between the amplitude is big in some areas, using a model based on the feedback will in some frequency ranges predict an output that is very different from the value experienced by a driver.

Going forward in this project, the decision to implement an IMU is taken, based on the results in Section 3.2. The reason for trusting the IMU readings is the differences between the Feedback acceleration and the IMU acceleration and the inconsistency of the difference. Why the decision IMU and not the Feedback was due to the IMU used by Kumar being a High-end IMU that was calibrated professionally before it was used. This will hopefully give a model that better represents the accelerations experienced by the driver.

4

Methods

The method of this project is divided into some general areas, first a literature study, followed by a pre-study of similar experiments. After the pre-study, the next step is to develop the code for implementing an IMU, to be used in the fourth step. The fourth step is to design and perform experiments with the goal of collecting data for the system identification process.

After developing a model with system identification, that model should be used to implement a latency controller in Sim IV. The latency controller should minimize the latency between reference acceleration, and the measured acceleration from the IMU. After finishing the controller design, the next step is to perform new tests, that will evaluate the performance of the implemented controller.

4.1 Inertial Measurement Unit

As discussed in Section 3.3, the results from the prestudy indicate a noticeable difference between the acceleration reported by the system and the measured acceleration with an external IMU.

4.1.1 Equipment

The hardware used for implementing an IMU is stated in table Table 4.1, most of this hardware was available at VTI for use in this project.

Table 4.1: Table over Hardware used to implement an IMU

Hardware	**Link to source**
Arduino	[24]
Adafruit 9-DOF IMU	[8]
Raspberry Pi4 8GB	[25]
IMU case	[26]
Pi Case	[27]
Arduino Case	[28]
Mounting Bracket	N/A

The hardware already present for this project was the Arduini, IMU-board, and Raspberry Pi4. In addition to that, some 3D-printed parts were used, including a

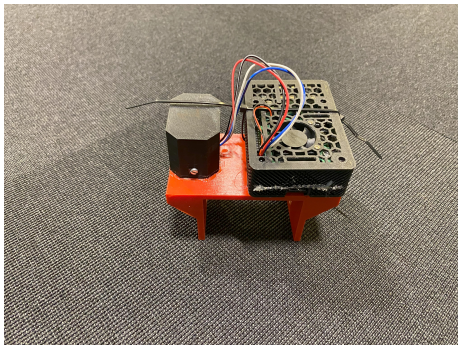
case for the Raspberry Pi, Arduino, the IMU, and a mounting bracket for fastening the IMU assembly to Sim IV.

4.1.1.1 3d-Printed parts

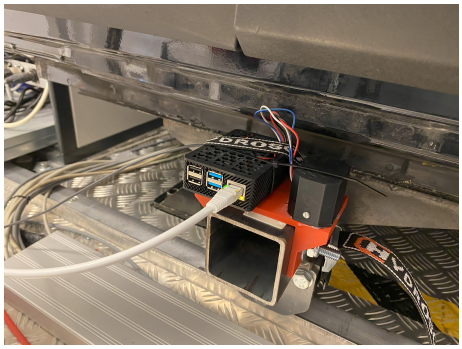
To mount the IMU assembly to Sim IV, a mounting bracket needed to be designed. The investigation, where to mount the IMU to Sim IV, a $70x70mm$ steel beam, was determined to be the best option. The beam provided a good reference for aligning the IMU and is also the same mounting point as used in the prestudy. In addition to the placement of the beam. Two pre-drilled holes also made it a perfect placement, see Figure 4.1 for the picture of the beam.

Figure 4.1: A picture of the beam used to mount the IMU.

The mounting bracket were designed using Catia V5 for this specific use-case. A U-profile turned upside down was determined to be the best option, (see figure 4.2(b) for the design). No structural calculations or FEM simulations were done on the design. This is because the loads were deemed low, and the bracket would have good support from the beam. If the design would have been too weak, that would be solved with a redesign. One other risk of not doing an analysis of the bracket could be the effects of the eigenfrequency and damping of the bracket. The bracket was then mounted with bolts to the pre-drilled holes on Sim IV, and secured with a strap if anything would break. Figure 4.2(a) and figure 4.2(b) show a picture of the IMU assembly.



(a) The IMU assembeled.



(b) The IMU mounted on Sim IV.

Figure 4.2: Pictures of the IMU.

As cases for the electronic-boards, designs under the CC-license was used, these are free to use, if "appropriate credit is given to the creator, and a link to the license is provided". In this thesis, the credit and license are fulfilled by stating all creators as references and referring to the license documentation. The decision to use the existing design was of time-saving nature, and it was not deemed to impact the test result [29].

4.1.2 Arduino and IMU

Then the decision to use an IMU in Sim IV was taken, and the search for suitable hardware was initiated. Previous tests have been done with in-house IMU-implementation for Sim IV. This Hardware(HW) was used as the starting point for the tests done in this thesis. This HW consisted of an *Arduni UNO*, and an *Adafruit 9DOF IMU*.

4.1.2.1 Physcical setup

The IMU board was connected to the Arduino using a guide from the manufacture [30]. The procedure is to connect the IMU boards to DC power (VIN and GND), clock (SCL), and data (SDA) from the Arduino. This can be done by either soldering or connectors. In this project, connectors were used. The Arduino was then connected to a Linux PC using a USB cable. The PC was placed in the Cabin of Sim IV, and acted as a link between the Simultor network, and the Arduino. Both the Arduino and the IMU were placed in their respective cases during the tests, seen in Figure 4.3.

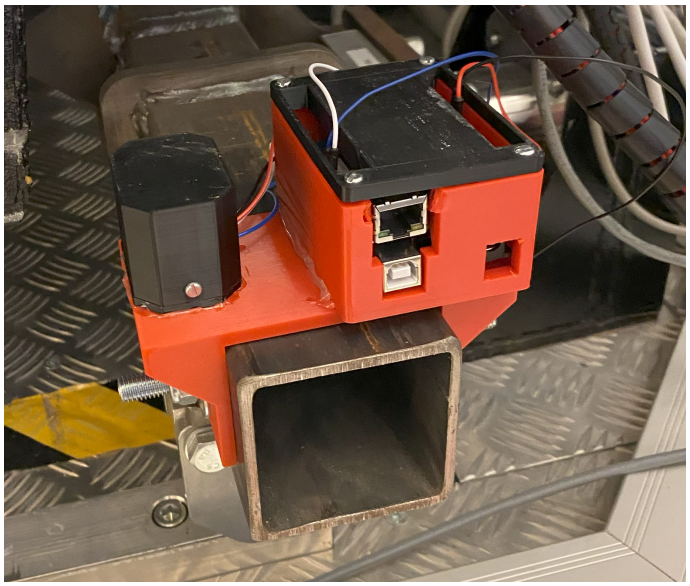


Figure 4.3: Picture of the Arduino-IMU assembley.

4.1.2.2 Code

Developing the code for Arduino was done in the program ArduinoIDE. ArduinoIDE is capable of writing and compiling the code and then flashing the code onto the unit. It is also capable of reading the output.

The principle of the Arduino-IMU-implementation is visually presented as a flowchart, in Figure 4.4.

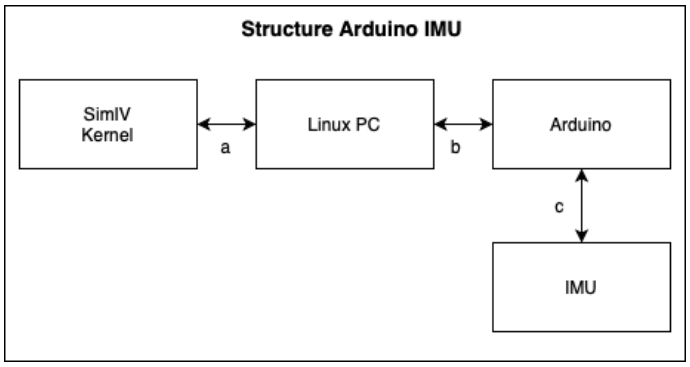


Figure 4.4: Flowchart of Arduino-IMU implementation, for signal description see Table 4.2.

The signals are described in Table 4.2.

Signal	Protocol	Information
a	SSH/SCP	Start/Stop File-transfer Time difference
b	Serial	Acceleration data
c	I2C	Raw Acceleration data

Table 4.2: Table over Signals in Figure 4.4.

The IMU logging code consisted of three main programs, one main Python program that was running on the Sim IV-kernel with the purpose of managing IMU logging. That is done by controlling the log time, experiment number, and also log files from the IMU and saving them at the kernel. Secure Shell (SSH) as well as Secure Copy Protocol (SCP) trough Python is controlling the IMU. SSH is used for starting, stopping, and setting time. SCP is used for retrieving the log files after each experiment. During the experiment, this program also measures the difference in system time between the Sim IV kernel and the Linux-PC with a frequency of 0.1 Hz. The time difference is saved and exported together with the accelerometer data.

The second program is running on the Linux PC placed in Sim IV, the program is the link between the Sim IV kernel and the IMU. It receives the start-command and starts writing the IMU data to a *.csv* file that can be retrieved by the Kernel after the experiment. The program reads the new information over the serial interface when it's availble. The data sort it to *x-acc, y-acc, z-acc*, and saves it to the *.csv*. At each data point, the program also stores the system time of the Linux PC.

The third program is running on the Arduino and is written as a *ino-language*. That is a programming language custom-made for the Arduino based circuit boards based on a subset of the *C++ programming language*. This program reads the Adafruit accelerometer with a frequency of 100 Hz and writes the readings over the serial interface.

4.1.3 Raspberry Pi4

As presented in Subsection 5.1.1, the Arduino IMU works but has problems and drawbacks. After analyzing the Arduino in Subsection 5.1.1, the decision was to replace the Arduino and Linux-PC with a Raspberry Pi4 8GB, this alternative solution had the benefit of including all functionality in one piece of hardware. Other hardware, such as the Adafruit IMU, and Sim IV kernel are the same in this setup too.

4.1.3.1 Physical Setup

The Raspberry Pi4 was connected to the IMU-board using the General-purpose-Input-Output(GPIO)-pins. The IMU-board communicates over the *I2C-protocal* and is therefore connected to pin number three(SDA) and five(SCL) to the corresponding pins(*SDA,SCL*) on the IMU. The power for the board is taken from pin number four(5v) and number six(ground) to *VIN and GND* on the IMU. The

Raspberry Pi and IMU were then mounted on Sim IV which was connected to the simulator network to enable communication with the Kernel through ethernet.

Mounting the IMU as in Figure 4.5 takes advantage to that takes advantage of the 90° angles of the bracket and IMU enclosure to align all axis of the IMU with the beam on Sim IV.

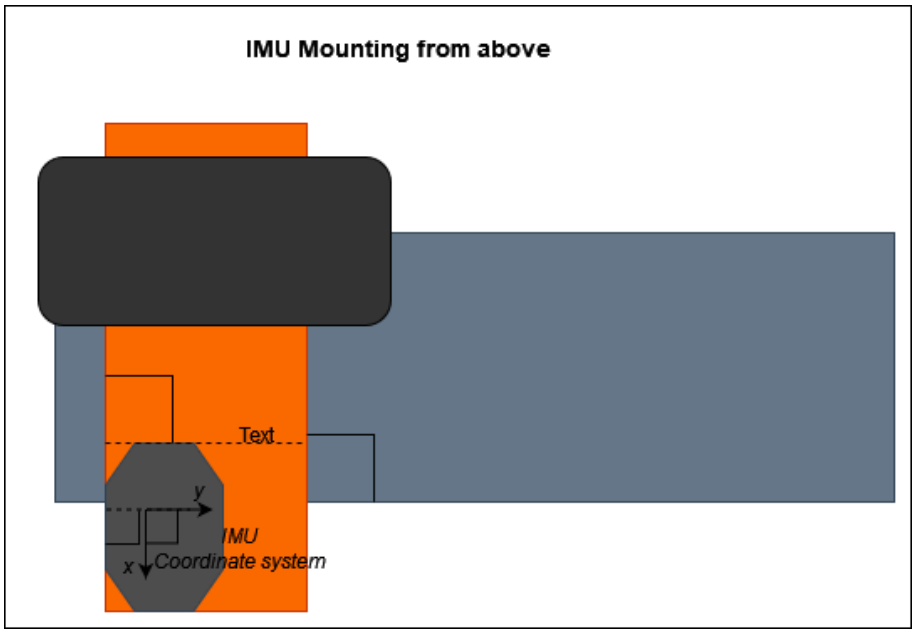


Figure 4.5: Schematic view of the mounting.

The bracket is then mounted at a 90 degree angle to the beam, and the IMU is placed parallel to the bracket's side.

Fast-Fourier-Transform analyses were done on the Z and Y axes to ensure that the IMU is aligned with the correct axis. If this analysis only shows noise, *i.e* the frequency peak seen in the longitudinal acceleration is not visible on the Z and Y readings. If this is true, the IMU is free from cross-readings.

4.1.3.2 Code

Connecting the IMU as described in Subsection 4.1.3.1 makes it possible to communicate with IMU through the Linux environment on the Raspberry Pi. The information flow when the Raspberry Pi is connected is described in Figure 4.6 and in Table 4.3.

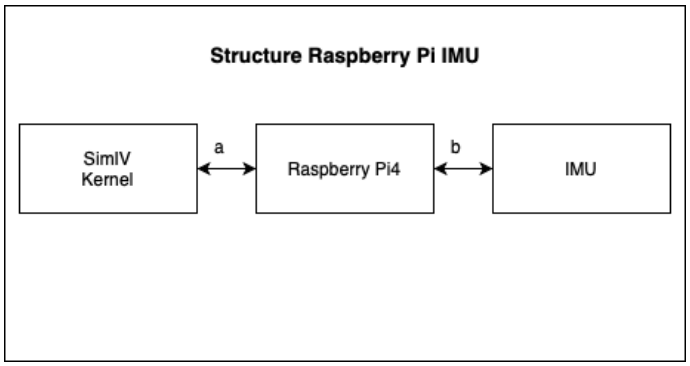


Figure 4.6: Flowchart of Arduino-IMU implementation, for signal description see Table 4.3.

Signal	Protocol	Information
a	SSH/SCP	Start/Stop File-transfer Time difference
b	I2C	Raw Acceleration data

Table 4.3: Table over Signals in Figure 4.6.

Then switching to the Raspberry Pi, many elements of the code in Subsection 4.1.2.2 were kept, and the program running on the Sim IV kernel changed to the following. The program is still handling log time, experiment number, and also log files from the IMU and saving them at the kernel. Secure Shell (SSH), as well as Secure Copy Protocol (SCP) through Python, is controlling the IMU. SSH is used for starting, stopping, and setting time. SCP is used for retrieving the log files after each experiment. During the experiment, this program also measures the difference in system time between the Sim IV kernel and the Raspberry Pi with a frequency of 0.1 Hz. The time difference is saved and exported together with the accelerometer data. The time difference vector is called Δt_{nm}.

The next program incorporates the functions of both reading the IMU and storing the data. The program is based on Python and is running on the Raspberry Pi. It receives the start command from the Sim IV kernel and starts writing the IMU data to a .csv file that can be retrieved by the Kernel after the experiment. The data sort it to x-acc, y-acc, z-acc, and saves to the *.csv*. At each data point, the program also stores the system time of the Raspberry Pi, in the format of milliseconds since the Unix-epoch.

For reading the IMU-board, the program utilizes the *CircutPython*-library from Adafruit. With the functions available the Raspberry Pi reads IMU over the *I2C* protocol. The IMU is read with a frequency of 200 Hz, which is the same as the simulation frequency of the Sim iV loop.

4.1.3.3 Syncing time between Kernel and Pi data

The aim of this part is to impliment *command*-acceleration (a_x) that is logged at a certain simulation time (t_s). It should then be compared with the IMU-acceleration (ai_x) that is logged at the same t_s. To make this possible, the local timeline on the Raspberry Pi is shifted to match the global simulation time at the Sim IV kernel. This is done by using the system times (t_n, t_m). The system time is the clock at each computer (Sim IV kernel, Raspberry Pi) since the Unix-epoch, in milliseconds.

In the Figure 4.7 to Figure 4.10, an acceleration (ai_x,a_x) value is visible, together with a global simulation time-stamp t_s, $t_s \in [0, T]$ in milliseconds where T is the simulation end time. The figures contain a system time t_n, t_m at each iteration as well, where t_n is the Sim Iv kernel system time and t_m is the Raspberry Pi system time.

In all figures, t_{n+0}, t_{m+0} is the starting time for the experiment. In an ideal case, the system time on the Sim IV kernel and *IMU* is fully synced, $n = m$. In this case, no shifting of the timelines is necessary and the data would line up.

The sync is done to align everything with the global simulation time t_s. The method below will explain how to correct the time difference in an imaginary example when the difference $t_{kernel} - t_{imu} = -10\ ms = t_d$. This means $t_{m+0} = t_{n+2}$ since the sampling time is $5\ ms$.

First, the mean difference between the two system times is calculated, in this example, that is $T_{mean} = 10\ ms$ as described. This is calculated by the formula in Equation 4.1 with the vector Δt_{nm} from Subsection 4.1.3.2.

$$T_{mean} = \frac{\sum \Delta t_{nm}}{|t_{nm}|} \tag{4.1}$$

The syncing process requires that the IMU vector is longer than the command vector. This is achieved by always starting the IMU logging before the simulation starts. In Figure 4.7 and Figure 4.8 this is exemplified that the command acceleration has ten values in Figure 4.7, and the IMU-acceleration has fourteen values in Figure 4.8. The extra values not used are deleted after the shifting is done.

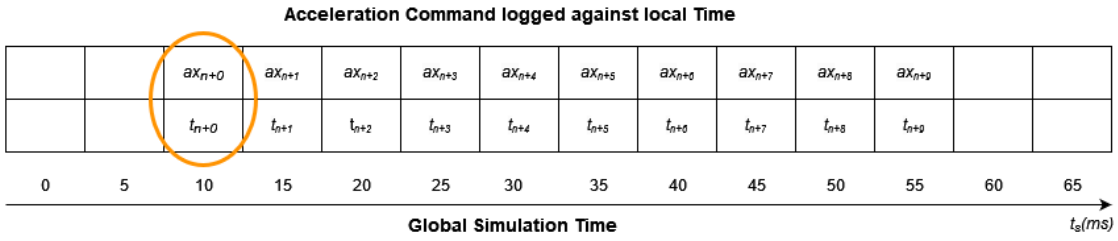


Figure 4.7: Acceleration command logged at the Sim IV-kernel system time.

In Figure 4.7, an example of the logged longitudinal acceleration command is shown. In every timestep, an acceleration value ax_{n+0}, a system time t_{n+0}, and the simula-

tion time is stored. The starting time of the experiment is t_{n+0}, and is shown in the orange circle.

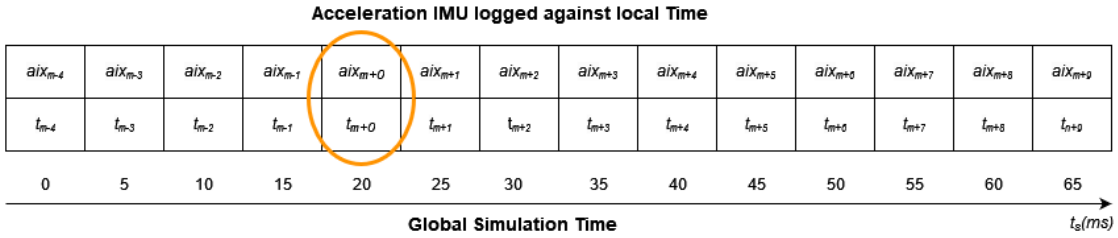


Figure 4.8: Acceleration from IMU logged at Raspberry Pi system time.

In Figure 4.8, the *imu* is 10 ms ahead of the *Sim IV-kernel*. This is visualized in the orange circle, that $t_m \Leftrightarrow t_s = 20\,ms$, while $t_n \Leftrightarrow t_s = 10\,ms$ in Figure 4.7.

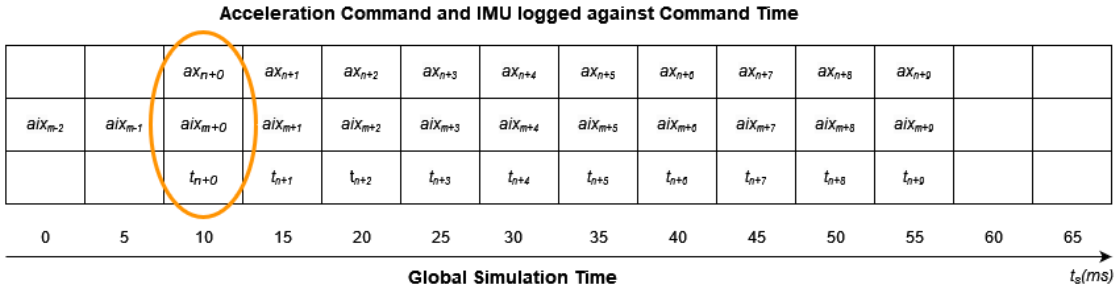


Figure 4.9: Acceleration Command and Acceleration from IMU logged at Sim IV-kernel system time.

The first step is to sync all the stored data in a common array, this is shown in Figure 4.9. As seen in this figure, without correcting the time difference the IMU data will be compared with the wrong command. As the the IMU-acceleration ax_{n+0} is now compared with the ax_{m+0}. But as described in the sections above, ax_{m+0} was logged at $t_{m+0} \Leftrightarrow t_s = 20\,ms$ and ax_{n+0} was logged at $t_{n+0} \Leftrightarrow t_s = 10\,ms$. To adjust for this shift the two system times (t_n, t_m) are used.

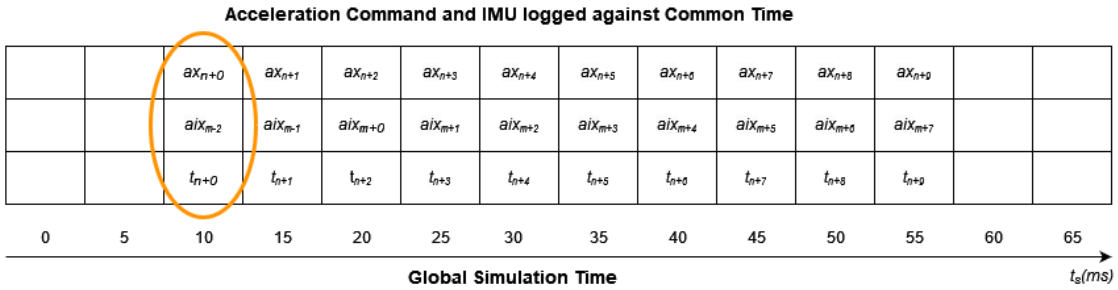


Figure 4.10: Acceleration Command and Acceleration from IMU logged at a Common-timeline based on Sim IV-kernel system time.

For shifting the Raspberry Pi values to match the global simulation time the mean time difference calculated in Equation 4.1 is used. The next step is to divide the mean time difference with the sampling time ($5\,ms$). In this example that gives a shift of negative two samples, *i.e* the Sim IV kernel is two steps behind the Raspberry Pi in system time. This is adjusted by moving all the IMU readings two steps before the data is compared. As shown in Figure 4.10 the comparison is now correct, that means ax_{m+0} at $t_s = 10$ is compared to ax_{m-2} at $t_s = 10$ as both the data was logged individually. The data from the IMU that was not used is also deleted at this step.

4.1.4 Accelerometer calibration

It was chosen to calibrate the IMU as part of the project. The method used is to calibrate the IMU to $\pm\,1\,g$ of accuracy, using a known leveled surface. To make this easier, the *IMU-case* have flat surfaces that are *orthogonal* [26].

In Figure 4.11 the six measurements used for IMU calibration are shown. The measurements were taken stationary under a period of 15 seconds for Z, Y, X axes. The equation that is used to correct the IMU readings is shown in Equation 4.2. In this equation, h_{cal} are the calibrated values, and h are the raw readings. For the compensation, A compensates for the nonorthogonal between the three accelerometers in the IMU, b compensates the IMU to read zero when no acceleration is measured.

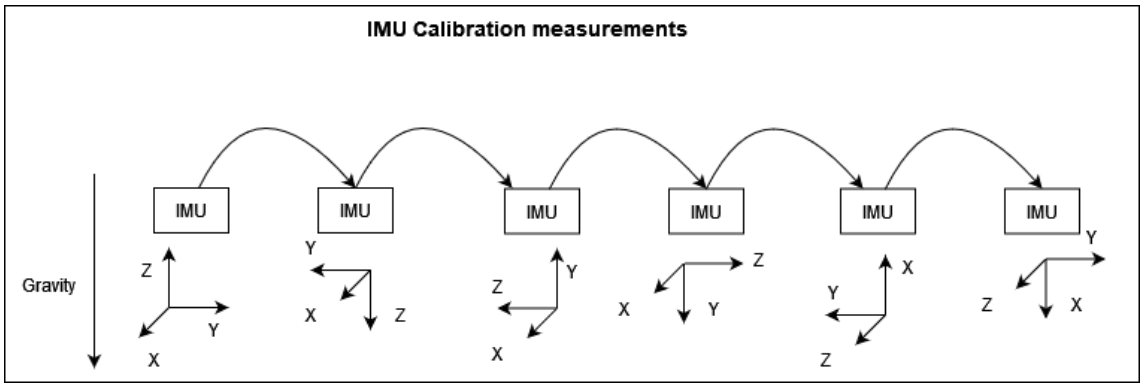


Figure 4.11: Measurements for IMU Calibration, along ± for all three accelerometers..

$$h_{cal} = (h - b)A \tag{4.2}$$

The A and b arrays are calculated using Matlab *MagCal*, which models the problem as a linear equation problem on the form $h = A \cdot h_{true} + b$ to find A and b. In this problem, h is the measured value on the form ax_i, ay_i, az_i, and h_{true} is the correct value for ax_i, ay_i, az_i. The subscript $i \in [1, 6]$ for the six tests is explained in Figure 4.11, so in this project, $h, h_{true} \in (6x3)$.

Validation for checking the IMU calibration is done through multiple methods, first by checking if the new static measurements end up on the $1\,g$-circle, and comparing the *rmse*-values between an optimal IMU and the uncalibrated and calibrated values.

The last method used for checking the IMU is to move the IMU a fixed distance and to calculate position according to Equation 4.3. Comparing the integrated position against the distance moved will give an indication that the accelerometer is reading correctly.

The A, b arrays that were calculated are used in Equation 4.2 and used in the process of importing the IMU data.

$$position = \iint a(t)dt^2 \tag{4.3}$$

A fixed distance of the Sim IV:s MS was used to move the IMU. The IMU was mounted to Sim IV as explained in Subsection 4.1.3.1. The simulator is moved from $x = 1.7\,m$ to $x = -1.7\,m$, this gives a total distance of $3.4\,m$.

4.2 System-Identification

The dynamical model of the Sim IV motion platform will be developed using System-identification, this method is a data-driven approach that uses measurement data for developing a dynamical model.

A simplification of the SI-process is presented in Figure 4.12, as shown SI is an iterative process that continues until a satisfactory result for the model is found.

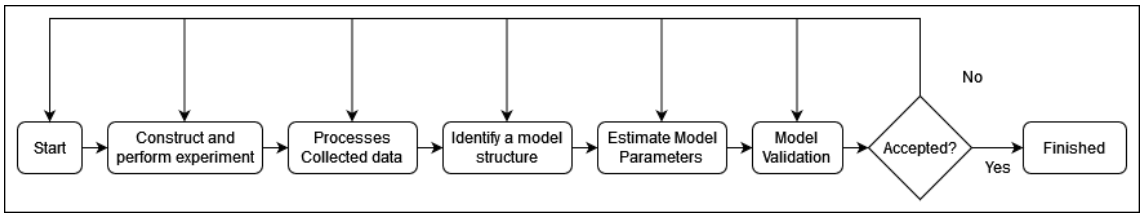


Figure 4.12: Flowchart of System Identification process.

When running simulations in Sim IV, many delays are present. This project aims to minimize the total delay, by identifying a model for the dynamics of Sim IV. This model will also include some of the time delays present.

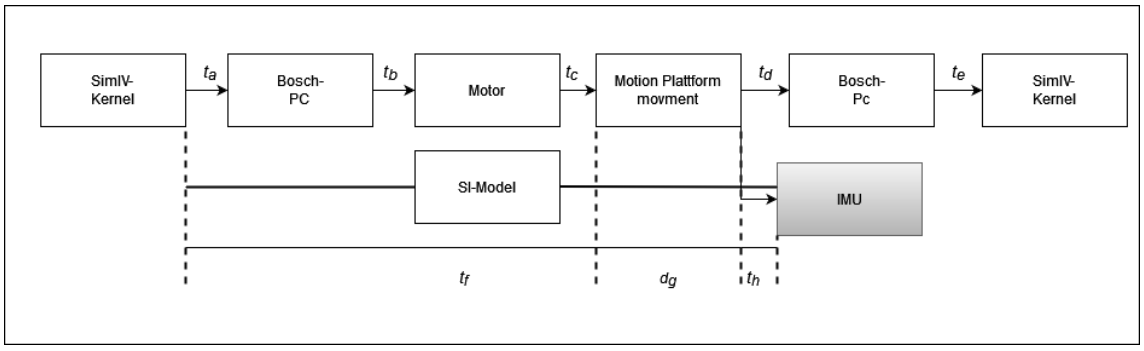


Figure 4.13: Structure of one simulation loop cycle in Sim IV.

In Figure 4.13 one cycle of the simulation loop is shown. This cycle includes a simplified information flow and some time delays. It also describes what part of this cycle the SI-model tries to identify, and what communication delays and dynamics are present.

As seen the model will cover the whole process from when the information leaves the kernel until the platform moves and the IMU registers it. This model will represent the two communication delays t_a, t_b. t_a and t_b are the delays from the command until the signal reaches the actuators. It will also be a delay t_c that corresponds to when the platform starts to move. These three delays are seen as one communication delay that is called t_f. The next part of the model is the dynamic in the MS of Sim IV, this is marked as d_g. This part is called d(g) to not be confused with pure transport delays. The last part included in the model is t_h which corresponds to the delay between acceleration in the platform to when the IMU registers it. In addition to these parts already explained, two more transport delays exist. This is the time t_d from the platform has moved until the Bosch PC is logging that data and the transport time from when the Bosch PC to the Sim IV-kernel back again.

For the model that will be developed, t_f, d_g, t_h will be considered as one part and be identified using System Identification.

4.2.1 Data aqusition

For performing SI, necessary data needs to be collected. This is done by performing in-output tests in Sim IV. The input was precalculated using the method described in Subsection 4.2.1.1. After the data was collected, it was split into development/-training data and validation data.

4.2.1.1 Calculation of test inputs

When calculating inputs for the I/O tests, it is many aspects to consider. The first is to ensure that the MS of Sim IV always stays within the limits for acceleration, velocity, and position. For the SI it is important that the MS is properly excited with different amplitudes and frequencies.

To always make sure that the inputs are within the limits, the limits from Table 2.1 are used, with a safety margin of 15%. The input generation code checks the inputs, and if any of the inputs are larger than the limits the code gives an error and makes saving the input impossible.

This project aims to develop a model from *acceleration command* to *measured acceleration*.

When running tests in Sim IV, the system has a starting process of approximately 90 s before it accepts commands. To make sure that all tests work without missing parts a 120 s *zero input* is applied in the beginning, shown in Figure 4.14. To make sure that the simulator stops safely, a 30 s of *zero input* is also added to the end. It is also important to have smooth transitions from stationary to a moving simulator. That was achieved with a 10 s linear ramp from $0 \rightarrow 1$ and $1 \rightarrow 0$ before and after the test. The full test input was performed under a period of 15s. All this was

put into a vector with values between $[0, 1]$, and that vector was multiplied by the desired input. The profile of this vector is shown in Figure 4.14.

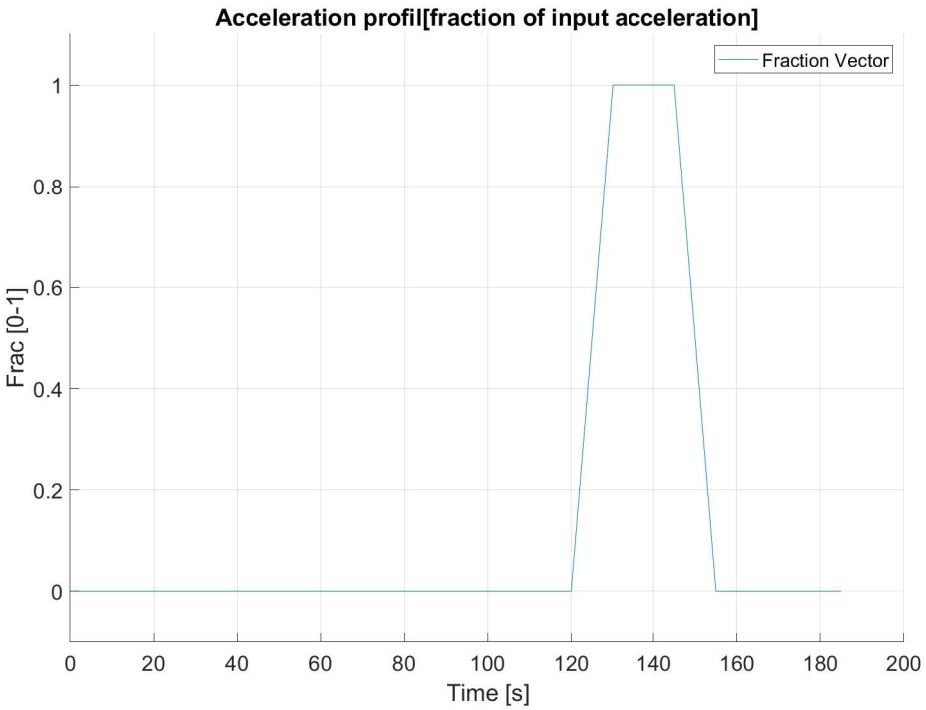


Figure 4.14: Fraction of full input. Including 120 s of zero input, 10 s ramp up/down and 15 s of full input.

To generate the different input vectors. The function in Equation 4.4 was used to generate different acceleration vectors. For velocity and position, the acceleration function was integrated $v = \int a(t)dt$ and $p = \iint a(t)dt^2$.

$$a(t) = amp * sin(2\pi ft) \tag{4.4}$$

Using this method, three vectors were generated automatically, and saved if they did not violate any of the above-stated limits.

In Figure 4.15 an example of how the input looks when it is visualized in a position/time diagram. As seen, the limits are also plotted to check that the inputs are within the limits visually. In Subsection 4.2.1.2 and Subsection 4.2.1.3 all the different frequencies and amplitudes are presented.

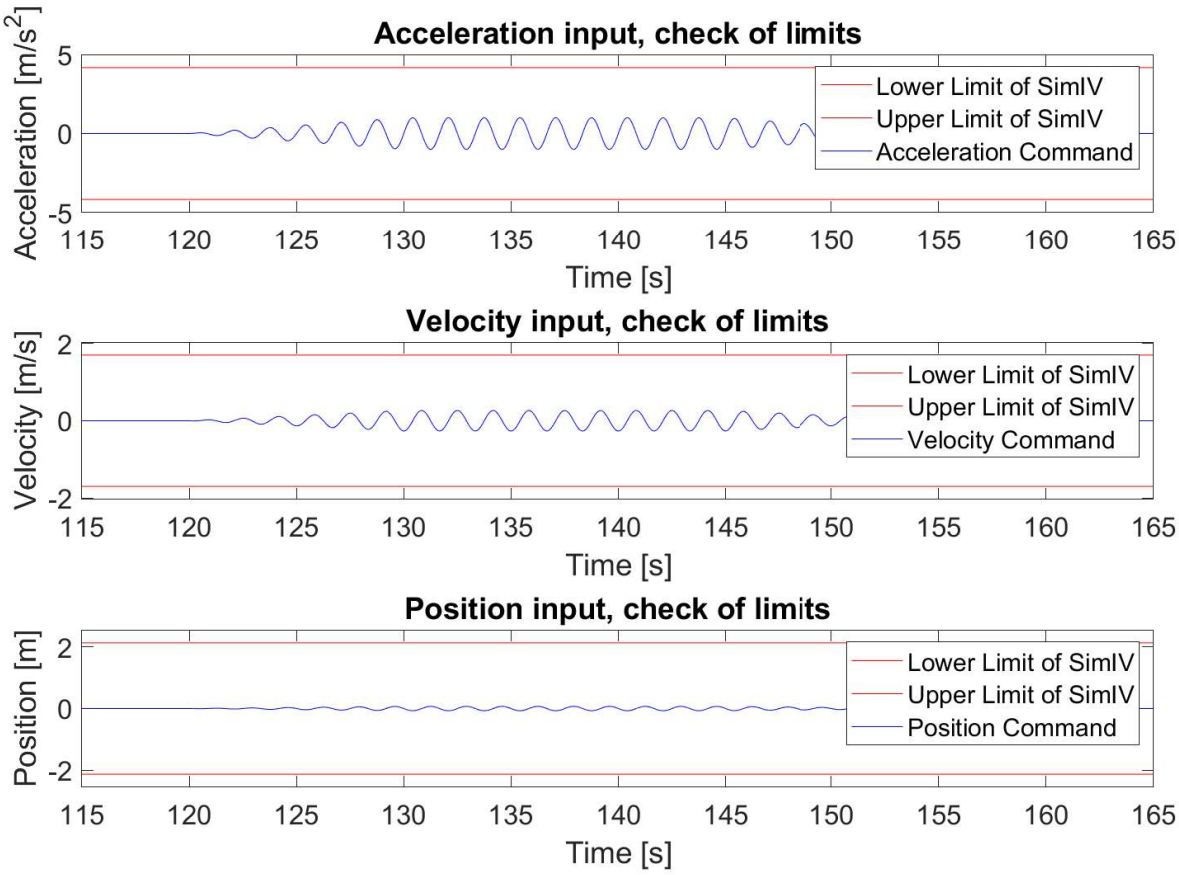


Figure 4.15: Example of how the three input vectors are looking, in the figure the acceleration amplitude is $\pm 0.6\, m/s^2$, and with a frequency of 0.1 Hz.

4.2.1.2 Data for model development

As described in the theory section about SI (Section 2.4), it is important that the tests for model development are properly exposing the dynamics of the system. For Sim IV the bandwidth of the XY-table is $0 - 3Hz$ and was therefore chosen to be the limit for SI-data. From analyzing what frequencies were represented by the XY-table from previous studies [31], it was found that the most frequencies were between $0 - 1Hz$, this is shown in Figure 4.16.

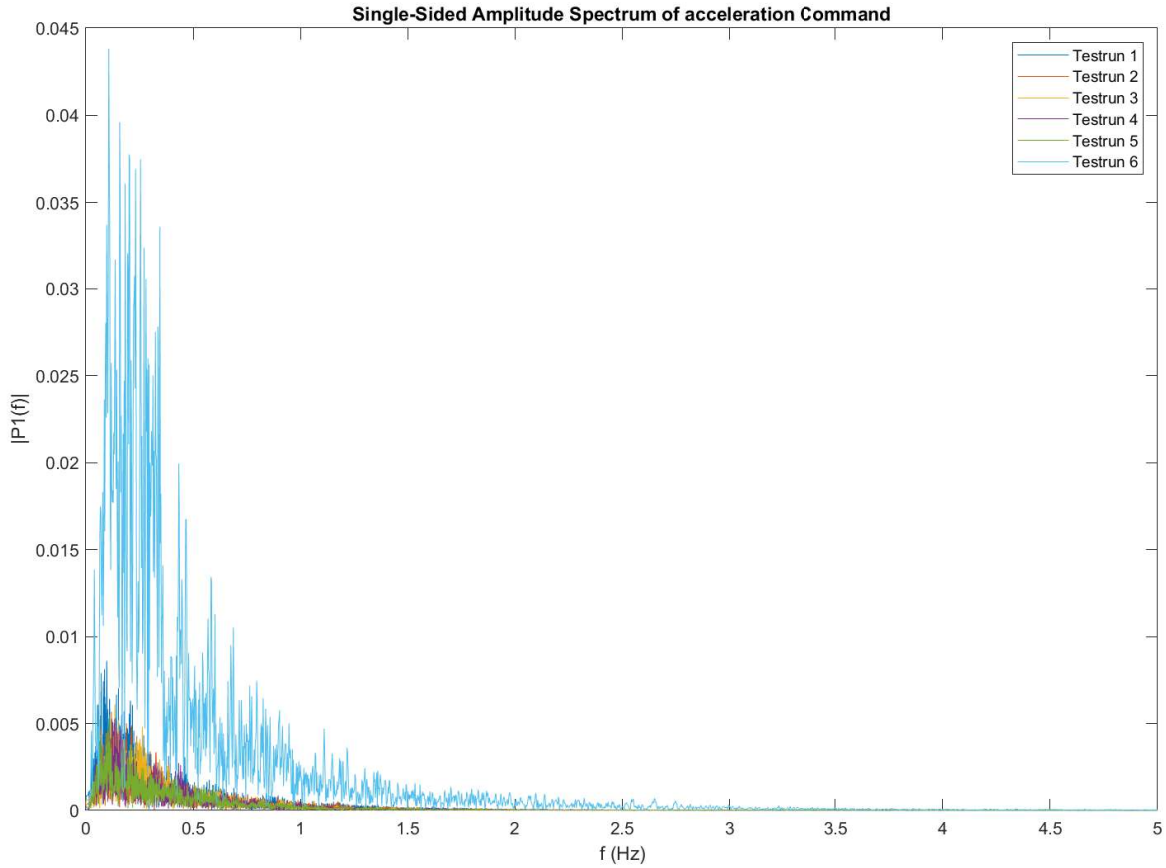


Figure 4.16: FFT of the acceleration command for existing driving data in Sim IV.

From the FFT analysis in Figure 4.16, the decision was to place most of the experiments with a frequency between $0 - 1\ Hz$, but tests up to the bandwidth of $3\ Hz$.

In Table 4.4 the nine different model development tests are presented. When used in model development, the zero part of the test, in the beginning, is left out. Only $7.5s$ at the beginning and end of each test is kept. The resulting part will look like the input in Figure 4.15. In the end, the data from $115\ s - 165\ s$ simulation time are used.

Table 4.4: The different tests used for model development.

Test nr	Amplitude	Frequency
801	0.5	0.2
804	1	1
805	1	2
806	1	3
881	0.6	0.1
883	2	0.9
885	2	0.7
887	2	0.5
888	2	0.4
889	2	0.3

When all the tests are shortened to only include the part of interest, the nine tests are merged into one long test. The longer test is made to have one long timeline of $50 * 9\,s$ that is easier to analyze.

To check if the longitudinal accelerations on other places on the motion platform are the same. One more test was done with the IMU moved. The IMU was moved from the beam on the rear right side of the Volvo XC60 to a similar beam on the front. In Figure 4.17 the movement is visualized with the 1st placement in orange and the 2nd placement in green. The test that was run as a reference is the number 801, with an acceleration amplitude of $0.5\,m/s^2$ and frequency of $0.2\,Hz$.

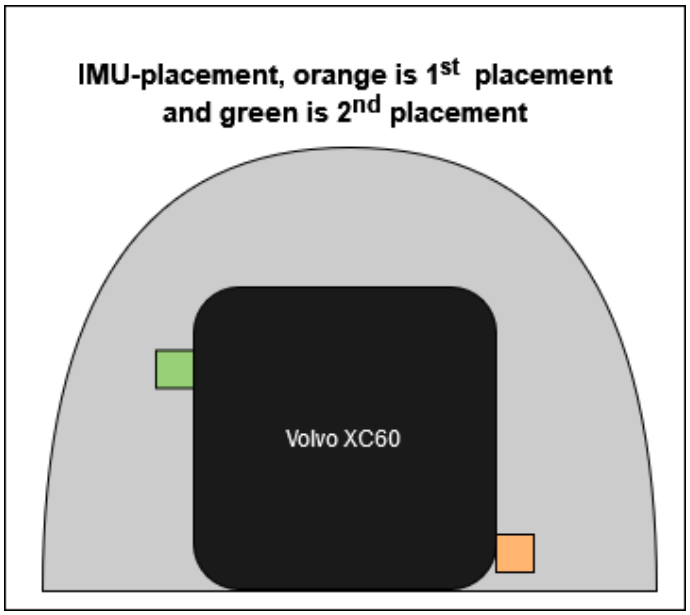


Figure 4.17: First and second placement of the IMU.

4.2.1.3 Data for model-validation

In addition to the data collected for model development, data was also collected for use in model validation. The test method is the same as for the tests in Subsec-

tion 4.2.1.2, but the amplitudes and frequencies will be different since the theory section describes in Subsection 2.4.1.3. Most frequencies for model validation are between $0-1Hz$ for the same reasons as for model development as described in the model development, see Subsection 4.2.1.2.

Before the IMU data could be used it needs to be filtered to reduce the noise. For filtering the data, a *forward-backward-filtering* method was used. The reason for using this filtering technique was to not induce any phase shift in the data when filtering it.

The filter used is a low-pass (LP) filter. The decision to use an LP filter was the knowledge of what output frequencies to expect depending on the input frequencies. This made it easy to set the passband frequency in the filter just above the highest frequency of the expected output. The passband frequency in the LP filter was set to 3.5 Hz.

The six different tests in Table 4.5 was used for model validation. The Validation data set is converted from six tests to one long test in the same way as described for the model development dataset.

Table 4.5: The different tests used for model validation.

Test nr	**Amplitude**	**Frequency**
802	0.5	1
803	0.5	2
880	0.8	0.2
882	2	1
884	2	0.8
886	2	0.6

4.2.2 Model development

For doing SI it was decided to use the *Matlab System Identification Toolbox*(MSIT). The use of MSIT was due to it being one of the most popular tools used for SI with good documentation. The documentation consists of both user-generated advice on different websites but also as mentioned in Section 2.4 an 800-page guide for using MSIT by Ljung [18].

For model development in Matlab, the first step was to sync the time between the Sim IV-kernel and the Raspberry Pi as described in Subsection 4.1.3.3. The next step was to create an array with the structure described in Table 4.6. In addition to the IMU readings, the *Ref.ax* is the signal that the simulator is should follow. The *Bosch.cmd* is the acceleration command that is sent to the simulator(without control $Ref.ax = Bosch.cmd$), and *Bosch.fdbk* is the reported feedback from the Simulator. For system identification purposes, the feedback signals are not used but exist for validation of the controller later on.

Table 4.6: Structure of SI-data.

Time	IMU.ax	IMU.ay	IMU.az	Cmd.ax	Fdbk.ax	Ref.ax
:	:	:	:	:	:	:

This data was then used for estimating two model structures with MSIT. The two model structures used were a first-order *State-space*, and a first-order *Transfer-function*. The work started with 1st-order models.

For estimating the models the $Time$, $IMU.ax$, and $Ref.ax$ from Table 4.6 is used. This is converted to an *iddata*-object, that can be used with the estimation functions. For estimating a state space model, the *iddata* and the order(1) of the model are fed to the *ssest* function. For estimating the transfer function, the same *iddata* is used. With the *tfest* function, the number of poles(1) and zeros(0) are provided.

With these two estimation functions, the models were developed. This is an iterative process. So, models with many different model sets were developed. This process is described in Subsection 4.2.3.1, but the final model was developed with the data sets in Table 4.4.

4.2.3 Model Validation

When a model has been developed, that seems to perform well against the training data, the model will be tested with another set of data. If the performance is satisfactory, the SI-process is "finished". Otherwise, the process continues in a new iterative cycle, as shown in Figure 4.12. For validating data, the *compare* function in MSIT was used. The *compare* function uses NRMSE to calculate the percentage fit between the validation input/output data and the models predictions of the aforementioned. This metric was the primary criterion used for evaluating the estimated models. In combination with creating a Bode plot of the model and comparing it to the Bode plot that Bosch provided when installing Sim IV.

4.2.3.1 Reiteration

As SI is an iterative process, many different models were developed. The first part was to develop a model that included the tests over the bandwidth, but these were separated from SI later. First, the length of the *initial zero* was shortened to better show the different dynamics instead of the standstill period. As well, the number of tests for validation and development was increased.

Other structures that were tested were to try to model the transfer function with a dead time delay. This approach was not successful. MSIT did not estimate any delay for the put-together data set, and then trying to estimate a delay for the different data sets individually, it estimated the different delays for different data sets.

Another test where done by changing the order of the SI-model. This was done for both the transfer function and the State Space to check if a higher model-order would result in a better fit.

4.2.3.2 Finial Validation

The final validation was done in the same way as the continuous validation under the project. The only difference was that after the final validation, the project moved to design the controller. Criteria that stopped the iterative process were that the fit percentage did not change when more dynamic tests were added. Lastly in the end the overall fit was high ($> 85\%$).

4.3 Prediction-Algorithm

From the final validation, the transfer-functions structure was chosen to be implemented in a prediction algorithm. This decision was due to the overall higher fit for the transfer function, and its Bode plot correlated better with the one provided by Bosch.

4.3.1 Development of Prediction-Algorithm

The motion cueing in Sim IV is done by a complex Simulink model that is compiled to a *C++*-code for use with Sim IV. This Simulink model is taking the VM accelerations and calculates what accelerations should be represented by Sim IV, depending on what the state of Sim IV is and the amplitude of the accelerations.

In Figure 4.18 the principle for where the controller will be implemented is shown, the controller were implemented in the MC-Simulink model. The controller uses the calculated simulator acceleration as a reference and will adjust the acceleration command to control the simulator accelerations.

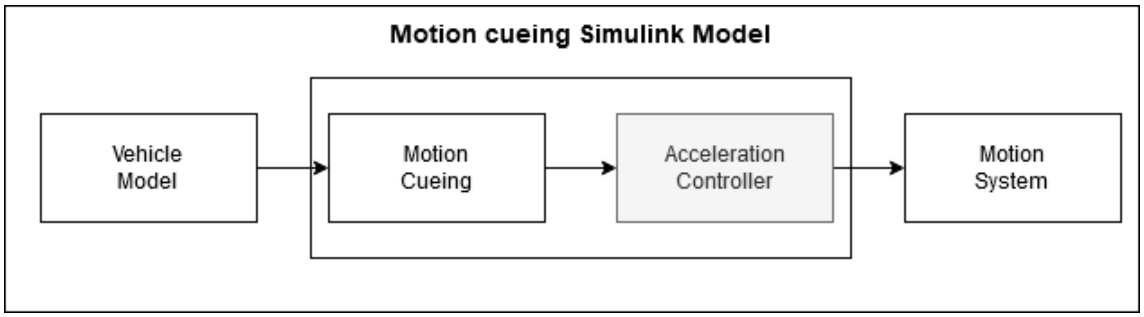


Figure 4.18: Principle for controller implementation. The two blocks inside the square represent the MC-simulink.

The principle for this controller will be to implement the model part of the Smith-predictor(described in Subsection 2.6.1). This approach will use the model developed in Subsection 4.2.2 to simulate a closed loop in a closed-loop-controller. The controller then uses the estimations from the model as the plant-state to compare with the reference state.

The structure of the implemented controller in this thesis is shown in Figure 4.19. It was important when designing this controller to not bypass the Sim IVs safety

systems. To ensure this, the controller includes an input limiter and it is also placed before the safety blocks.

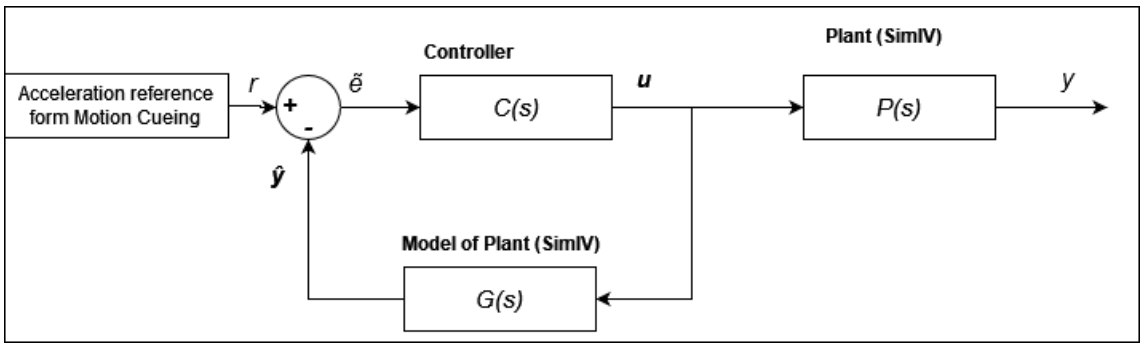


Figure 4.19: Controller structure used in this thesis.

Since this project aims to design a method that proves that it is possible to lower the acceleration latency in Sim IV, the controller structure used is a simple *PID* represented by a Simulink *PID-Controller* block. The tuning of this controller is done by the tool *pidTuner*, which is part of the Control System Toolbox in Matlab. This is a tool that easily helps the user to tune the controller.

To test the controller in the development phase, the SI model and controller were simulated in Simulink with different inputs. In the end, controller parameters that gave a noticeable improvement were chosen. Then that controller was implemented in the MC-simulink for use in the simulator.

4.4 Simulator experiments for validation of Prediction-Algorithm

After a working PA has been implemented in Sim IV the next step was to validate it. This was to investigate if, and what type of difference the controller is doing with respect to longitudinal accelerations. The method used for validating the controller is based on the method developed for collecting the data for System Identification. The same interface was used for inputting acceleration, generating input, measuring data, and sync time.

4.4.1 Synthetic tests

The first experiment was to perform the same tests as in Subsection 4.2.1.2 and Subsection 4.2.3, but instead use the prediction algorithm. The aim is to compare the response of the simulator with and without the PA, to investigate what the impact is on the platform response and acceleration delay.

The tests with the controller were performed approximately one and a half months later than the data collection tests, with heavy use of Sim IV in a simulator study in between. To ensure that the IMU hasn't moved, one reference test where done.

If the reference test differed from the new tests, it would not be possible to compare them, and all tests without the PA need to be rerun.

After checking the IMU mounting, all test in both Table 4.4 and Table 4.5 is rerun with the controller activated. This makes it possible to compare the results of the controller. Both for the delay, and if the amplitude of the acceleration has changed.

4.4.2 Tests with real driving data

In addition to the synthetic tests done for evaluation of the PA, three tests with data from previous driving sessions were also used, shown in Figure 4.21. The first two test runs were with data collected during the study for the forthcoming publication by Weibull et al. These two tests were driving on a highway with little speed changes, resulting in small longitudinal accelerations. To test how the controller reacts to driving with higher longitudinal accelerations a third test with driving data was done. This data was generated by driving the same highway but with large changes in speed. Accelerations from $\approx 30km/h \rightarrow \approx 130km/h$ and then hard brakings from $\approx 130km/h \rightarrow \approx 30km/h$ were done. The vehicle acceleration profile for the aggressive drive is shown in Figure 4.20. This resulted in very high longitudinal accelerations for the simulator [31].

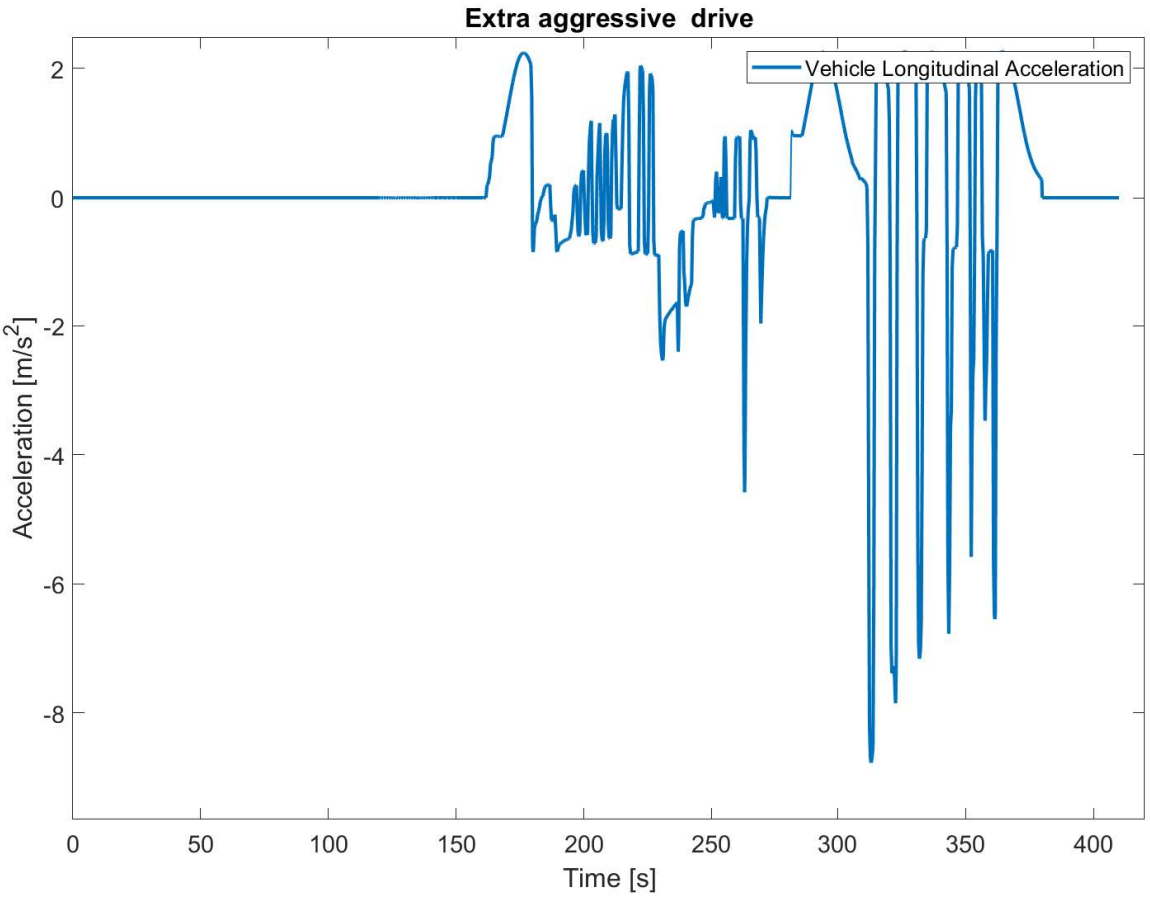


Figure 4.20: The vehicle longitudinal acceleration for the extra aggressive drive.

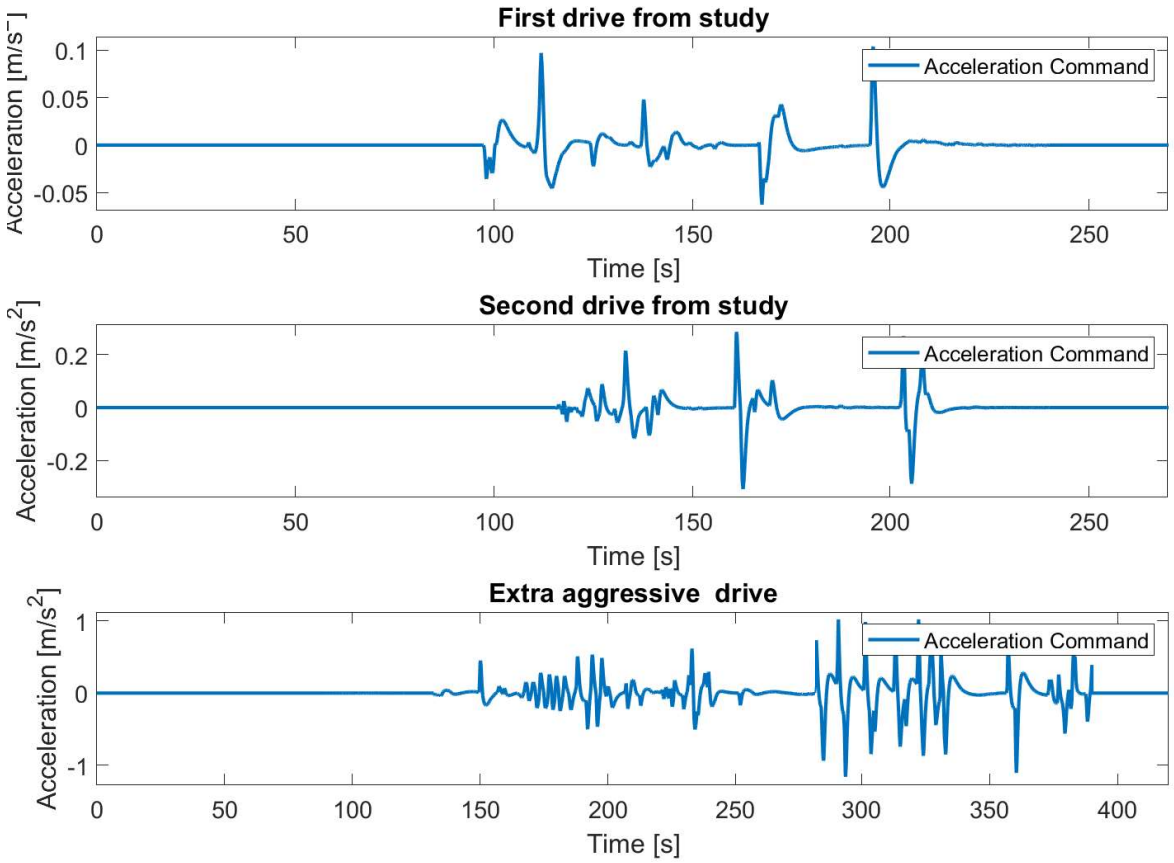


Figure 4.21: The longitudinal Acceleration reference command for XY-table from the previous drives.

All these three tests were done twice, the first time with the controller switched off, and then with the controllers turned on.

The acceleration reference was set to be the logged acceleration command from the previous drives. The benefit of using longitudinal accelerations from existing drives is that no calculation with respect to physical boundaries needs to be done, since the simulator already has performed all this motion without any problem.

4.4.3 Validation of the Prediction-algorithm

After performing the tests with the controller in Subsection 4.4.1 and Subsection 4.4.2 the next step was to analyze the test and present a final result of the controller impact. The metrics that are used for this analysis are listed in Table 4.7

In Figure 4.22 the principle for calculating latency is shown. The method is to calculate the difference in $x - value$(time) when a certain $y - value$(acceleration)

is measured. In this project, the difference between the peaks *signal 1* and *signal 2* are used as latency.

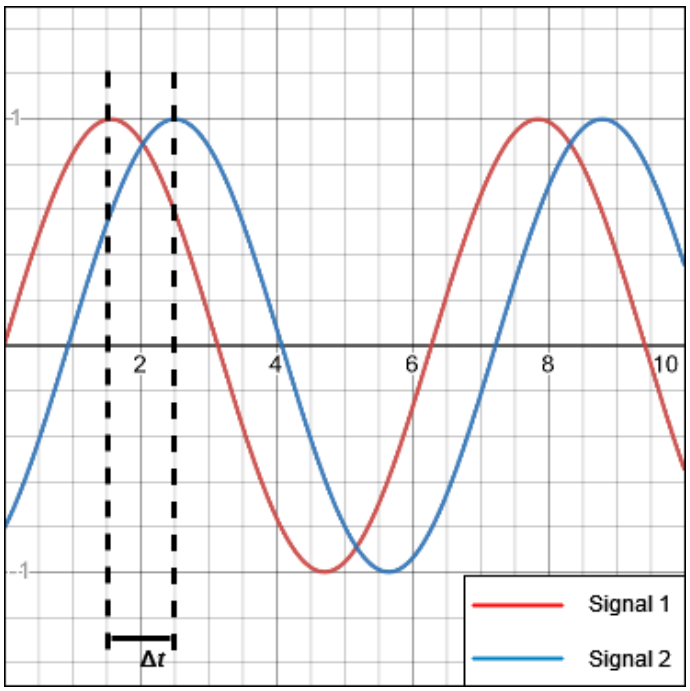


Figure 4.22: The definition of latency when analyzing the data.

This latency calculation is done with *Acceleration Reference* as *Signal 1* and *IMU acceleration* as *Signal 2*, and with the *Acceleration Reference* as *Signal 1* and *Acceleration feedback* as *Signal 2*.

The Table 4.7 is showing the tests done to calculate what the difference is in the latency for the different tests, after calculating the latency the mean, median, and standard deviation are calculated. Calculating these statistic values is to compare if the controller is consistent in the latency change compared to the latency without a controller.

Table 4.7: Parts of the different measurements that are used for analyzing the latency controller.

Nr	Tests (Reference and IMU)	Nr	Tests (Reference and Feedback)
1	Difference Reference to IMU-acceleration latency without controller $(ref_{ax} - IMU_{ax})$	9	Difference Reference to Feedback acceleration latency without controller $(ref_{ax} - fdb_{ax})$
2	Difference Reference to IMU-acceleration latency with controller $(ref_{ax} - IMU_{ax})$	10	Difference Reference to Feedback acceleration latency with controller $(ref_{ax} - fdb_{ax})$
3	Mean of test number 1	11	Mean of test number 9
4	Median of test number 1	12	Median of test number 9
5	Standard deviation of test number 1	13	Standard deviation of test number 9
6	Mean of test number 2	14	Mean of test number 10
7	Median of test number 2	15	Median of test number 10
8	Standard deviation of test number 2	16	Standard deviation of test number 10

In addition to the metrics in Table 4.7 some more values are calculated. These tests are with the same *RMSE*-values that were calculated in Subsection 3.1.2 for both with and without the controller activated.

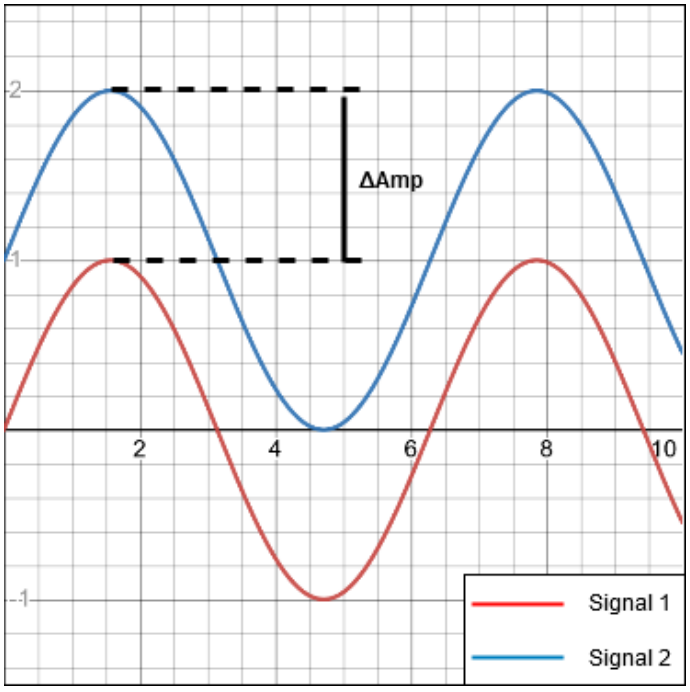


Figure 4.23: The definition of difference to reference amplitude then analyzing the data.

In addition to the tests focused on time delays, the amplitude of the acceleration

was also evaluated. The delta in amplitudes is defined as shown in Figure 4.23. For evaluating the amplitude, the tests in Table 4.8 is done.

Table 4.8: Parts of the different measurements that are used for analyzing the latency controller.

Nr	**Test(Reference and IMU)**
17	Difference reference amplitude to measured IMU-amplitude without the controller.
18	Difference reference amplitude to measured IMU-amplitude with the controller.
19	Mean of test number 17
20	Median of test number 17
21	Standard deviation of test number 17
22	Mean of test number 18
23	Median of test number 18
24	Standard deviation of test number 18

5

Results

The results in this thesis will contain four main parts, first the implementation of an IMU setup. Secondly, the results from the IO test will be presented. The results from the IO tests will then be used to develop a model using System Identification, which is the third result. The fourth result is the controller, how it is designed, and what difference the controller makes when doing the same tests as without the controller. All of the produced code is stored on a private GitHub repository, for access and review of the code please contact the author on GitHub [32].

5.1 IMU

As presented in the Conclusion(Section 3.3) from the prestudy, the first step of this project was to implement an IMU for measuring the acceleration in Sim IV. The IMU implementation was done in two iterations. The first implementation was with an Arduino, and then with a Raspberry Pi4.

5.1.1 Arduino-IMU

The first IMU implementation was based on an Arduino UNO and an Adafruit 9-dof IMU. All this hardware was existing at the VTI Gothenburg office before the project and was therefore used. The Arduino implementation worked and was able to measure, store and send the acceleration values.

As presented in Subsection 4.1.2, the Arduino was used together with a PC, storing the acceleration printed over serial. This setup had a couple of problems, making it not optimal for this project.

It was three main drawbacks to using Arduino as a base for the IMU. The first and most important issue was the Arduino performance, the simulation software at Sim IV is sampling at $200\,Hz$. Then using the Arduino it was not possible to consistently run the IMU with a frequency higher than $100\,Hz$.

As a result of this problem, the least computational-heavy way of running the Arduino was to *print* the IMU-accelerations over serial-USB. This setup requires the use of an additional computer that logs the data to a file. Using an extra Computer comes with the drawback that the computer needs to be secured and safe on the motion platform of Sim IV.

The third drawback with this method was the time synchronization, it required three different systems to be synced to the same time or the differences logged and adjusted for. The data will also be hard to synchronize the time since it will be read by the Arduino at time t_n and then stored at the PC at time t_m. An then compared with the command that is stored at the Sim IV-kernel at time t_k. In addition to the problem of syncing three systems, the Arduino was also having problems with keeping a consistent time. Due to the absence of an internal clock for keeping the time.

5.1.2 Raspberry Pi-IMU

After evaluating the Arduino-IMU, the decision was to switch to a Raspberry Pi4 instead. This section will describe the result of that IMU.

Communication with the IMU for doing the experiment was done by the Linux terminal(shown in Figure 5.1), this provided a simple and user-friendly way of inputting what experiment(number) was performed and for how long the accelerations should be logged.

```
sim@sim4kernel:~/code/others/Feedback_compensation/Py$ python3 runIMUTests.py
Start
Please enter 'true' or 'false', for start log data with IMU. Or enter 'exit' for quit the program:true
Please input how many seconds the data logging should be:10
Please input the testnumber:1
```

Figure 5.1: The program for interfacing with the IMU.

With the information inputted in Figure 5.1 the programs create a *CSV-file* on the kernel with the name *test_* $<$ *test number* $>$ *.csv* that includes an initial measurement of the time difference. This file is transferred to the Raspberry Pi and is used for saving the IMU readings when the experiment is running. After the experiment is finished, the file is transferred to the Sim IV-kernel and stored with the name *result_test_* $<$ *test number* $>$ *.csv*.

```
1  "Time difference between local and remote machines:; -47.904541015625; [ms]"
2  1679386031242,-0.61193496,-0.07649187,10.517632125
3  1679386031243,-0.61193496,-0.15298374,10.517632125
4  1679386031244,-0.61193496,-0.15298374,10.517632125
5  :,            :,          :,          :,
6  :,            :,          :,          :,
7  1679384182740,-0.61193496,-0.15298374,10.517632125
8  "Time difference between local and remote machines:; -48.3212890625; [ms]"
9  [-47.7578125,:, :, :, -47.797607421875]
```

Figure 5.2: Log file form the IMU, (":" is representing cutout values).

At the Kernel, the Python program is adding the time difference between the Kernel at the end of the experiment and the time difference that has been logged during the experiment at 0.1 Hz. The file will then have the structure shown in Figure 5.2, with the structure *[Raspberry Pi system time, x-acc, y-acc, z-acc]*. The system time is logged in the Linux standard format, which is the number of seconds since

the UNIX-epoch (1st Jan 1970 UTC). Since this project is requiring millisecond precision, the first column is milliseconds since the UNIX-epoch.

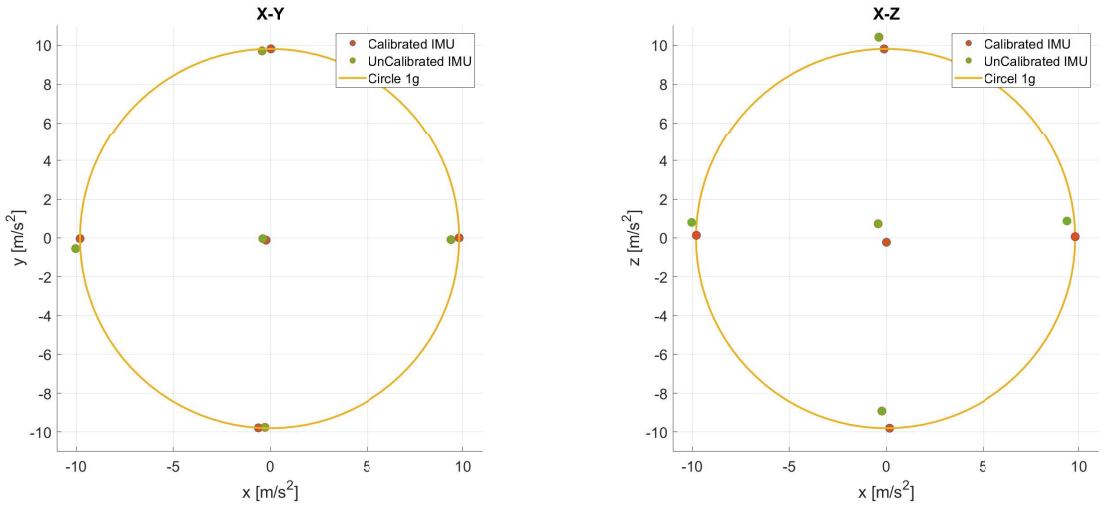
5.1.3 Accelerometer calibration

Before the IMU could be used to collect data for SI it needed to be calibrated. This was done with the method explained in Subsection 4.1.4. The result of the IMU calibration are presented in the figures and tables below.

Table 5.1 shows *rmse* values to compare the uncalibrated and calibrated IMU compared to an optimal IMU. An optimal IMU is an IMU that reads the acceleration without errors. In addition to the *rmse* values, the result is also shown in 5.3(a) and 5.3(b). These figures show the uncalibrated and calibrated imu acceleration in a X-Y and X-Z graph.

Table 5.1: The RMSE values for the Non calibrated IMU and the calibrated IMU compared to an optimal IMU.

Axis	Non Calibrated	Calibrated
X	0.3369	0.0854
Y	0.2381	0.2702
Z	0.7861	0.1525



(a) XY-Axis

(b) XZ-Axis

Figure 5.3: Non-Calibrated and Calibrated values.

Another way of verifying an accelerometer is to move it a known distance and integrate the result into position. This test is done and the result is shown in Figure 5.4.

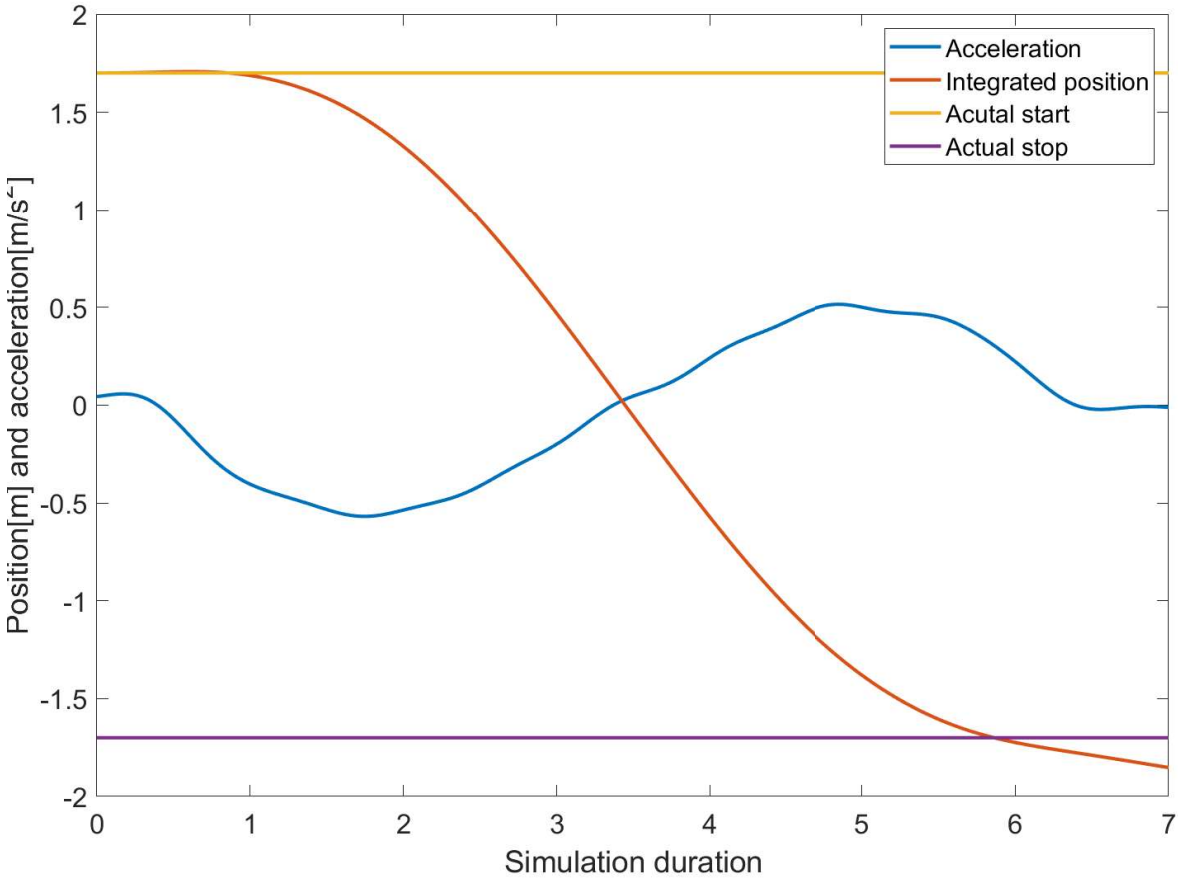


Figure 5.4: Acceleration integrated to position.

5.1.4 IMU alignment

To check that the IMU is aligned with the coordinate system of Sim IV, an FFT analysis is done on the X,Y, and Z axis. If the frequency of the X data is not present in the Y and Z data the coordinate systems are well aligned. These test results are shown in Figure 5.5.

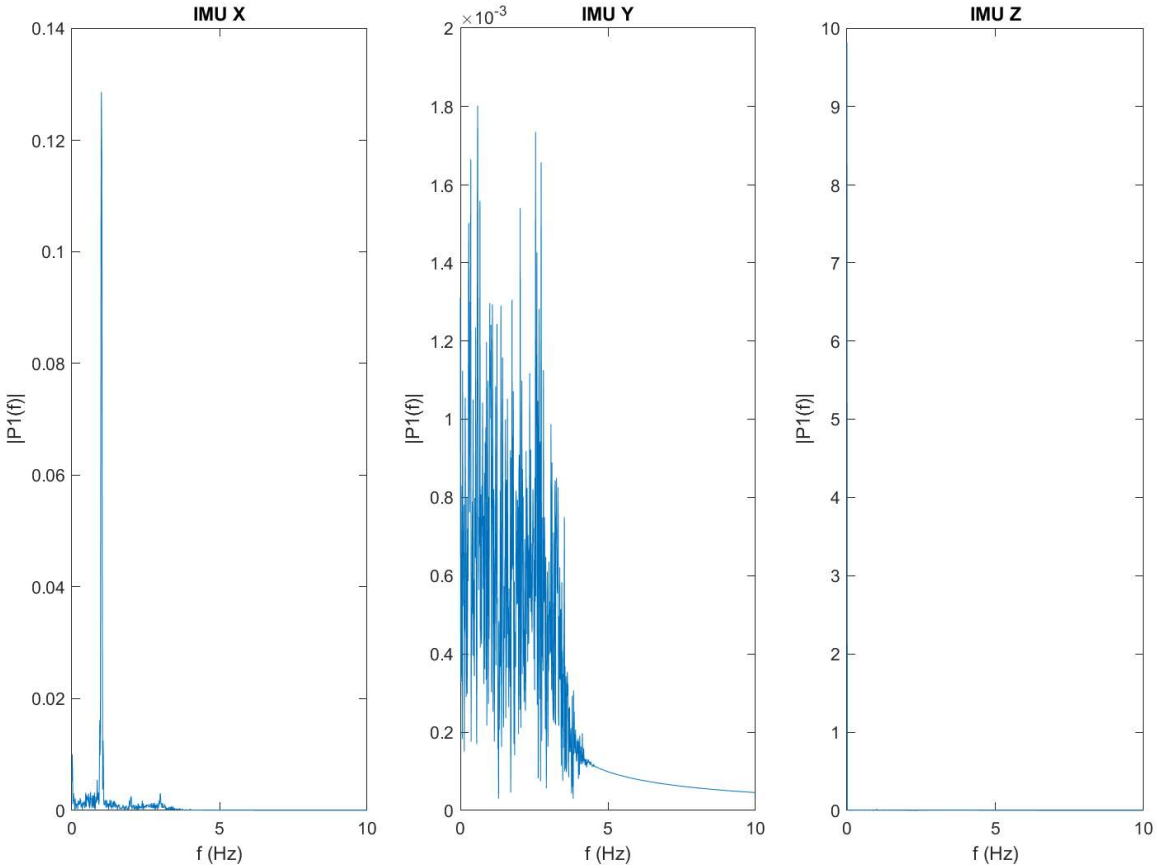


Figure 5.5: FFT och IMU-accelerations for a test with frequency 1 Hz and amplitude 1 m/s^2 in X-direction. (Note the different scales in the y-axis of the plots).

5.1.5 Time difference

The time difference for each test is recorded as described in the method section (Subsection 4.1.3.3). This best and worst time difference is shown in Table 5.2. All the test's time differences are shown in Appendix A at Table A.1.

Table 5.2: The best and worst test with regard to the time difference.

Test Number	Mean Time diff[ms]	Time diff max-min[ms]	Category
813	-5905.26	2.4102	Largest Time diff
947	8279.40146	0.81055	Smallest Time diff

The data *max-min* is the delta within the test, from the largest time difference to the smallest logged. For all tests, the mean delta is 1.2636 ms and the median delta is 1.2470 ms.

5.1.6 IMU-Placement

The result of the test with another placement is shown in Figure 5.6.

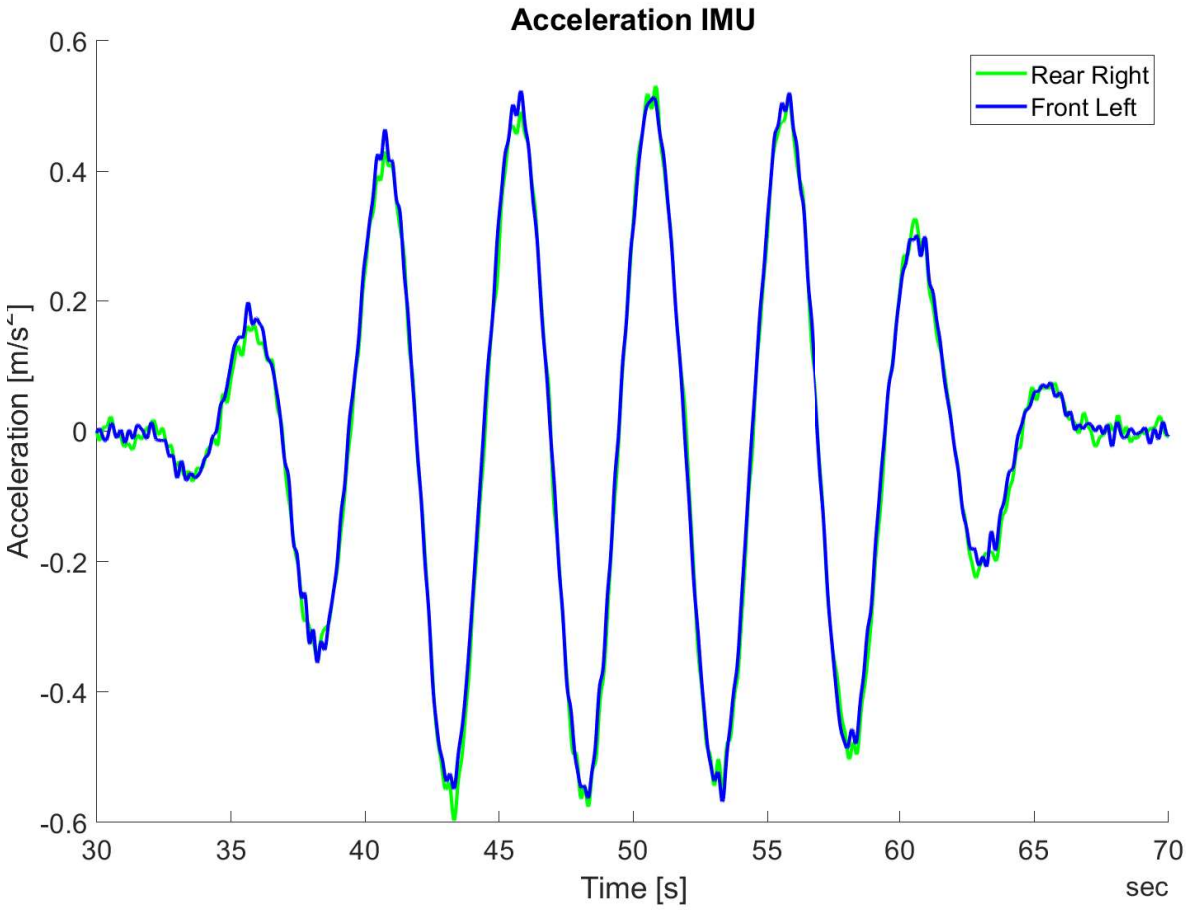


Figure 5.6: IMU accelerations for the two different placements.

5.2 System Identification

The purpose of the data collection was to use it for SI. SI requires data that exposes the dynamics of the system. This section will present the result of the SI.

5.2.1 Data-sets

In total, 10 different tests were used for model development, and six tests were used for model validation. Since SI is an iterative process, the datasets were collected during multiple test sessions.

5.2.1.1 Itterations

In the first test sessions, five datasets for SI were collected. That was the test numbers $801 - 805$ in Table 4.4 and Table 4.5. The SI models from this test is shown in Equation 5.1 and Equation 5.2.

$$G(s) = \frac{25.33}{s + 23.84} \tag{5.1}$$

$$\begin{aligned} \dot{x} &= -11.93x + 0.64u \\ y &= 25.48x \end{aligned} \tag{5.2}$$

The fit percentage for the Transfer Function (Equation 5.1) with this data is 79.98% and for the State Space (Equation 5.2) 65.32%.

Figure 5.7 shows the bode plot for the TF and State-space model, compared to the bode plot from the Bosch installation of Sim IV.

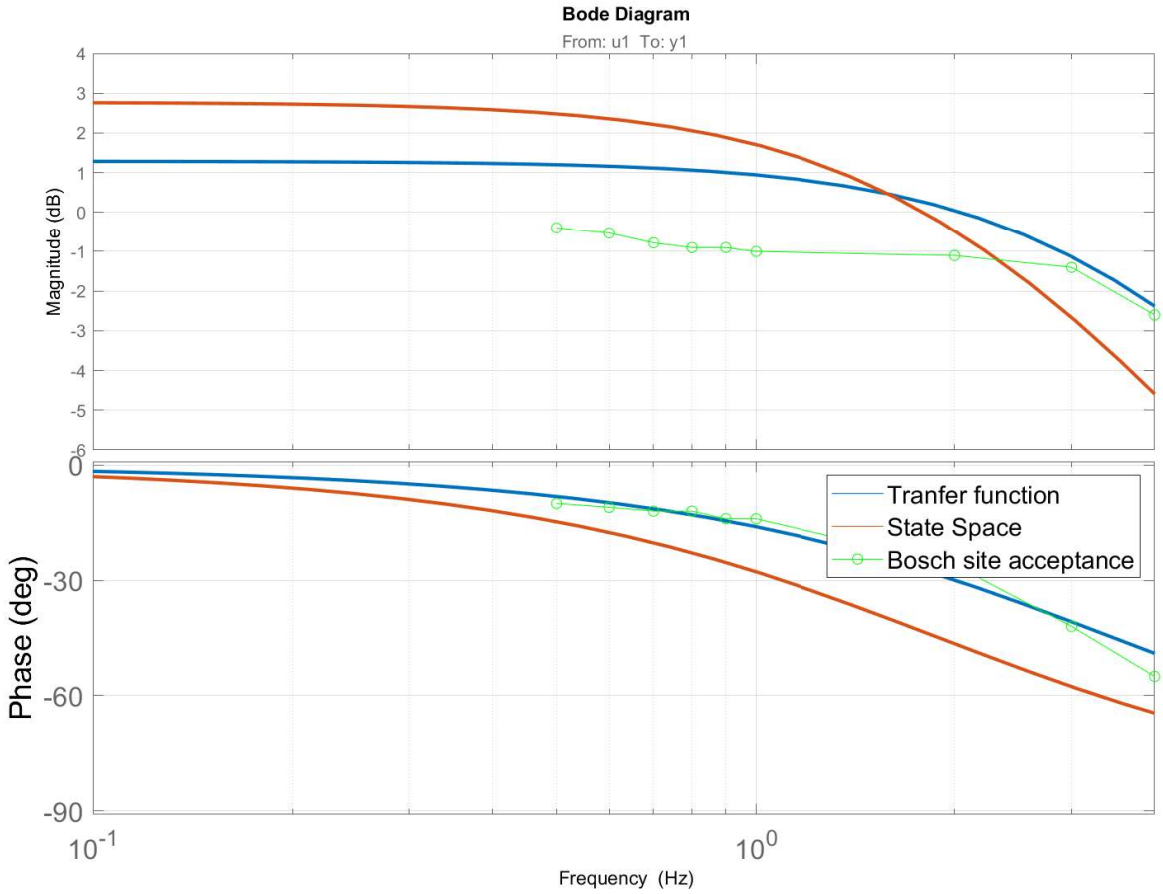


Figure 5.7: Bode plot for the first iteration of system identification

The result for changing to a 2^{nd}-order model resulted in the following fit the checking the cross-validation. For the transfer function, the fit was 86.57 % and for the state-space the fit was 79.18 %.

5.2.2 Final Model

To better expose the dynamics of Sim IV, the remaining tests in Table 4.4 and Table 4.5 were done and added to the SI-data. The model was then updated to the final models shown in Equation 5.3 and Equation 5.4.

$$G(s) = \frac{23.82}{s + 23.46} \tag{5.3}$$

$$\begin{aligned} \dot{x} &= -18.87x + 0.64u \\ y &= 30.29x \end{aligned} \tag{5.4}$$

The fit percentage for the Transfer Function (Equation 5.1) with this data is 87.32% and for the State Space (Equation 5.2) 86.02%.

In Figure 5.8 the Bode plot for the final iteration is shown, for both the TF and state-space models.

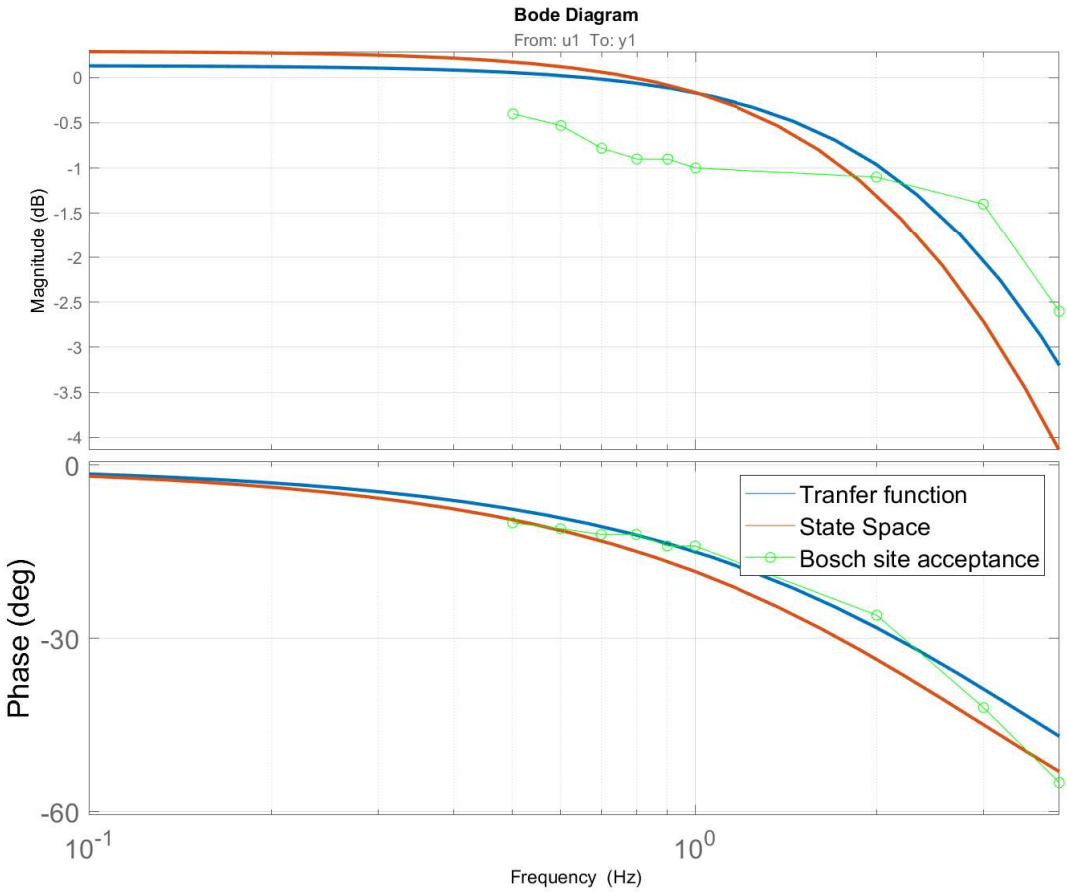


Figure 5.8: Bode plot for the final iteration of system identification

Cross-validation is shown for two cases, with an entire plot and a zoomed-in for both tests. The amplitude is 0.5 m/s^2 for both but the frequency is different. For Figure 5.11 and Figure 5.12 a frequency of 2 Hz was used and for Figure 5.9 and Figure 5.10 a frequency of 1 Hz was used.

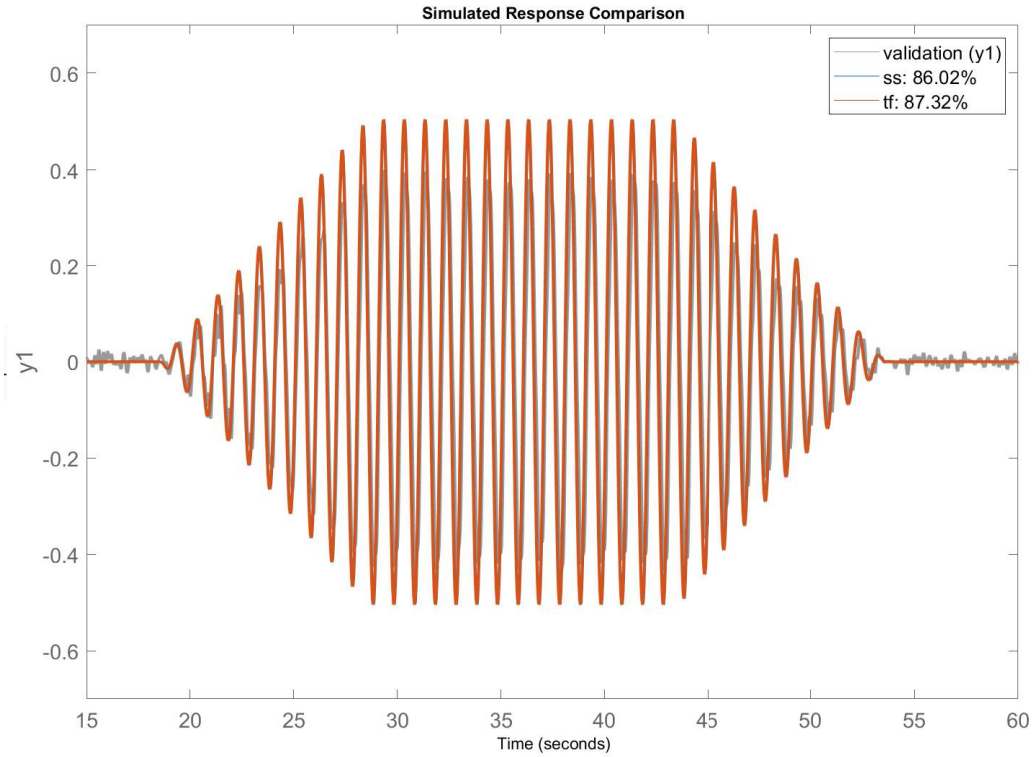


Figure 5.9: Cross-validation for $1Hz$ and $0.5m/s^2$.

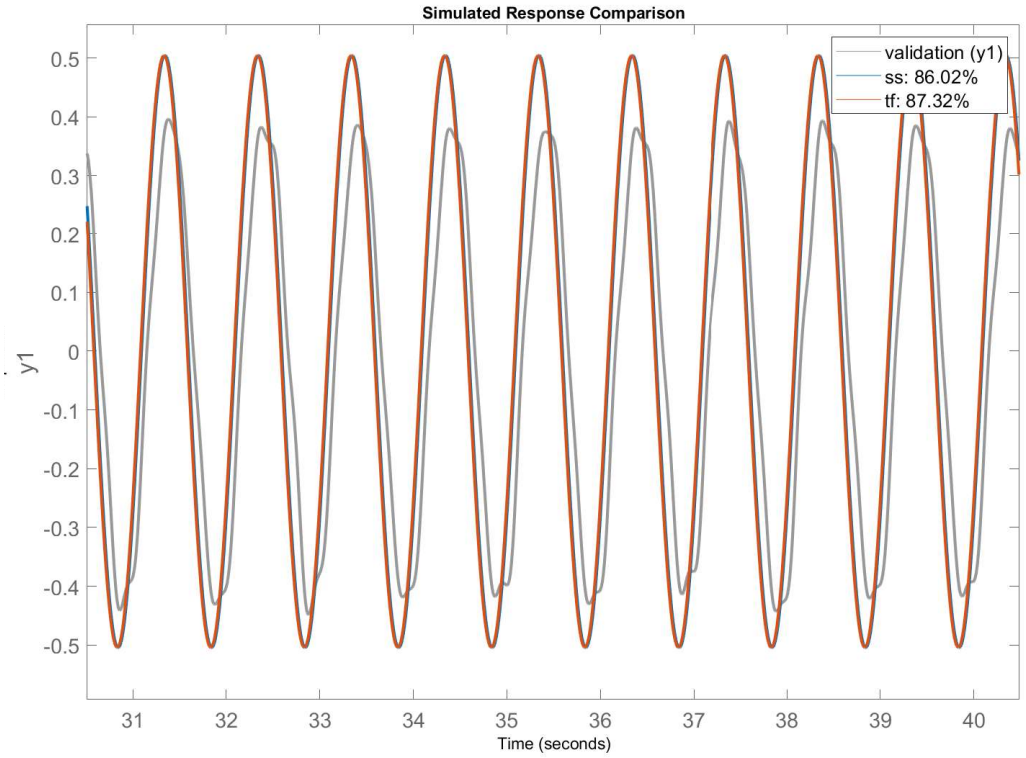


Figure 5.10: Cross-validation for $1Hz$ and $0.5m/s^2$.

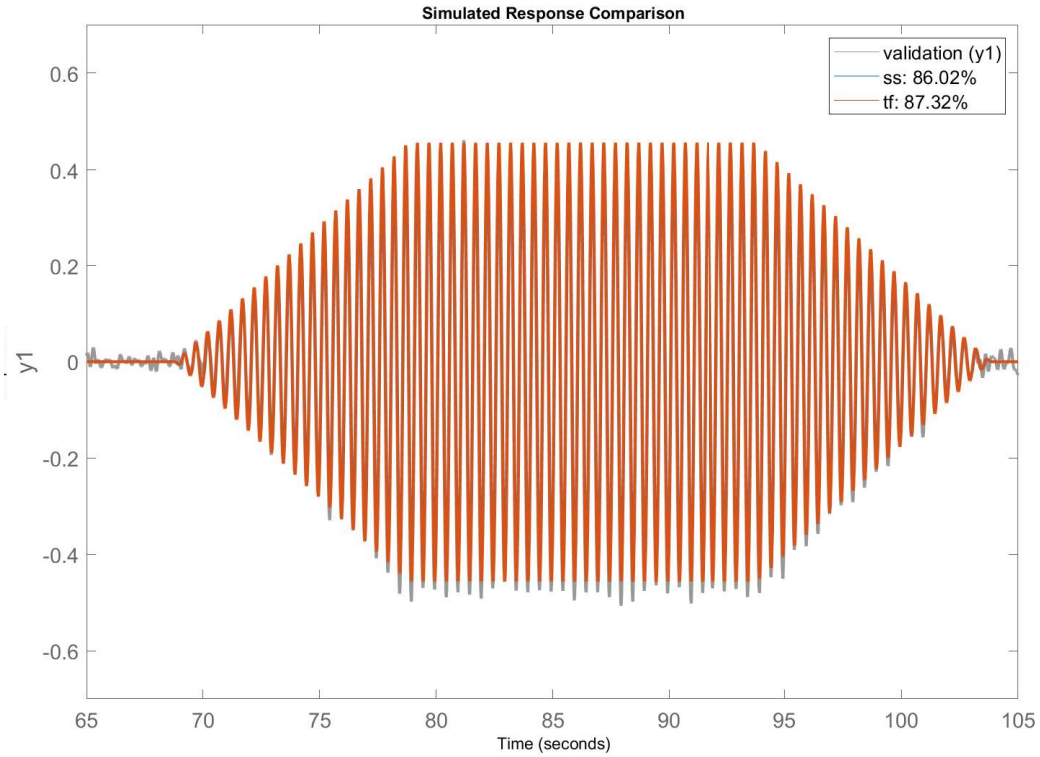


Figure 5.11: Cross-validation for $2Hz$ and $0.5m/s^2$.

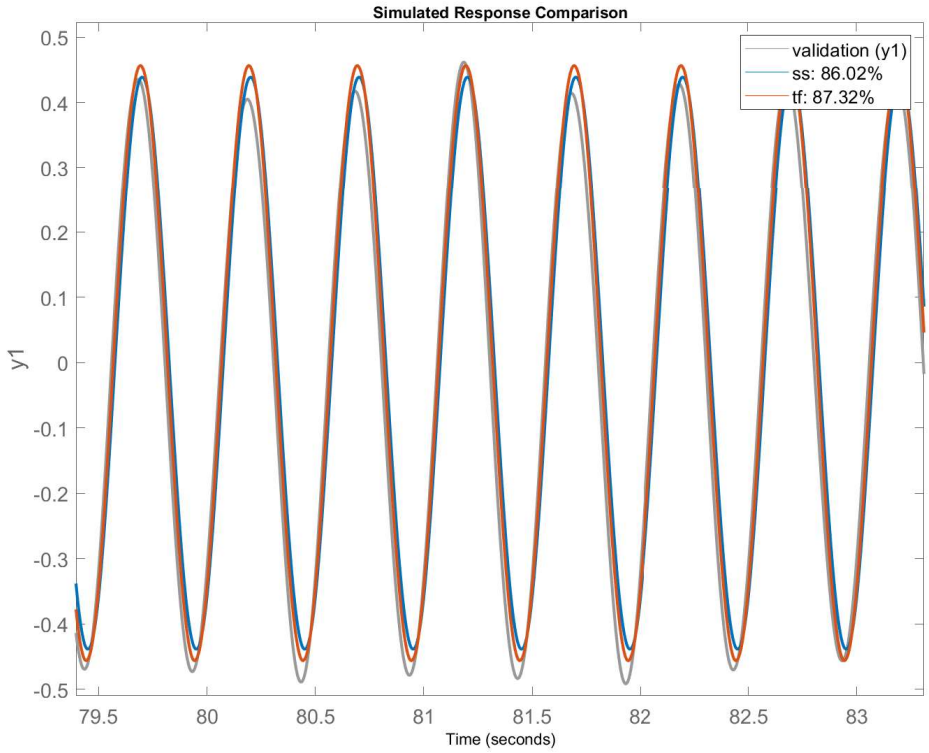


Figure 5.12: Cross-validation for $2Hz$ and $0.5m/s^2$.

For the prediction Algorithm, the model that will be used is the TF. This decision is based on both the model-fit and the Bode plots

5.3 Prediction algorithm

The prediction algorithm that are implemented in the motion cueing is based on the Smith predictor approach, as described in Section 4.3. The controller is a PI-controller that is tuned with the Matlab *pidTuner* application. The control parameters that are used are $K_p = 2.56$ and $K_i = 109.99$.

With the TF-model from Subsection 5.2.2 and the controller from the above paragraph, the transfer function for the negative closed-loop system is shown in Equation 5.5.

$$G(s) = \frac{22.25s}{s^2 + 79.71s + 2447} \tag{5.5}$$

From the transfer function in Equation 5.5, the poles for the closed-loop-poles are presented in Equation 5.6.

$$\begin{aligned} p_1 &= -39.8554 + 29.3076i \\ p_2 &= -39.8554 - 29.3076i \end{aligned} \tag{5.6}$$

5.4 Validation of the Prediction algorithm

For validating the prediction algorithm, a total of 19 tests were done. The result from that tests is presented in this section, and in Table B.1, Table B.2 and Table B.3.

In Table 5.3, the statistical metrics for the PA validation is shown. The definition of *Latency* is explained in Figure 4.22. The metric *Difference to goal* is explained in Figure 4.23

Table 5.3: The statistical metrics for the validation of the prediction algorithm.

Test type:	**No controller**	**No controller**	**Controller**	**Controller**
Data source	**IMU**	**FDBK**	**IMU**	**FDBK**
Mean Latency	0.0805	0.117	0.0434	0.0755
Median Latency	0.075	0.115	0.035	0.07
StDiv Latency	0.0152	0.009770764	0.0195	0.0145
Amplitude:	**Diff to Ref**		**Diff to Ref**	
Mean	0.0759		0.0778	
Median	0.0505		0.062	
s	0.0635		0.0531	

6

Discussion

In this thesis, some limitations were set. These limitations will be addressed in this section. The section will also discuss, the results and future work.

For a better improvement of driver experience in SimIV, it would be beneficial to complete all three possible ways of producing longitudinal acceleration. In this thesis, the work was limited to the x-table but the hexapod can produce longitudinal acceleration with both linear motion and a rotational motion with tilt-coordination (rotating the driver to use gravity to produce sustained longitudinal accelerations).

In addition to the work done with the MS, combining this work with the delays in the VS would also be a good next step. It could enable the idea presented by Cruden of "zero perceived latency" [33]. This idea is a concept of minimizing and synchronizing the MS, and VS with prediction algorithms.

6.1 IMU

When working with Arduino, Raspberry Pi, and IMU-board provided good learning about the benefits and drawbacks of using cheap and low-power hardware instead of a commercial IMU. When switching from the Arduino to the Raspberry Pi it became clear that using hardware that is not on the limits computational-wise makes the work of programming it much easier. If the Raspberry Pi would have been used from the beginning, many hours of work could have been saved.

The benefit of using this hardware is mainly the low cost of acquiring it, the large open-source community if support is required, and the possibility of large customization since the user writes the code for the specific use-case. One drawback with this method instead of buying a commercial product could be the uncertainty of the quality of the measurements, and the work by the user to calibrate and verify readings.

The Calibration results of the IMU showed a large improvement for the X and Z accelerometers but the results did not improve for the Y-accelerometer as shown in the table and figures at Subsection 5.1.3. Since this project focused on the X-direction, the Y-accelerometer was not recalibrated. The reason for not trying to recalibrate the Accelerometers was both due to the time limits of the thesis and to not risk making the X-calibration worse. In addition to the Plot and, *RMSE*-values for the calibration, the test (Figure 5.4) of moving the accelerometer a fixed distance

also showed that the X-direction was successful. For future work, an idea would be to recalibrate the IMU and redo the tests shown in Figure 5.4 for all DOF:s.

The future use of the IMU developed for SimIV, could have an improved user experience by integrating the IMU operation from the standard VTI simulation control. As the IMU control currently works by a separate terminal (shown in Figure 5.1) besides the simulation controls. Removing this separate interface would make it easier for other users to log IMU data when performing Simulator studies. Another improvement would be to activate the gyroscope's 3-DOF:s in the Python code that reads the IMU-board. This data was left out due to concerns about computational performance at the start, but the Raspberry Pi4 would probably be sufficiently fast for this data too.

6.1.1 Placement

As shown in Subsection 5.1.6, the IMU was moved and one test was rerun. This test shows that the longitudinal acceleration is very similar between the two placements. The difference visible in Figure 5.6 is probably the noise of the two signals. In the future, this test needs to be done for rotational accelerations too. It could also be beneficial to test the accelerations at other places on Sim IV to check that this behavior is consistent.

The reason to only check two positions is that the IMU bracket is only designed to be mounted at the beams underneath the car. The decision was when to move the IMU diagonally to maximize the position change.

6.1.2 Time syncronisation

A big concern when deciding to use an IMU to collect the SI-data was how to synchronize the time. In this thesis working with latency in *milliseconds*, accurately syncing the time of the collected data is very important. First, the idea was to sync all computers to one local *NTP-server* and use ntp-protocol for handling the task of syncing the time. This method was then changed to one consisting of recording the time differences and adjusting the time in the data afterwards to compare to the references and commands at each certain time-stamp.

The main benefit of this method was to avoid changing the time setting at the SimIV-kernel and risking potential complications in communications with other systems. The second benefit is that this method makes it possible to check that no time drift happens during the test and if this occurs, re-run the test.

From the information in Table 5.2 and Table A.1 all tests except 813 have a max delta for time difference within the test of less than $2\,ms$. In test 813, the delta is $2.4\,ms$ which still is smaller than half the sampling time. For all tests, the mean value is $1.2636\,ms$. From this average delta value, it is clear that the method of taking a mean value for the whole test session works, but it is also important to know that no delay results have higher accuracy than the delta value for each test.

In the future, it could be interesting to investigate what differences a higher fre-

quency for logging the system times would have on the result. Is it possible to use a higher frequency for applying an adaptive time adjustment?

6.2 System Identification

SI was used to develop the model of SimIV since developing a physically derived model would be very time-consuming.

6.2.1 Data

In total 16 tests were used for SI, with ten for development and six for validation. This split was done to have tests with a broad frequency range for both model development and model validation. This decision of using ten tests for the development and six was the trade-off described in Subsection 2.4.1.3 that some tests need to be spared for cross-validation. If more time for collecting data would have been available, continuing the tests with $+0.1\,Hz$ up to the end frequency of $3\,Hz$ could be a method for collecting more data-sets.

The decision to use a pure sinusoid method was an easy decision to make sure that the simulator stayed within its physical boundaries. If more tests would be done, it would be interesting to develop a way of using other types of signals and test if that would expose the dynamics of SimIV better. A common input is as described in Subsection 2.4.1.1 a *RBS*. If it is not possible to use this signal, doing sinusoid tests that go closer to the maximum acceleration could provide knowledge of the simulator's maximum capabilities.

6.2.2 Model

The main criterion for validating the model was cross-validation with other data-sets. The metric used for determining which model structure that should be used the developing the PA was the *model-fit*. Using the *model fit* provided the benefit of clearly seeing if the model becomes better when more data was added. As described in Subsection 5.2.1.1, when more data was added, the transfer function model was not changing much, indicating that the dynamics were well exposed by the tests, or that the tests is not exposing the full dynamics of the system.

The tests done with SI-models of higher order did not result in a better result when checking cross-validation. From this, the decision was to stick with a simpler 1^{st}-order model. Reasons why the fit did not improve could be that the system is of 1^{st}-order or that the inputs do not expose any 2^{nd}-order characteristics.

As seen in Figure 5.11 and Figure 5.9 the model is overestimating the output when the frequency is low and underestimates the output when the frequency is high. This will impact the system output, as discussed in Subsection 6.3.1.

6.2.3 Development of acceleration controller

When developing the controller, the controller was a *PID* tuned with pidTuner with $k_d = 0$ (representing a PI-controller). Choosing a more advanced controller could be a better strategy for better results. It would be interesting to compare the developed *pi*-controller to an LQG or MPC controller.

The chosen way of implementing the controller was as the model part of a Smith predictor, but with lacking the delayed feedback loop.

6.2.4 Implementation of the predictor in SimIV

When implementing the controller in the MC-Simulink model, the existing interface for sending acceleration commands was used. This placed the acceleration in the last part of the model before the signal leaves the SimIV-kernel. That implementation was good for sending pre-determined acceleration commands as utilized in this thesis.

For future use in combination with a vehicle model, it could be beneficial to move the controller to the part of the model that calculates the acceleration reference. If the placement used now would be used when driving the simulator, it would require some extra differentiation and integration to have the $\int\int a(t)dt = \int v(t)dt = p(t)$ as the right parameter in the right place.

6.3 Validation of the Prediction-Algorithm

The PA was validated with the same type of tests as used for collecting the SI-data. Using I/O-tests for validating the PA made the result objective since no human driver was involved. From this data (Table 5.3), it is possible to see that the controller lowers the latency with $\approx 40\,ms$ or $8t_s$. The original latency between a reference and the IMU was $\approx 75\,ms$ or $15t_s$. To ensure that the result was correct, the same tests were done with the feedback too. Doing tests with the feedback signal takes out errors that could occur from a potential inaccuracy in syncing the times.

When checking latency for the feedback, the same trends as for the IMU-acceleration are found. The controller lowers the latency with $\approx 45ms$, from $\approx 115ms$ to $\approx 70ms$. Worth noting is that the standard deviation is larger for the IMU. When checking the tests in Table B.1 and Table B.2 it is clear that the spread is larger for the IMU tests than the feedback tests. One reason for this could be that the IMU signal is containing more noise than the feedback signal, the noise in combination with the large t_s of $5\,ms$ could affect the latency calculations. In the future, it could be interesting to increase the sampling rate of the IMU to check if the result would have less spread.

The next step done was to check how the acceleration amplitude changed for the different tests and what the difference to the goal acceleration amplitude was. As seen in Table B.3, the result is not clear, no control was better in eight tests and control was better in eight tests. When calculating the mean and median values for the difference, no control has a smaller value in the two metrics.

For high-frequency tests, the $3Hz$-test is in particular increasing the average value, especially if the $3Hz$ is taken out of the metrics test is taken out the metrics are very close. One reason for this difference could be that K_i in the controller is too large. More tuning of the controller could result in a more clear difference in some/all tests.

The next step after implementing the controller would be to design a simulator study focused on longitudinal acceleration and test if the improved acceleration tracking also improves the driving experience in SimIV.

6.3.1 Mathematical evaluation of Prediction-algorithm

As the PA was implemented in SimIV, the SI-model was "simulating" a feedback loop. By this, the control error is $\tilde{e} = r - \tilde{y}$. For a normal feedback loop as shown in Figure 6.1, the loop transfer function is the one shown in Equation 6.1.

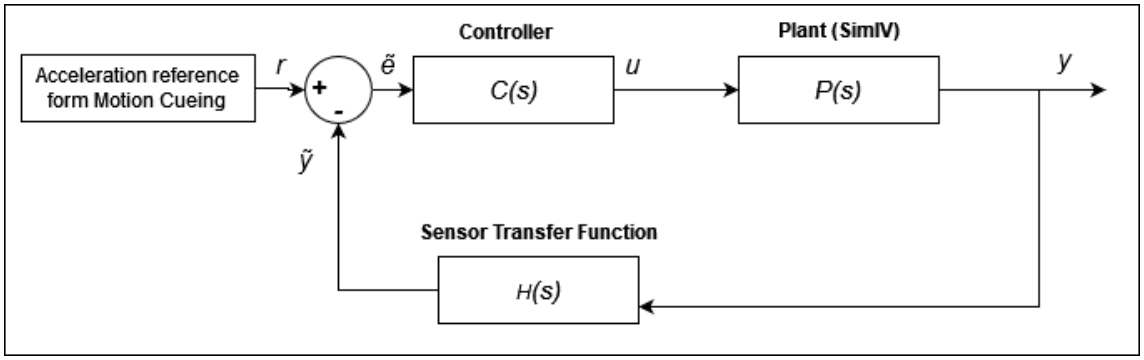


Figure 6.1: The block schedule if a proper feedback loop would be used.

$$\frac{Y(s)}{R(S)} = \frac{C(s)P(s)}{1 + C(s)P(s)H(s)} \tag{6.1}$$

For the implementation in this thesis, the structure is instead shown in Figure 6.2.

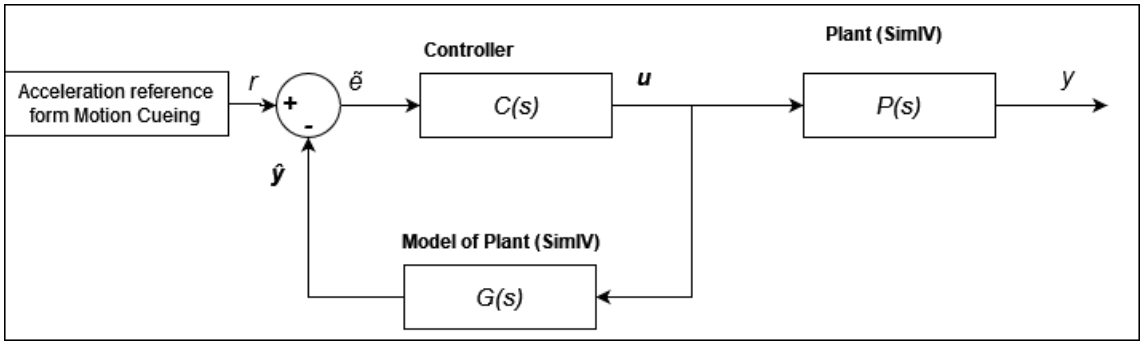


Figure 6.2: Controller used in this thesis.

For the used simulated loop and case $G(s) \neq H(s)P(s)$ the transfer function for the loop will need to be derived. this derivation is done in Equation 6.2-Equation 6.8. The term $E(s)$ is the Laplace-transform of the control error.

$$Y(s) = P(s)C(s)E(s) \tag{6.2}$$
$$E(s) = R(s) - \hat{Y}(s) \tag{6.3}$$
$$\hat{Y}(s) = G(s)U(s) = G(s)C(s)E(s) \tag{6.4}$$
$$\implies E(s) = R(s) - G(s)C(s)E(s) \tag{6.5}$$
$$E(s) = \frac{R(s)}{1 + G(s)C(s)} \tag{6.6}$$
$$\implies Y(s) = \frac{P(s)C(s)R(s)}{1 + G(s)C(s)} \tag{6.7}$$
$$\implies \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + G(s)C(s)} \tag{6.8}$$

From Equation 6.8 it is clear that if an optimal model would be developed, *i.e* that $G(s) = H(s)P(s)$ the transfer function would instead be the same as in Equation 6.1.

$$\frac{Y(s)}{R(S)} = \frac{C(s)P(s)}{1 + C(s)G(s)} \tag{6.9}$$

7

Conclusion

After finalizing this thesis, the three main research questions were answered. The main questions are the following, is the existing feedback a good data source when doing experiments for System Identification, or is an IMU a better choice? The second question is, how well are a 1st-order model representing the dynamics of SimIV. The third question is, could this 1st-order model be used to lower the acceleration latency?

To answer the first question early in the project, a prestudy of existing experiments was done to determine whether the Bosch feedback was good. The method used was to compare *RMSE, variance* between IMU, Command, and the Feedback signal. The result of this study is that the frequency and amplitude of the acceleration have a large impact on the IMU and the feedback values. From this, the decision was to use the IMU signal instead of the feedback for system identification.

The second question was to investigate how well the system identification model will represent the I/O dynamics of SimIV. The model was developed by collecting acceleration data with the IMU and using the acceleration command as the input. The final model was a transfer function that is shown in Equation 7.1.

$$G(s) = \frac{22.25}{s + 22.84} \tag{7.1}$$

For the Validation tests that were used to determine the *fit of the model*, this model has the highest percentage. The fit for this model was 87.32% in this test, in addition to this structure, a *state-space* was also tested, but this structure had a lower fit of 86.02 %.

The last question was to investigate if this model could be used to lower acceleration latency in SimIV. To lower the latency a controller was implemented in the motion cueing model, this controller implements the model as is done with parts of a *Smit-predictor*. This means that the model will be used as a *virtually closed loop* instead of using actual sensor data from the plant.

For testing this controller, both tests with synthetic sine wave acceleration and acceleration from driving in SimIV were used. These tests were run with the controller on and off to see what the difference will be. The difference measured for this data was a lowering, from a median of $80ms$ to $35ms$. This reduction is the result of the controller parameters that were used at the end of this project.

Bibliography

[1] Donald Fisher et al. *Handbook of driving simulation for engineering, medicine, and psychology*. 2011-05, pp. 1–752.

[2] Martin Fischer, Håkan Sehammar, and Göran Palmkvist. "Motion cueing for 3-, 6- and 8- degrees-of-freedom motion systems". In: 2010.

[3] Z. Fang, G. Reymond, and A. Kemeny. "Performance identification and compensation of simulator motion cueing delays". In: *Journal of Computing and Information Science in Engineering* 11.4 (2011). DOI: 10.1115/1.3622751.

[4] Liwen Guo et al. "New Predictive Filters for Compensating the Transport Delay on a Flight Simulator". In: 2004.

[5] Woon-Sung Lee. "Compensation of transport delay in driving simulator". In: *Proceedings of the American Control Conference*. Vol. 1. 1991, pp. 1050–1055. DOI: 10.23919/acc.1991.4791538. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0026372322&doi=10.23919%2facc.1991.4791538&partnerID=40&md5=f367bda3eea4357780c3a4f17282a621.

[6] C.M. Oman. "Motion sickness: A synthesis and evaluation of the sensory conflict theory". In: *Canadian Journal of Physiology and Pharmacology* 68.2 (1990), pp. 294–303. DOI: 10.1139/y90-044.

[7] Jeroen H Hogema. "Compensation for Delay in the Visual Display of a Driving Simulator". In: *SIMULATION* 69.1 (1997), pp. 27–34. DOI: 10.1177/003754979706900103. URL: https://doi.org/10.1177/003754979706900103.

[8] Adafruit Industries. *Adafruit 9-DOF IMU breakout - L3GD20H + LSM303* . URL: https://www.adafruit.com/product/1714.

[9] Soumendu Sinha et al. "Design and Simulation of MEMS Differential Capacitive Accelerometer". In: 2014-05. DOI: 10.13140/2.1.1074.8809.

[10] Francis Mulloy, Olivia Brown, and David R Mullineaux. "COMPARISON OF SIMPLE GRAVITY BASED ACCELEROMETER CALIBRATION PROCEDURES". In: (2019-07). URL: https://commons.nmu.edu/isbs/vol37/iss1/124.

[11] Mahmood Hassan and Qilian Bao. "A Field Calibration Method for Low-Cost MEMS Accelerometer Based on the Generalized Nonlinear Least Square Method". In: *Multiscale Science and Engineering* 2 (2020-05), pp. 1–8. DOI: 10.1007/s42493-020-00045-2.

[12] T. Ylonen and Ed C. Lonvick. *RFC 4251 - The Secure Shell (SSH) Protocol Architecture*. 2006-01. URL: https://datatracker.ietf.org/doc/html/rfc4251.

[13] Marc Wilson. "SCP - Secure Copy Protocol - What is it & Full Definition & Example Cmds!" In: *PC & Network downloads* (2023-06). URL: https://www.pcwdld.com/what-is-scp#wbounce-modal.

[14] *GitHub - jbardin/scp.py: scp module for paramiko*. URL: https://github.com/jbardin/scp.py.

[15] Jeff Forcier. *Welcome to Paramiko's documentation! — Paramiko documentation*. URL: https://docs.paramiko.org/en/stable/.

[16] Lennart Ljung. *System Identification: Theory for the User*. Vol. 2. Prentice Hall PTR, 1999.

[17] Brian Douglas. *System Identification*. 2021-01. URL: https://se.mathworks.com/videos/series/system-identification.html.

[18] Lennart Ljung. "System Identification Toolbox for use with MATLAB". In: *Mathworks* (2011). URL: https://www.researchgate.net/publication/37405937_System_Identification_Toolbox_for_use_with_MATLAB.

[19] Bengt Lennartson. *Reglerteknikens grunder*. 4th ed. 2002.

[20] Didac Recio. *Smith Predictor: when to use it and how to tune it*. 2022-09. URL: https://blog.incatools.com/advanced-process-control-pid-tuning-is-the-first-step-3-0.

[21] A.Terry Bahill. *A simple adaptive Smith-predictor for controlling time-delay systems: A tutorial*. 1983-06. URL: https://www.researchgate.net/publication/3206982_A_simple_adaptive_Smith-predictor_for_controlling_time-delay_systems_A_tutorial.

[22] Arun Kumar. "Subjective perception and prediction model of vehicle stability under aerodynamic excitations". PhD thesis. Gothenburg: Volvo Cars, 2021, pp. 25–34.

[23] DEWESoft. "DS-IMU/GYRO SOFTWARE USER MANUAL". In: (2022-10).

[24] Arduino Team. *Arduino Uno Rev3 — Arduino Official Store*. URL: https://store.arduino.cc/products/arduino-uno-rev3.

[25] Raspberry Pi foundation. *Buy a Raspberry Pi 4 Model B – Raspberry Pi*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[26] Guyaros. *Adafruit 10DOF IMU breakout board Enclosure by Guyaros - Thingiverse*. 2016-04. URL: https://www.thingiverse.com/thing:1513726.

[27] Harryb1991. *Raspberry Pi 4 case with break-out GPIO tab + fan versions by Harryb1991 - Thingiverse*. 2019-07. URL: https://www.thingiverse.com/thing:3735600.

[28] amason101070. *Arduino Uno Case for Adafruit Ethernet Shield by amason101070 - Thingiverse*. 2017-11. URL: https://www.thingiverse.com/thing:2634316.

[29] *Creative Commons — Attribution 4.0 International — CC BY 4.0*. URL: https://creativecommons.org/licenses/by/4.0/.

[30] Adafruit Industries. *Connecting It Up | Adafruit 9-DOF IMU Breakout | Adafruit Learning System*. 2014-02. URL: https://learn.adafruit.com/adafruit-9-dof-imu-breakout/connecting-it-up.

[31] Kajsa Weibull et al. "Introduction to Emergency Vehicle Approaching to decrease delay". In: (2023).

[32] Alexander Hägglund. *brynasalex*. 2023-06. URL: https://github.com/brynasalex.

[33] Jelle van Doornik et al. “Implementing prediction algorithms to synchronize and minimize latency on a driving simulator”. In: *Proceedings of the Driving Simulation Conference 2016 Europe*. Ed. by Andras Kemeny et al. Paris, France, 2016, pp. 51–59.

A

Appendix 1

Table A.1: Table over the time differences SimIV-kernel - Raspberry Pi4

Test Number	Mean Time diff [ms]	Max Time diff - Min Time Diff [ms]
801	-47.4468231	1.2935
802	-47.5749969	0.8936
803	-47.7156677	1.4492
804	-47.5982819	1.2727
805	-47.1544495	1.1548
806	-46.7591095	1.0559
807	-44.8070984	0.9360
808	-44.4839325	1.3547
810	-44.9228668	1.3855
812	-5902.07562	1.4692
813	-5905.26067	2.4102
880	10582.6139	1.1262
881	10581.4596	0.9517
882	10580.217	1.3889
883	10579.1419	1.1711
884	10577.9236	1.637
885	10576.7633	1.1799
886	10575.5663	1.2886
887	10574.2482	1.7966
888	10573.1728	1.3171
889	10572.1025	1.0168
920	10380.0987	1.3596
921	10378.8522	1.2864
923	10146.7946	1.4407
924	10145.1797	0.9834
925	9928.17007	1.0771
926	9926.76929	1.022
927	9923.98007	1.0603
928	9921.48376	1.3306
930	9907.28984	1.1985
931	9902.31772	1.5994

934	8331.08229	1.3025
935	8326.83282	1.2688
936	8325.63678	0.8403
937	8324.11935	1.5637
938	8322.65053	1.1765
939	8321.32376	1.1533
940	8319.49995	1.0764
941	8317.68379	1.1665
942	8316.36688	1.6917
943	8284.55162	1.3916
944	8283.1517	1.0916
945	8281.91479	1.3735
946	8280.65678	1.8357
947	8279.40146	0.8106
948	8278.02867	1.1497
949	8275.65414	1.0078
950	8273.94514	1.5818
951	8272.76968	1.2607
952	8271.32983	1.0176
953	8269.6664	1.1216
954	8268.3643	1.2566
955	8267.05772	1.2375
957	7227.54314	0.9607
958	6725.5529	1.3726
959	6724.25516	1.147

B

Appendix 2

Table B.1: The measured delay between the Reference and the IMU acceleration.

Tests	Delay Ref-IMU no control [s]	Delay Ref-IMU control [s]	Diff [s]
2m/s^2 0.3Hz	0.09	0.04	-0.05
2m/s^2 0.4Hz	0.08	0.04	-0.04
2m/s^2 0.5Hz	0.075	0.03	-0.045
2m/s^2 0.6Hz	0.075	0.055	-0.02
2m/s^2 0.7Hz	0.09	0.04	-0.05
2m/s^2 0.8Hz	0.07	0.025	-0.045
2m/s^2 0.9Hz	0.065	0.03	-0.035
2m/s^2 1Hz	0.065	0.03	-0.035
0.6m/s^2 0.1Hz	0.085	0.035	-0.05
0.8m/s^2 0.2Hz	0.09	0.055	-0.035
1m/s^2 3Hz	0.075	0.03	-0.045
1m/s^2 2Hz	0.07	0.07	0
1m/s^2 1Hz	0.065	0.025	-0.04
0.5m/s^2 2Hz	0.065	0.025	-0.04
0.5m/s^2 1Hz	0.085	0.03	-0.055
0.5m/s^2 0.2Hz	0.105	0.08	-0.025
1st drive from study	0.065	0.03	-0.035
2nd drive from study	0.095	0.075	-0.02
Aggressive drive	0.12	0.08	-0.04

Table B.2: The measured delay between the Reference and the Feedback acceleration.

Tests	Delay Ref-FDBK no control [s]	Delay Ref-FDBK control [s]	Diff [s]
2m/s^2 0.3Hz	0.13	0.09	-0.04
2m/s^2 0.4Hz	0.125	0.08	-0.045
2m/s^2 0.5Hz	0.12	0.075	-0.045
2m/s^2 0.6Hz	0.115	0.07	-0.045
2m/s^2 0.7Hz	0.11	0.065	-0.045
2m/s^2 0.8Hz	0.11	0.065	-0.045
2m/s^2 0.9Hz	0.11	0.065	-0.045
2m/s^2 1Hz	0.105	0.06	-0.045
0.6m/s^2 0.1Hz	0.13	0.085	-0.045
0.8m/s^2 0.2Hz	0.135	0.09	-0.045
1m/s^2 3Hz	0.11	0.07	-0.04
1m/s^2 2Hz	0.105	0.065	-0.04
1m/s^2 1Hz	0.11	0.065	-0.045
0.5m/s^2 2Hz	0.11	0.065	-0.045
0.5m/s^2 1Hz	0.11	0.065	-0.045
0.5m/s^2 0.2Hz	0.135	0.09	-0.045
1st drive from study	0.12	0.12	0
2nd drive from study	0.12	0.075	-0.045
Aggressive drive	0.12	0.075	-0.045

Table B.3: A table over IMU-acceleration amplitude with control, and with no control (NC). The amplitude is compared to the goal amplitude and between NC and C. (Akronyms in the table: Diff to goal amplitude(DG), Control(C), Vo Control(NC), Goal Amplitude(GA), Amplitude(A) and Diff(D)).

Tests	GA $[m/s^2]$	A NC $[m/s^2]$	A C $[m/s^2]$	Diff NC-C $[m/s^2]$	DG NC $[m/s^2]$	DG C $[m/s^2]$
2m/s^2 0.3Hz	2	2.226	2.192	-0.034	0.226	0.192
2m/s^2 0.4Hz	2	2.187	2.155	-0.032	0.187	0.155
2m/s^2 0.5Hz	2	2.137	2.124	-0.013	0.137	0.124
2m/s^2 0.6Hz	2	2.097	2.084	-0.013	0.097	0.084
2m/s^2 0.7Hz	2	2.061	2.068	0.007	0.061	0.068
2m/s^2 0.8Hz	2	2.032	2.052	0.02	0.032	0.052
2m/s^2 0.9Hz	2	1.994	2.038	0.044	0.006	0.038
2m/s^2 1Hz	2	1.957	1.998	0.041	0.043	0.002
0.6m/s^2 0.1Hz	0.6	0.65	0.656	0.006	0.05	0.056
0.8m/s^2 0.2Hz	0.8	0.877	0.875	-0.002	0.077	0.075
1m/s^2 3Hz	1	1.13	1.535	0.405	0.13	0.535
1m/s^2 2Hz	1	0.982	1.126	0.144	0.018	0.126
1m/s^2 1Hz	1	0.952	0.98	0.028	0.048	0.02
0.5m/s^2 2Hz	0.5	0.477	0.54	0.063	0.023	0.04
0.5m/s^2 1Hz	0.5	0.471	0.461	-0.01	0.029	0.039
0.5m/s^2 0.2Hz	0.5	0.551	0.543	-0.008	0.051	0.043

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY