Motorcycle Simulator with Virtual Reality

Final Report

December 22, 2017

Supervised by
Bengt O Eriksson

Cai Jiacheng
Helle Thibault
Hansen Karl
Sten Gustav
Olsson Marcus
Saleh Seid
Yousif Robert
Svensson Rickard

Mechatronics Advanced Course
MF2059
KTH

Abstract

The objective of the project is to build a motorcycle simulator which could offer not only visual and audio stimuli but also physical sensations such as G-forces, motion and rotations. Through the integration of visual, sound and physical movement, an immersive feeling is created for beginners, amateurs trainings and professional athlete alike.

To this end a motorcycle chassis with handlebar is mounted on an hydraulic motion platform that can utilize 6 degree of freedom (Stewart Rig). The visual stimuli of the simulation is provided by a VR headset. A host computer receives the inputs from the rider, calculates the motorcycle dynamics in a simulation model, translates those motions to appropriate coordinates with motion cueing and then sends appropriate coordinates to the motion platform after filtering. The host PC also runs the simulator game which is connected to the VR headset. To connect the different involved entities the Robot-Operating-System was utilized.

The team has successfully managed to create a motorcycle simulator that achieves three main things. The motorcycle is operating in real-time, the sensation is somewhat believable and it has lowered cyber-sickness compared to a stationary simulator. It is the teams hope that the motorcycle simulator can be used not only by motorcycle drivers but also as a valuable tool for the development of motorcycles or other vehicles.

The work was performed by a team of eight master level students attending the Department of Machine Design at KTH, The royal institute of technology.
All files related to the project is available in a GitHub folder called *MC-Simulator-with-VR* [47].

Acknowledgement

We would like to thank our supervisor at KTH, Bengt Eriksson, for the overall support throughout the project, giving us valuable feedback before deadlines.
Thanks to everyone at the KTH Department of Machine Design for helping us during the course. Especially Björn Möller for putting this project together the project and helpings us to order parts for the project which saved us a lot of time. Thanks to Vicki Derbyshire for taking care of all the administration behind this course making it easier for us to focus on the project. Thanks to Lars Hässler for discussing different ideas and welding the steel box together. Thanks to Staffan Qvarnström, Thomas Östberg and KTH prototype assistants for all the electronics and hardware support.
We would also like to thank Robin Palmberg at the CSC department for helping us with everything related to VR and Unity.
Last but not least, a big thanks to all the Mechatronic Master students at KTH for making days and nights, in and outside school much more enjoyable.

Team MCSim
Stockholm, December 2017

Contents

Nomenclature

AC	Alternate Current
API	Application Programming Interfaces
ARP	Address Resolution Protocol
CSC	school of Computer Science and Communication
DC	Direct Current
$DCPS$	Data-Centric Publish-Subscribe
DDS	Data Distributed Service
DDS	Data Distribution Service
DOF	Degrees of Freedom
DRE	Distributed, Real-time and Embedded
FPS	Frames Per Second
GDS	Global Data Space
GUI	Graphical User Interface
$HDMI$	High-Definition Multimedia Interface
IDL	Interface Definition Language
$JSON$	JavaScript Object Notation
MC	Motorcycle
MCC	Motion Control Cabinet
PC	Personal Computer
QoS	Quality of Service
ROS	Robot Operating System
RPM	Rounds per minute
SEK	Swedish Krona
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
VIC	The visualization studio at KTH
VR	Virtual Reality

1 LIST OF FIGURES

List of Figures

2 LIST OF TABLES

List of Tables

3 INTRODUCTION

This chapter gives a short introduction to the project with limitations and requirements.

3.1 Background

Driving a motorcycle can be a thrilling experience but it does come with some risk. It would be interesting to drive a motorcycle in a safe environment where a novice driver could learn how to handle both expected and unexpected situations without the risk of injury. Motorcycles have a complex dynamic behavior that is effected by the design and construction of the motorcycle. So while developing a new motorcycle, it would be interesting to test the "feel" of it before building a prototype. This is something that the industry would gain from. It was from these ideas that this project was founded.

3.2 Project Description

This project was founded to design and manufacture a motorcycle simulator for the Machine design department at the Royal Institute of Technology, Stockholm. The project combines a motorcycle structure attached to a hydraulic motion platform together with a VR-headset, which provides the visual stimuli. The simulation should mimic the forces that the rider feels as close as possible so while driving the simulator the rider would know how it would feel to ride a real motorcycle. The ambition of this project is to develop a simulation that can be utilized in industrial development processes, MC-riding practice and for recreational use.

3.3 Requirements, Delimitations & Risk Assessment

To be able to mimic the behavior of a real motorcycle that the driver could control, several requirements needs to be fulfilled stated in this section.

Requirements

The requirements defined for the system are listed below.

R 1 *Mechanical Structure* The mechanical construction between the motion platform and the motorcycle shall be strong and safe enough for any user.

R 2 *Motion Delay* The delay from user input to motion output shall not exceed 50 ms [12].

R 3 *Visual Delay* The delay from user input to visual output shall not exceed 20 ms.

R 4 *Sound Delay* The delay from user input to sound output shall not exceed 20 ms.

R 5 *VR Immersion* The user should sense an increased immersion while using the VR-headset.

R 6 *Communication Connectivity* The middleware in use shall have API:s for the main applications (Simulink/Matlab, Unity3D and Arduino).

R 7 *System Modularity* The middleware in use shall be modular so that any node easily can be added, removed or substituted.

R 8 *Data Centric Infrastructure* The middleware in use should have the system information (delays, bandwidth, frequency, node activity) centralized for gathering data, debugging and visualization.

R 9 *Motion Cueing degrees* A washout filter for motorcycle simulation shall be made for 6 dimensions: surge, sway, heave, roll, pitch and yaw.

R 10 *Motion Cueing constant accelerations* The washout filter shall create the sensation of constantly accelerating by tilting the user. The tilting motion should be senseless.

R 11 *Motion Cueing - false cues* The washout filter should not introduce false cues (large movements in the other direction than the given acceleration). When returning to zero the accelerations shall be in the other direction but so small that they are not perceived to hinder immersion.

R 12 *Cyber-Sickness* The user should not feel sick while using the simulator with the VR-headset for at least 5 minutes.

R 13 *System Stability* Each subsystem, together with it's communication should not fail when the whole system is running for at least 5 minutes.

R 14 *Driver Interface* A new user shall not feel hindered by the rider interface.

R 15 *Driver Interface Realism* A professional rider should feel that the driver interface behaves like a real motorcycle.

R 16 *Torque Handlebar Feedback* The torque handlebar feedback shall be strong and fast enough to feel realistic when riding.

3.3.1 User Needs

A user of the MC simulator should be able to ride the motorcycle in a virtual environment using a handlebar input controller. The system should also give the user a realistic simulation immersion.

3.3.2 Time Frame

The time frame for the project was between 20 March and 11 December for presentation of the final product with a summer break between 18 May and 28 August.

3.3.3 Risks & Safety

The hydraulic motion platform together with the mechanical structure can be quite dangerous for the rider and people near the simulator. Even if the risk of motion platform failure is very low (it has built-in safety functions), a stunt man was hired during demonstration days to eliminate risks of harm. The stunt man did set-up a safety harness equipment attached to the user that securely hold and prevent the person from falling if he/she would loose control of the motorcycle. During the final presentation an area around the simulator was a setup in which only authorized personnel would be.

3.4 Architecture

The project contains several different components. These are

1. Input from the rider.
   Receives the inputs from the driver, i.e. torque and steering. This information is then feed to the Dynamic model and VR-environment.

2. Dynamic model / simulation model.
   The simulated model that governs the behavior of the MC and outputs behavior to the Stewart motion platform such that the rider experiences the "right" sensation.

3. VR-environment.
   The virtual reality environment. The environment in which the rider is able to look around. The environment is also where the course is simulated.

4. Stewart motion platform.
   The hydraulic motion platform that allows for physical movement. The motion platform is controlled by the motion computer, also known as the motion PC.

5. Communication between all components.

Figure 1 shows the communication between the aforementioned components.

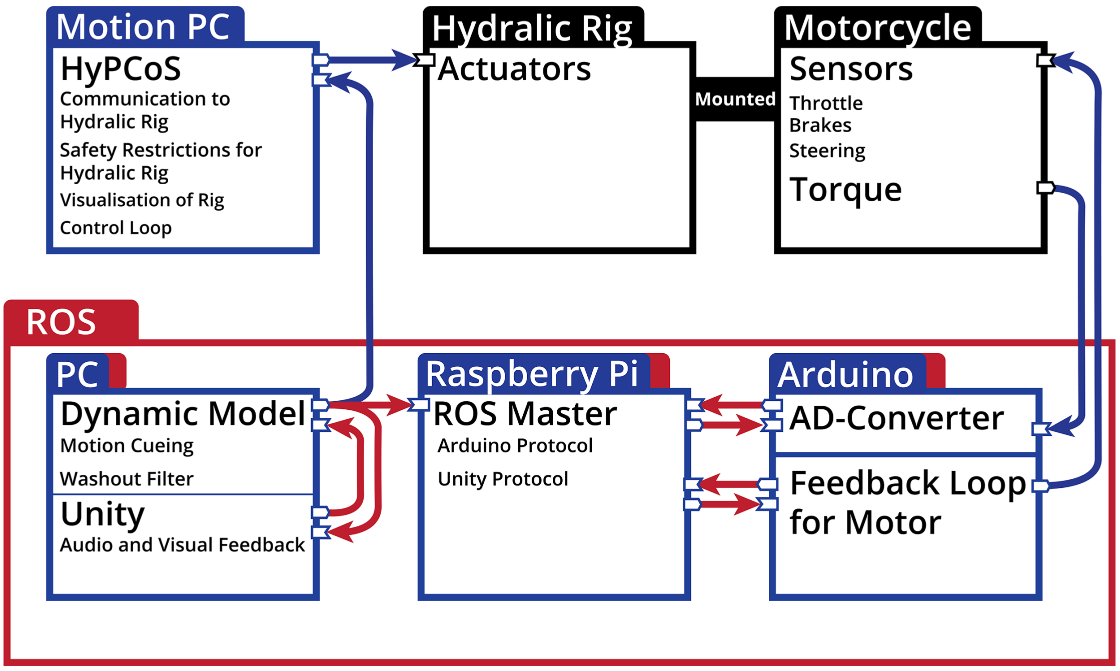


Figure 1: Overview of the system architecture.

Since this project includes several different components it is important to have a main coordinate system to keep the coordinate systems consistent within each component. To handle the confusion that could arise the main coordinate system that can translate between the other different coordinate systems is introduced in Figure 2.

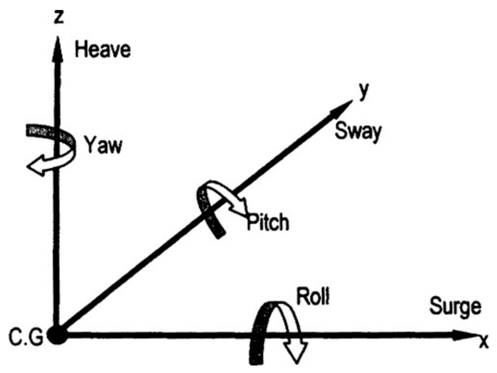


Figure 2: Coordinate systems and rotational axis, movement along the X,Y and Z axises correspond surge, sway and heave respectively.

4 LITERATURE REVIEW & STATE OF THE ART

This part details a state of the art prestudy which explains some background research required to understand the implementations that were done in this project.

4.1 Existing simulators

While automotive and avionic simulators have been developed for decades the field of MC-simulators has been more stagnant. There exists a handful MC-simulator prototypes in the world and many of these are not open to scrutiny due to copyright [50]. In order to find the state of the art for MC-simulator one must look to the more open field of general vehicle simulators.

4.1.1 Flight Simulators

Flight simulators are grouped into categories A-D where D class simulators allow for pilots to attain their license without having to flown a real airplane [1]. Most D class simulators are based on 6 DOF with a full cockpit system mounted on top of the Stewart motion platform. To achieve the visual part of the simulation screens are typically mounted upon the cockpit instead of windows. However, since a Stewart motion platform have inherent limitation on how quick motion can be achieved (and therefor limitations in acceleration) military grade simulators have moved away from Stewart motion platforms in favor of centrifuge simulators(which can generate large accelerations).

4.1.2 Chalmers Vehicle Simulator

Chalmers institute has developed a driving simulator for drive in a four wheel vehicle. The project was started in 1999 but has since been continually developed. The simulator consists of a Stewart motion platform, a partial car (drivers seat, steering and drivers side door) and five computers responsible for the simulation. The ambition of the project is not only to recreate the correct movement but also to create the correct sensation for the driver [25]. To achieve sensations the simulator utilizes a washout filter.

4.1.3 The Moris Motorcycle Simulator

The Moris motorcycle simulator was an early MC-simulator project. The project utilized a Stewart motion platform, a washout filter based motion queuing algorithm and focused on creating a 7 DOF(heave, surge, sway, tilt roll, yaw and steer) motorcycle simulator. The team utilized both tilt-coordination as well as washout-filters the project but also paid attention to sound and vibrations. A mock up MC was placed on the Stewart motion platform with most subsystems intact, only the wheels were removed [4].

4.1.4 Alternatives to Stewart Motion Platform

Another simulator that is available for study is one designed by the authors of Driving simulator [50]. The purpose of the project was to develop a tool for Risk training and behavioral observations (especially in high risk situations) that can be used by novice riders at no risk to themselves, equipment or others. To achieve this the team made the yawing motion a central part of their motorcycle design such that the simulator could accurately reproduce skidding(loss of friction). Another added goal was to make it cheap such that motorcycle driver training centers and schools could use it. The team considered a similar solution to that used in this report, a 6-DOF Stewart motion platform. However, as such a device would make the simulator expensive and therefor the team opted to go with another design. The team attached two power cylinders with spherical joints to the MC chassis and two cylindrical joins to the ground, see Figure 3. This allows for both pitch and rolling motion, without requiring to move the aft part of the chassis.

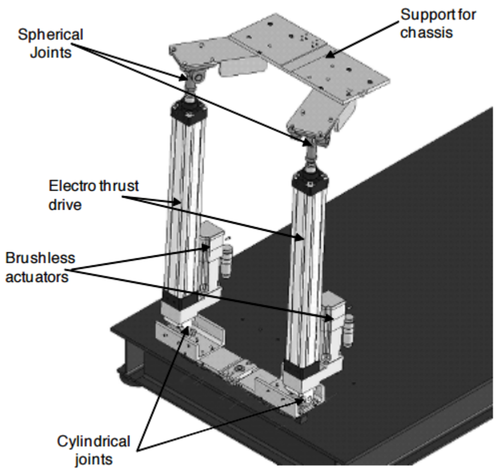


Figure 3: Pitch motion and rolling design [48].

The team attached the aft part of the chassis to a spherical joint attached to a slide which can be moved using a brush-less actuator and belt, thereby creating the yaw motion, see Figure 4. To compensate for the lack of surging motion the team designed the steering handlebar such as to allow for linear displacement [48].

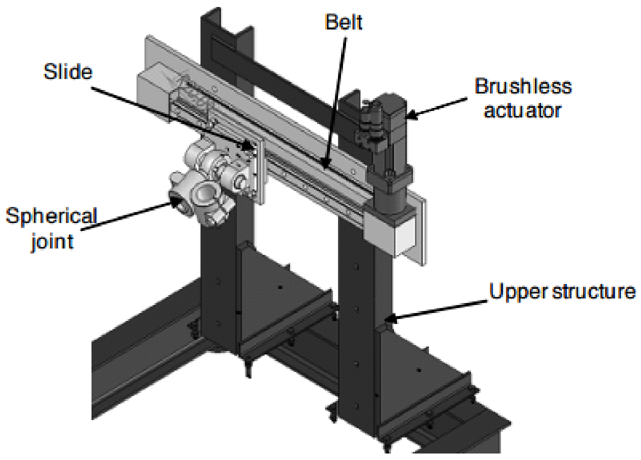


Figure 4: Yaw motion design [49].

4.2 Immersion

Immersion is the topic of how to trick the rider's mind into believing this is not a simulation and that the rider is actually riding a motorcycle. To achieve a satisfactory feeling of immersion in virtual reality and simulators one can apply a multitude of methods and aids. Visual and audible stimuli is how the rider see and hears the simulation. It can appear somewhat trivial but studies have shown that their effects should not neglected.

4.2.1 Visual vection

The visual perception is addressed as it is the key subject for VR simulation. One way to do visual vection (the illusion of self moving) is to have the foreground centralized and static while the background and peripherals undergo constant change and graphical updating [32].

4.2.2 Field of View

When choosing the rider's field of view for the simulator one can either chose a large field of view with lowered picture quality or a smaller field of view with better quality. In practically all virtual simulators the view of the user is fixed with a smaller field of view than the normal human eye [33].

4.2.3 Visual Fidelity

Impressive visual graphics is not as important as one may first believe. State of the art realistic flight simulators sacrifice visual fidelity in favor of computational speed. Having the system behave in real-time is more important than having a high visual fidelity [31]. Another aspect to consider is the frame rate, or the updating frequency of the monitor. The frame rate is typically higher in state of the art simulators than what is commercially available for VR-systems [33]. According to Robin Palmberg at KTH VIC studio a recommended frame rate for good immersion should not dip below 60 frames per second. Furthermore, simulations can be made even more realistic by taking the user body dimensions into account when rendering the virtual world [37].

4.2.4 Audible Immersion

A study using a helicopter simulator shows that the addition of sound improved the ability of pilots to fly the simulated helicopter [14]. Not only does sound increase the performance but other studies have shown improvements when using sound to increase immersion in virtual games [36] and environments [15].

4.3 Cyber-Sickness

Virtual reality sickness, also called cyber-sickness, occurs when the eyes detects movement but the body does not feel it. This is a major problem when using VR and is stated to be avoided in requirement R 12 (Cyber-Sickness). Cognitive psychologist Cyriel Diels said "It's a natural response to an unnatural environment." [46] Once the sickness starts, the user should stop immediately to avoid feeling ill for potentially several hours afterwards. The effects of Cyber-sickness can to some extent be decreased by playing VR repetitively. Since every person is unique in this respect, some people become ill more easily than others. Studies have shown that women are more likely to feel ill than men [13]. If a person is sensitive to motion sickness in reality, they are also more sensitive to VR sickness. Another problem is the more realistic something is, the more likely you are going to feel sick." This is similar to the problem called uncanny valley which in short is how humans feel about a robot depending on how close it behaves like a human (see Figure 5).

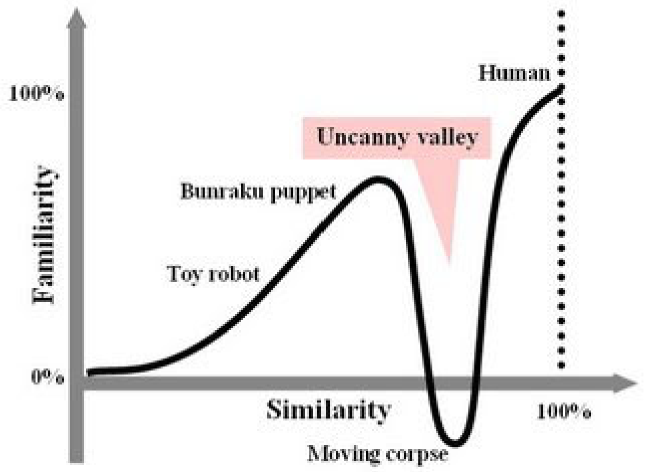


Figure 5: Uncanny valley graph [13].

Below are some important topics that influence VR sickness [29]:

- FPS, if it is too low (<60fps) the brain will notice it.

- Too low screen resolution.
- If the screen is not synchronized with the users head movement.
- If there is movement in the VR but the body is standing still.
- The more time spent using a VR headset will increase the risk of discomfort symptoms. Periodically brakes is recommended.
- High acceleration movements in VR.

4.4 The Stewart Motion Platform

Since the Motion control platform is already determined to be MicroMotion-600-6DOF-200-MK6 one must look at the limitations of this platform to better understand the limitations of this project.

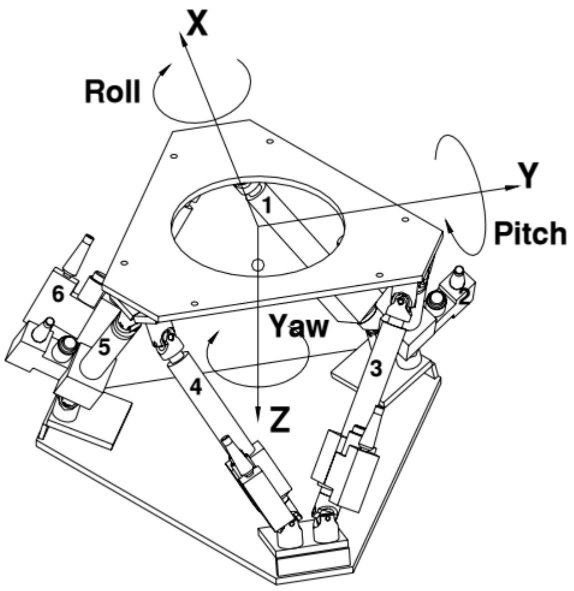
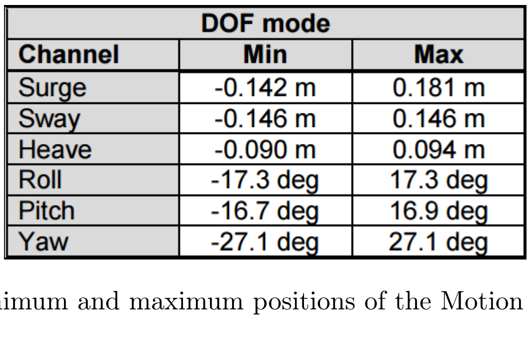


Figure 6: Coordinate systems and rotational axis, movement along the X,Y and Z axises correspond surge, sway and heave respectively [47].

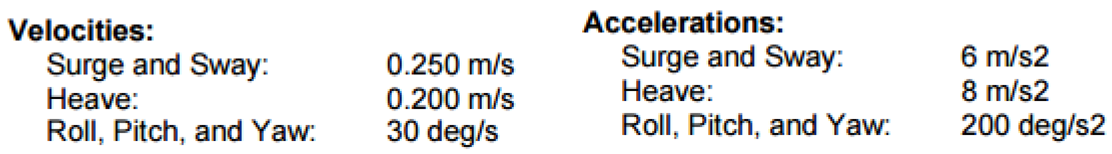
MicroMotion-600-6DOF-200-MK6 is a Stewart platform driven 6 hydraulic cylinder actuators. The actuators are controlled by the MCC which houses the motor relays and fuses. The MCC is in turn controlled by the motion computer. The motion computer controls the movement on the rig directly, receiving positions, velocities or accelerations. The rig allows for movement in 6 dimensions, 3 linear and 3 rotational as shown in Figure 6. However, the motion platform can obviously not move infinitively in any directions. The minimum and maximum values of these dimensions are listed in Figure 7.



DOF mode		
Channel	Min	Max
Surge	-0.142 m	0.181 m
Sway	-0.146 m	0.146 m
Heave	-0.090 m	0.094 m
Roll	-17.3 deg	17.3 deg
Pitch	-16.7 deg	16.9 deg
Yaw	-27.1 deg	27.1 deg

Figure 7: Minimum and maximum positions of the Motion platform [47].

While it is possible to run the rig with values which overrule the maximum values this is not recommended as it can lead to leaking, safety hazards and failure of the motion platform. In the same way that movement has its limitations, so does the velocities and accelerations, see Figure 8.



Velocities:		Accelerations:	
Surge and Sway:	0.250 m/s	Surge and Sway:	6 m/s2
Heave:	0.200 m/s	Heave:	8 m/s2
Roll, Pitch, and Yaw:	30 deg/s	Roll, Pitch, and Yaw:	200 deg/s2

Figure 8: Minimum/Maximum velocity & acceleration of the Motion platform [47].

4.4.1 Communication Protocol For The Motion PC

The motion computer runs a Linux based operating system that allows for easy interface to another computer using Ethernet UDP/IP. To establish a connection between two computers ARP is utilized. The connection requires a dedicated Ethernet connection, i.e. no other computers are allowed on the network. Upon receiving a message the motion computer will send a transmission back to the host with an update about the current information of the motion platform. The format of the transmission from the motion computer will not discussed, however is available at [47].
The communication to the motion computer requires a 127 long 32-bit array that is sent by UDP to a specific port on the motion computer. The majority of the 32-bit data elements are floats except for 3 specific elements in the array which are integers. The integer elements are related to the control of the rig. For example the second integer element is motion command. The value on the motion command will determine whether or not the motion platform will actuate the motion platform or only simulated outputs .
The message must use little-endian. The maximum frequency updating frequency of the motion computer is 250Hz.The full array must be sent each transmission, regardless of how many elements are used.

4.5 Simulation Models

There exists a number of Dynamic models or tools for dynamic models. Bikesim is one that allows for Hardware-in-loop testing with real-time applications. However, this software also comes with a hefty price tag. There also exists the motorcycle dynamics model made by R.S.Sharp and D.J.N.Limebeer [35]. The model is designed primarily for simulation of motorcycle dynamics at high speeds, >20m/s. However, it is not optimized for lower speeds. The motorcycle model uses simscape multibody and splits up the dynamics of the motorcycle into a more manageable components. The different multi-body components are

- Front wheel and tire
- Front suspension-body
- Front steer body
- Main-frame
- Rider
- Suspension-strut and rear swing-arm
- Back wheel and tire

The whole system is shown in Figure 9 below.

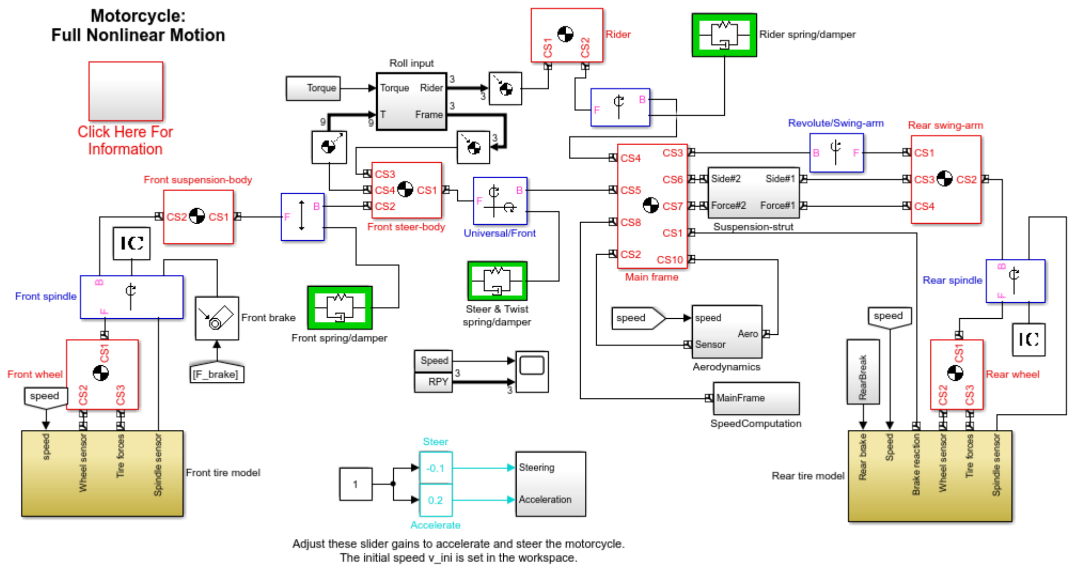


Figure 9: Full non linear motorcycle model.

As one can see in Figure 9 there are also some spring and dampeners in the system as well as a component for the aerodynamic forces and naturally the input from the rider in terms of steering, acceleration, front brake and back brake. Figure 10 shows the visualization of the multi-body system. A better explanation of this model is provided by R.S. Sharp [35] but this report will focus on two things, the tire model and the computations of positions, speed and accelerations.

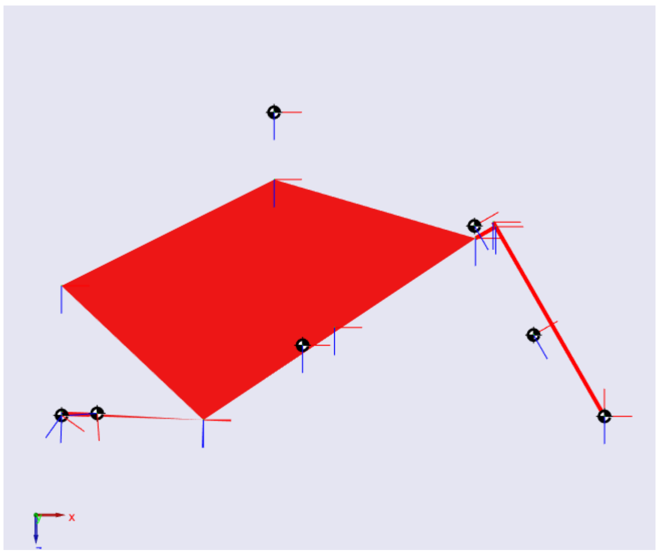


Figure 10: Multibody simulation visualization.

4.5.1 Tire Model

The model utilizes Pacejka Magic formula for calculations of the tire forces. The formula can be described in *much* more detail in another article. However, for the purpose of this project it is important to understand that it utilizes a very large number of parameters to determine the tire forces and that not all parameters can be described physically, they are seemingly magic. A contact point is calculated, which is where the tire is in contact with the ground. Forces are then calculated depending on:

- slip angle - the difference between facing of the tire and its velocity
- slip ratio - a ratio between how much the wheel is spinning as compared to its transversal speed
- camber angle - the angle of the tire with respect to the ground
- load - load on tire.

Generalized the formula is described by

$$R(k) = d \cdot sin(c \cdot \arctan(b(1-e)k + e \cdot \arctan(bk))) \tag{1}$$

Tire model easy is a different tire model. TMeasy utilizes a modeling approach based on simple dampening and spring with some lumped parameters. One of the simplifications TMeasy does is the use of generalized forces and making the forces in the different axis are dependent only on the slip coefficients in said axis.

$$F_x = \frac{F}{S} s_x^N = f s_x^N \tag{2}$$

$$F_y = \frac{F}{S} s_y^N = f_y^N \tag{3}$$

where F is the generalized forces. The generalized force can be described by

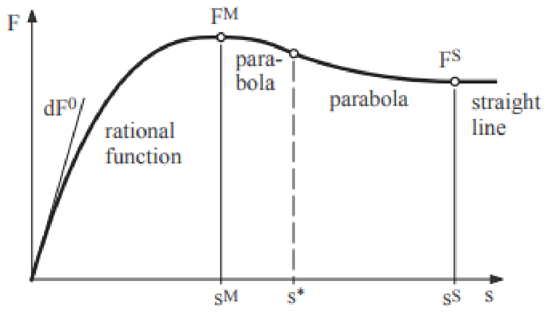


Figure 11: Generalized forces TMeasy.

where s^M, $s*$ and s^S are tire parameters related to slip and F^M,dF^0 are related to forces.
The slip coefficient is described by (in two dimensions)

$$s_y^N = -\frac{(v_x r_D + \Omega)}{r_D \Omega \hat{s}_x + v_N} \tag{4}$$

$$s_y^N = -\frac{(v_y)}{r_D \Omega \hat{s}_y + v_N} \tag{5}$$

where v_N is a imaginary velocity only there to avoid a singularity and

$$\hat{s}_x = \frac{S_x^M}{s_x^M + s_y^M} + \frac{F_x^M / dF_x^0}{F_x^M / dF_x^0 + F_y^M / dF_y^0} \tag{6}$$

$$\hat{s}_y = \frac{S_y^M}{s_x^M + s_y^M} + \frac{F_y^M / dF_y^0}{F_x^M / dF_x^0 + F_y^M / dF_y^0}. \tag{7}$$

Now the general slip can be described by

$$S = \sqrt{(S_X^N)^2 + (S_Y^N)^2}. \tag{8}$$

It is also possible to describe the displacement (or elongation) of the contact point. This relationship is described by

$$((r_D |\Omega| \hat{s}_x + v_n) d_x + \frac{F}{S}) \dot{x_e} = -\frac{F}{S}(v_x - r_D \Omega) - (r_D |\Omega| \hat{s}_x + v_n) c_x x_e \tag{9}$$

$$((r_D|\Omega|\hat{s}_y + v_n)d_y + \frac{F}{S})\dot{y_e} = -\frac{F}{S}v_y - (r_D|\Omega|\hat{s}_x + v_n)c_y y_e. \tag{10}$$

The equations can also be extended one additional dimension which is described in [34].

4.5.2 Positions, Velocities and Accelerations

The motorcycle model uses the joint between the swing-arm and mainframe to calculate the rotational positions, rotational accelerations, transversal velocity and transversal accelerations. Due to the nature of this joint some of these variables are in the absolute world, such as transversal accelerations, while others are only in respect to the local body, such as roll.
For instance Figure 12 shows the behavior of the speed and yaw.

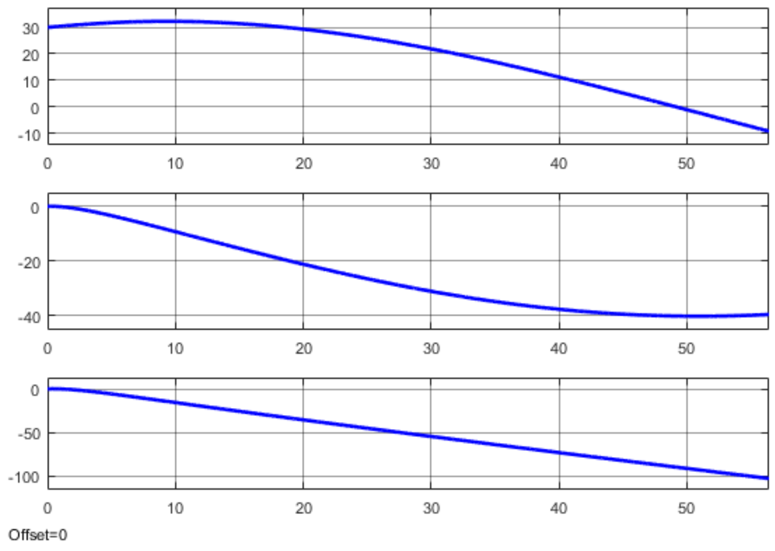


Figure 12: Velocity in x,y and yaw.

One can see that the when the bike moves with a 90 degree yaw angle(that is 90 degrees from the direction it started) the v_x goes to zero. This is naturally because the velocity are in the absolute not in relation to the bike. However, this is not the case for roll or pitch angle as these are defined as the rotations of the joint.

4.6 Motion Cueing

4.6.1 The role of washout filters in simulation

The purpose of a washout filters in motion simulation is to transform desired vehicle motion in to realizable motions for a motion platform. The washout filter is responsible for confining desired motion into a limited space while also creating the sensation of motion. This is achieved by using the desired incoming translational accelerations and angular displacements as input and manipulating them such that they fit the motion actuators limit. The washout filter is also tasked with returning the motion simulator to its neutral position so new commands can be made.

Problems arise with sustained accelerations. Motion platforms actuators are limited in their motion and therefore sustained accelerations must be simulated in other ways. One way of doing this is to tilt the motion platform so the gravity acceleration vector simulate accelerations forward and to the side. Sustained accelerations are therefore treated as low frequency signals. High frequency accelerations are simulated by using the motion simulator to move the operator in the same direction in jerks.

4.6.2 Classic Washout

The classic washout filter aims to perform the basic washout filter functions whilst not having the vestibular system built in. It can be expressed with three different channels which will be used in more advanced washout filters as well. In the following illustration the inputs used are model accelerations a_{AI} (x, y,z) and Euler angles β_A (roll, pitch, yaw).The outputs a_{SI} and β_S are motion platform accelerations and rotations.

- The translational channel for high frequency translational accelerations
- The tilt coordination channel for low frequency accelerations
- The rotational channel for high frequency rotations.

Figure 13 shows how the different channels are combined.

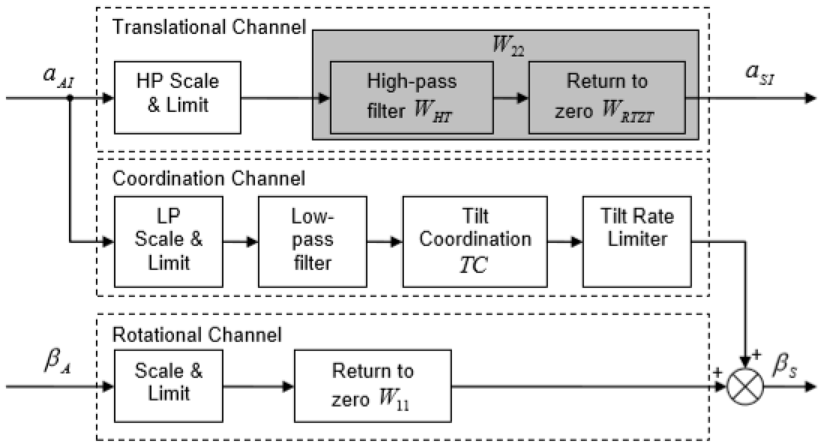


Figure 13: Base for washout filters (classical washout filter) [26]

4.6.2.1 Limiters and Scalars

Before the signals from the motion model can be applied to the washout filter they will first need to be limited and scaled. Hard limiters are used to keep the motion simulators movements in bounds and prevent too violent motions. When a limit is reached the signal is saturated to a constant value. The scalers are used to decrease or increase the magnitude of the motion controller's movement at all frequencies.

4.6.2.2 Translational Channel Load

The translational accelerations pass through the scalars and the translational channel. The high frequency accelerations pass through while the lower frequencies signals are discarded. The minimum order of the high pass filter can be derived using the descriptions of filters in general aircraft simulation [30]. Consider a constant acceleration A_t.

$$\ddot{X} = A_t \tag{11}$$

Now consider a translational acceleration through the washout filter to the motion simulator in the same direction called X_s:

$$\ddot{X}_s = \ddot{X} A_t s^n \frac{N(s)}{D(s)} \tag{12}$$

Where N(s) and D(s) represent the nominator and denominator the filter and are:

$$N(s) = \sum_{k=0}^{K} a_k s^k \tag{13}$$

$$D(s) = \sum_{i=0}^{I} b_i s^i \tag{14}$$

and

$$J > K + n \tag{15}$$

$$a_0 \neq 0, b_0 \neq 0 \tag{16}$$

Laplace transforming equation 11

$$\hat{X} = A_t / s \tag{17}$$

The displacement of the motion simulator can be expressed as

$$X_s = \ddot{X}_s / s^2 \tag{18}$$

By combining equation 11 and equation 20 X_s is:

$$X_s = A_t s^{(n-3)} \frac{N(s)}{D(s)} \tag{19}$$

Using the final value theorem it then follows that

$$\lim_{t\to\infty} = \lim_{s\to 0} A_t s^{(n-2)} \frac{N(s)}{D(s)} \tag{20}$$

As the washout filter is needed to return to zero after some time has passed the order n is to be

$$n > 3 \tag{21}$$

4.6.2.3 Tilt Coordination

The constant accelerations are discarded in the Translational channel but they still need to be felt. The tilt coordination channel translates accelerations from a lower frequency to Euler angles by first limiting and scaling the translational accelerations and then converting them to Euler angles. A second order low-pass filter is used to only keep accelerations of a lower frequency.

$$W_{TCC} = \frac{\omega_c^2}{(s^2 + 2\psi\omega_c s + \omega_c^2)} \tag{22}$$

The translational accelerations are then converted to Euler angles by tilting the user and using the gravity vector to emulate translational accelerations. To simulate a constant acceleration in the x direction the motorcycle is tilted with an angle $\theta_t c$ (pitch increase) around the y-axis. Decomposing the gravity vector in the x and z direction we arrive at

$$g = -A_x = \begin{bmatrix} -g\sin(\theta) \\ 0 \\ g\cos(\theta) \end{bmatrix} \tag{23}$$

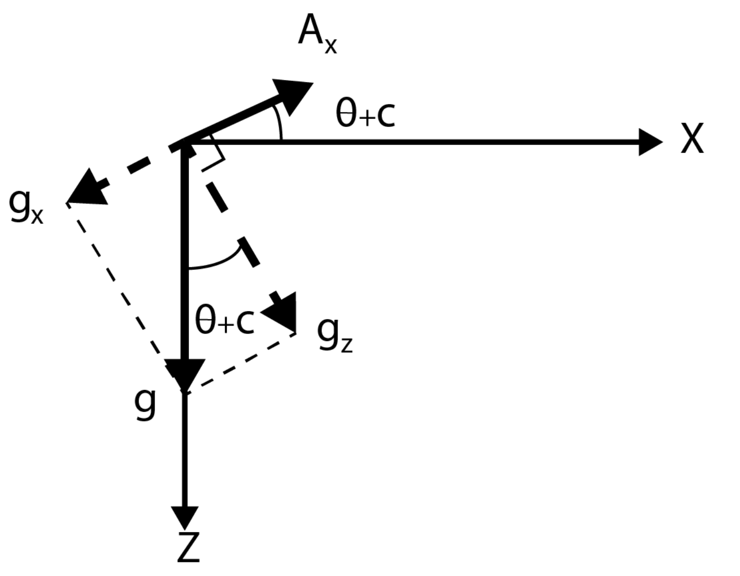


Figure 14: Induced acceleration A_x and the gravity vector.

Similarly, to simulate a constant acceleration in the y axis the motor cycle is tilted with an angle ϕ_{tc} (roll increase) around the x-axis. Decomposing the gravity vector in the y and z axis with the angle ϕ_{tc}:

$$g = -A_y = \begin{bmatrix} 0 \\ -\sin(\phi_{tc}) \\ g\cos(\phi_{tc}) \end{bmatrix} \tag{24}$$

Combining A_x and A_y for a complete translational vector A_{trans} yields:

$$g = A_{trans} = \begin{bmatrix} g\sin(\theta_{tc}) \\ g\sin(\phi_{tc})\cos(\theta_{tc}) \\ -g\cos(\phi_{tc})\cos(\theta_{tc}) \end{bmatrix} \tag{25}$$

The maximum rotation in the motion simulator is under 20°. The trigonometric substitution of $cos(\alpha) \approx 1$ and $\sin(\alpha) \approx \alpha$ can then be made. The simplified acceleration vector is then:

$$g = A_{trans} = \begin{bmatrix} g\theta_{tc} \\ g\phi_{tc} \\ -g \end{bmatrix} \tag{26}$$

By removing the influence of gravity the Euler angles for pitch and roll can be extracted with the following transformation matrix.

$$TC_{transform} = \begin{bmatrix} 0 & 1/g & 0 \\ 1/g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{27}$$

The translational accelerations for low frequencies can then be expressed as Euler angles β_s as:

$$\beta_s = TC_{transform}A_{trans} \tag{28}$$

4.6.2.4 Rotational Channel

Repeating the steps started at 11 to determine the minimum order of the high pass filter for rotations n leads to:

$$n > 2 \tag{29}$$

If the input is an angular rate. If the angular position is used the order is reduced another time leading to the rotational channel being stated as a first order high-pass filter.

$$W_{RC} = \frac{s}{s + \omega_r} \tag{30}$$

4.6.3 Optimal Washout

The optimal washout filter differentiates itself from the classic washout filter by including a control loop. By using the vestibular system to produce a reference signal, translational accelerations and angular rates y_a, on the sensed movement and comparing it to the sensed signal after the washout filters output an error signal is found. The perceived motions from the vestibular system are then made more realistic by minimizing the error signal before implementing the washout filter and therefor reduces motion sickness [6].

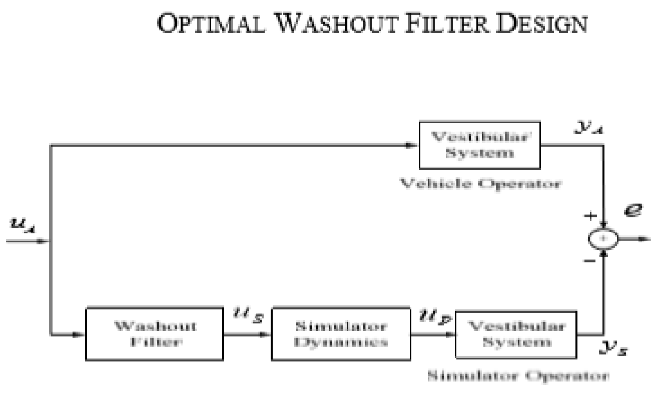


Figure 15: Optimal washout filter design.

As illustrated by [38] using u_a as input accelerations from the model, y_a as the sensed signal and y_s as output. The error signal e is used for minimizing the control error

4.6.4 Adaptive Washout

The adaptive washout filter shares the same algorithms as the classical washout filter but with a great advantage. Variables that are constant in the optimal washout filter are instead varied in real time to minimize a user defined cost function based on the error signal described in the optimal washout filter [9]. Potential issues with the adaptive approach is the increased execution time for the minimization calculation and the difficult tuning. The strengths in adaptive washout filters lies in producing a more realistic motion as it rejects to residual motion displacements, “washes out” motions faster and applies smaller motions for longer periods of time [10].

4.6.5 The Vestibular System

The vestibular system is more respondent to forces rather than acceleration. However, the sensed for has the same units as the acceleration as it’s the force on the body per unit mass. The sensed force F_{sense} can therefore be calculated by using the current accelerations in m/s^2 on the head.

$$F_{sense} = a - g \tag{31}$$

Each inner ear has a vestibular system and so the heads position over time need to be tracked for the washout filter to have a reference signal.

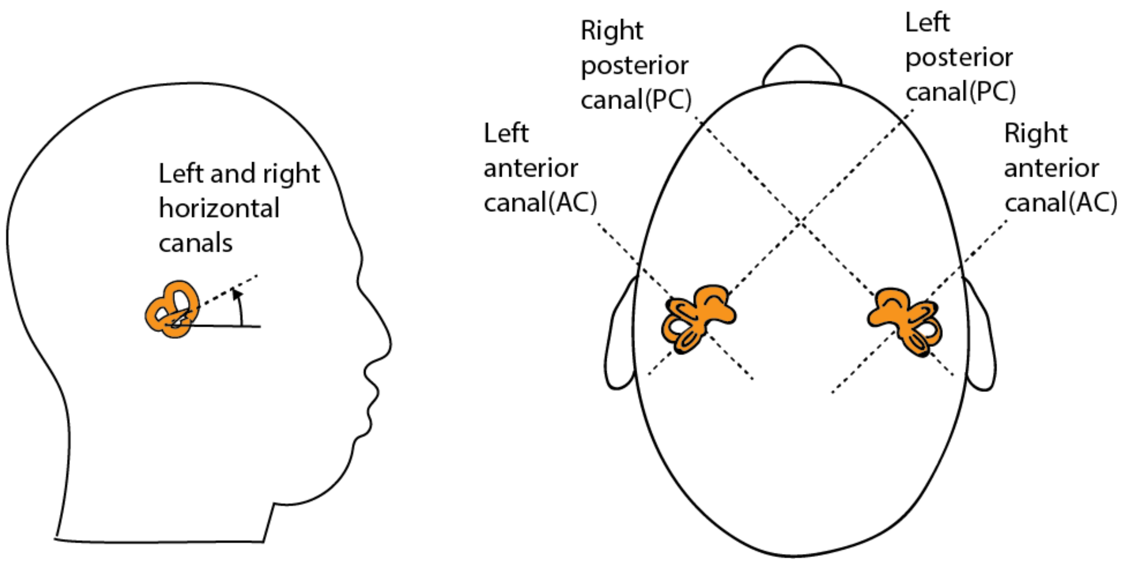


Figure 16: Location of the vestibular system.

4.6.5.1 Otoliths

The otoliths sense linear movements and head tilts in relation to the gravity vector. There are two otoliths in each inner ear, one for sensing movements in the horizontal plane (Ultriccle) and one for the sensing movements in the vertical plane (Saccule). When moving the otholic membrane is moved as well causing a displacement which is measured.

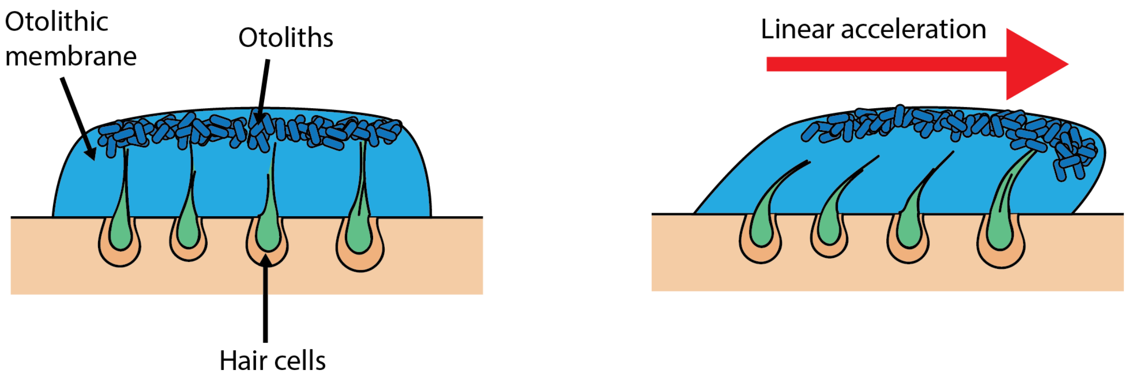


Figure 17: The Otolith and the effect of linear acceleration.

The Otolith sense the gravitational force linearly which allows for tilting the driver to simulate translational accelerations. Using equation 31 a transfer function for sensing linear accelerations by comparing the otolith to a mass-spring-shock absorber [26] is then made. In the transfer function a dead zone for small accelerations has been omitted to not include a discontinuity. This has no effect on the user unless the signals from the washout filter are significantly scaled.

$$H_{oto} = K_{oto}\frac{(\tau_{\alpha}s+1)}{(\tau_l s+1)(\tau_s s+1)} = \frac{\hat{f}}{f}$$

Where k_{oto}, τ_s, τ_l are constants and $\hat{f}$ is sensed force. This representation is more recent, research has led to a more responsive model for simulation in sustained accelerations. The constants in H_{oto} can be found in [28].

4.6.5.2 Semicircular Canals

Rotational movements are sensed by three semicircular canals. Each canal represents a different plane and are filled with a liquid called endolymph. The endolymph is divided by a seal called Capula which is able to move while preventing the endolymph from passing. Rotating in the same plane as a canal causes a force due to the inertia of the endolymph to move the Capula in the opposite direction of the original rotation.

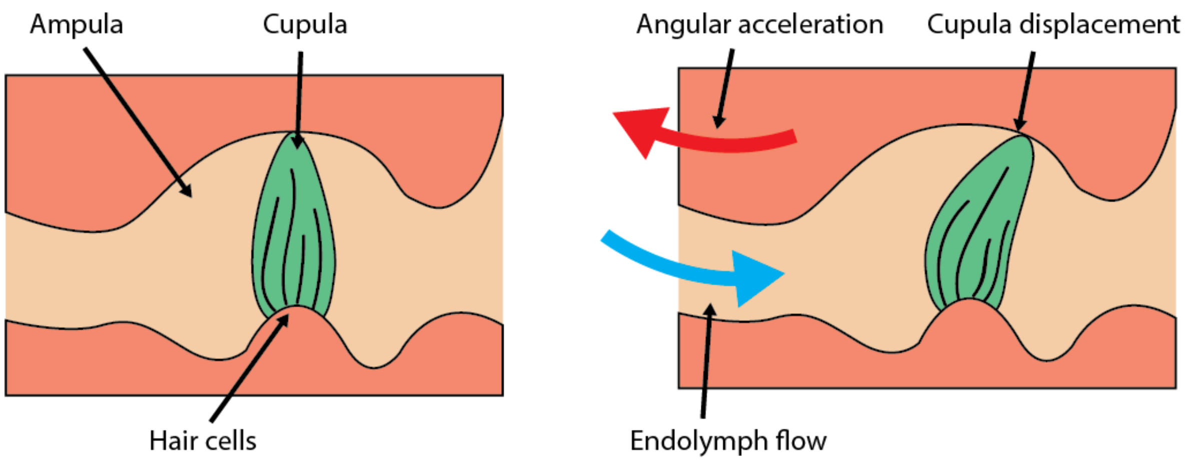


Figure 18: The cupula and displacement of it due to angular acceleration.

By treating this as an over-damped torsion pendulum the following transfer function is found [26]. Just as with the otoliths a dead zone for small displacements have been omitted.

$$H_{semicanal} = \frac{\hat{\omega}(s)}{\omega(s)} = K_{scc}\frac{T_L T_a s^2}{(T_s s+1)(T_L s+1)(T_a s+1)} \tag{32}$$

Where K_{scc}, T_L,T_a, T_S are constants different for each plane that can be found constants in H_{oto} can be found in [28].

4.6.6 Torque In Handlebar

To simulate correct forces in the handlebar, an understanding of where and when the forces appears in the handlebar is needed. In "Investigation of Motorcycle Steering Torque Components" the torque and it main components is analyzed in depth. In Figure 19 we can clearly see the torque in the handlebar while making a constant right turn in different speeds and with different degrees of curvature, i.e. lateral acceleration with respect to the defined coordinate system in Figure 6.

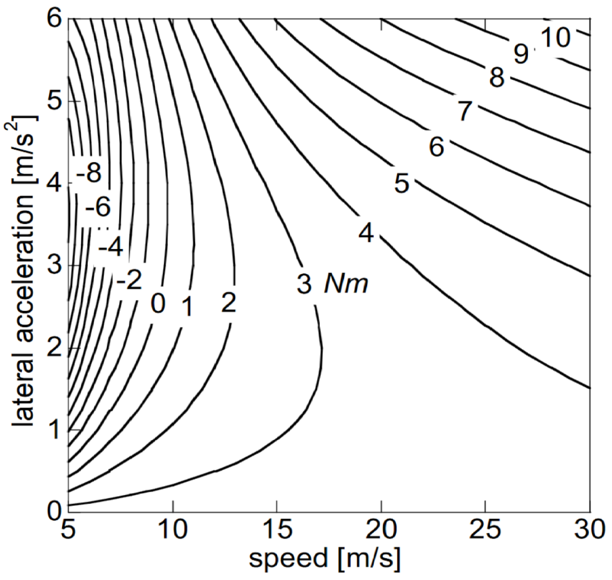


Figure 19: Rider steering torque.

4.7 Handlebar Torque Feedback

The handlebar is the direct connection between the user and the dynamic model. The rider must feel an almost immediate reaction when throttling (as one would expect while riding a MC). Meanwhile, the dynamic model must also provide feedback to the user through torque in the handlebar. In the journal [3] an equilibrium of the front frame is set up with the riders torque input as one of the forces. The two largest torques when in a turn comes from the normal load on the wheel and the lateral force at the contact point between the wheel and the ground. These torques are however counteracting each others leading to smaller forces having an effect on the steering torque, such as inertial and centrifugal force. In the journal the effects of accelerating and breaking in a curve and the conclusion was that breaking had larger effects and generated large torque in the handlebar while accelerating did not generate as large torque [3]. This is because when breaking there is a force transfer towards the front of the motorcycle which increases the torque in the front frame which the rider must handle.

4.7.1 Incremental Magnetic Rotary Encoder

An incremental magnetic rotary encoder generates pulses on two different output lines when the encoder shaft is rotating. It does this by having magnets poles placed on the shaft which are the detected by magnetic sensors. Many of these encoders uses a quadrature design which means that two sensors are spaced out so that there is a 90-degree phase shift between the pulses that are generated.

This phase shift makes it possible to detect in what direction the encoder is rotating in, this is done by looking at what order the pulses come in. To calculate rotational speed the time between two pulses is measured and by knowing how pulses are generated during one full rotation it is possible to calculate the current rotational speed. One of the main drawbacks with using an incremental encoder instead of an absolute encoder is that there is no way to detect at what absolute position the encoder is in, since it only generates outputs while rotating. On the other hand, it is still possible to know its relative position to starting point and it is very useful while trying to measure rotational velocity. This has made incremental encoders very popular to use for motor control.

4.7.2 DC-motor, Gearbox and Motor Driver

When merely mounted on the Stewart platform the handlebar on the motorcycle is easy to turn which leads to unrealistic simulated driving, especially at higher speeds. Therefore, counter forces on the steering rod must be added when the handlebar is being moved. Producing forces on the

handlebar can be done in many ways [11]. Using an electric direct current (DC) motor is one common solution since it is easy to use and comes with a good price compared to brushless dc (BLDC) motors or alternative current (AC) motors.For this project, DC motors are sufficient. The handlebar torque can reach high values as discussed in section 4.2.1. Therefore, a transmission is needed to reduce the size and cost of the electric motor. Chain with sprockets, belts with pulleys or gears are common solutions. The downside with them are friction and clearance between tooth's causing unwanted delay. Clearance in the handlebar could potentially be detected from the driver. A cable transmission can be used which comes with two main advantages, little friction and no backlash. A cable transmission consists of a wire, a capstan mounted on the motor and a drum rotating the steering rod of the motorcycle. A plastic wire such as a fishing-line might seem to be a better alternative than a steel wire since it is more flexible. A fishing-line stretches over time however, and never stops doing so. A steel wire also stretches during load but stops doing so over time. Some companies sell pre-stretched steel wires to remove this problem but this comes at a high price. An alternative solution is to fix the metal wire at one end and allow for tightening on the other.
One downside with a cable transmission is that the wire might slip during load. This threshold can be calculated using a formula called the capstan equation

$$T_{load} = T_{hold} \cdot e^{\mu}\theta \ , \tag{33}$$

where T_{load} is the applied tension on the line, $\mathrm{T_{hold}}$ is the resulting force, mu is the coefficient of friction between the rope and the capstan and theta is the total wrap-around angle measured in radians. This means that more wraps around the capstan increases the carrying load without slipping.

A motor driver is needed to be able to control the DC-motor. One robust alternative is the Sabertooth dual 32A motor driver created by the company Dimension Engineering 20. It contains a lot of features, it is well documented and can easily be interfaced with Arduino using the Sabertooth libraries. Different operating modes can be set using switches on the motor driver and its software which can be accessed via a USB port and a program called DEScribe. It can send live operating data such as drawn current, transistor temperatures, battery voltage and more. It can run continuously on 32 ampere and peak at 64 amperes. Nominal voltage is between 6 volts and 30 volts. The pulse width modulating (PWM), also known as the switching frequency runs on 30kHz. A 8 bit integer between -127 and 127 is used as input to the Sabertooth motor driver to set the motor speed. The motor driver converts this value into a PWM signal using four transistors. A value of 0 means standstill which is converted to a 50% PWM signal, -127 is converted to 0% PWM giving full speed reverse and vice versa. The PWM signal controls the current through the transistors which are aligned in a H shape enabling the current to go in either direction through the motor. This is the reason why some motor drivers are often called a H-bridge. A 50% PWM-signal means that the current travels forward and backwards the same amount of time. Since the switching rate is very fast the motor only notice the mean value which means standstill in this case.

Figure 20: Sabertooth 2X32 dual motor driver.

4.7.3 DC-Motor Torque Control

The torque generated by a DC-motor is linearly proportional to the current going through the motor at any time. This makes DC-motors a good candidate when you want to control torque. There are several ways to design a torque control system, the easiest in terms of control is to use a current sensor to measure the current and then converting the current into torque by using the motors torque constant. This approach has the benefit of having the feedback being very closely related to what is being controlled. The problem is that you need a current sensor that can measure both positive and negative current accurately and without any major noise.

Another approach is using the angular velocity of the motor shaft and then use the DC-motor equations to calculate the current. The voltage is also needed to calculate the current but since the voltage often is the controller output when controlling a DC-motor some approximations are needed to utilize this approach, such as using the last output signal to calculate the new one. The second approach is highly dependent on how accurate the motor constants are and adds a level of calculations which can affect the result.

4.8 Virtual Reality and Simulator Game

Unity3D is a very easy and intuitive program to start developing a game from scratch. As our team do not have any knowledge of 3D-game development, this is an obvious choice for the VR development. Currently there exist few motion simulators with Virtual Reality where all aspects (visual, realistic motions, sound, haptic feedback etc.) are fully consistent. A big problem with these systems is that the accelerating motion generated by the platform will impact the VR-headset's accelerometer and cause the visual point-of-view to change. However, a Belgian company named *Vection VR* has developed a VR driving school simulator [52] with an Oculus Rift Headset, similar to the concept of this project but with a car instead of a motorcycle. To tackle the accelerometer problem, they used their own written public software patch for the headset to add the possibility to dynamically offset the VR orientation to the user seat instead of the ground. It measures the motion of the simulator motion platform and subtracts that from the graphical view. Combined with a sensor it reduces simulator sickness and improves the sense of presence and feeling of motion. This patch or method could be used to reduce VR-sickness.

To send and receive data with Unity, TCP or UDP as communication protocol is mostly used for this type of projects and example code are available from Microsoft [24]. The rider will use a head-mounted-display for VR to see the 3D-environment as seen in Figure 54. VIC, the Visual Studio at KTH's Computer Science Department provided a list of appropriate hardware that will be utilized. Using this, the rider will be able to move around his head and see the world around him as well as the rest of the motorcycle for increased immersion.

4.9 Middleware

Middleware is a type of software that works as a communication infrastructure, handling the layer between the network operating system and the application layer, translating information between the system nodes. The main purpose of a middleware in this project is to construct a communication infrastructure that can be easily observed and modulated. Furthermore, a middleware increases the stability of a system in terms of predictability, latency efficiency, scalability and dependability [16]. To develop a Real-time system including multiple nodes using different software and programming languages, a middleware can be very beneficial as a gateway between the nodes. Each underlying software can translate the messages to a standard that is used by the middleware, so that every node have access to the information. Furthermore, using a standard for the communication facilitates the process of expanding the system with more nodes and functionalities. Figure 21 below illustrates the middleware layer between the system nodes.

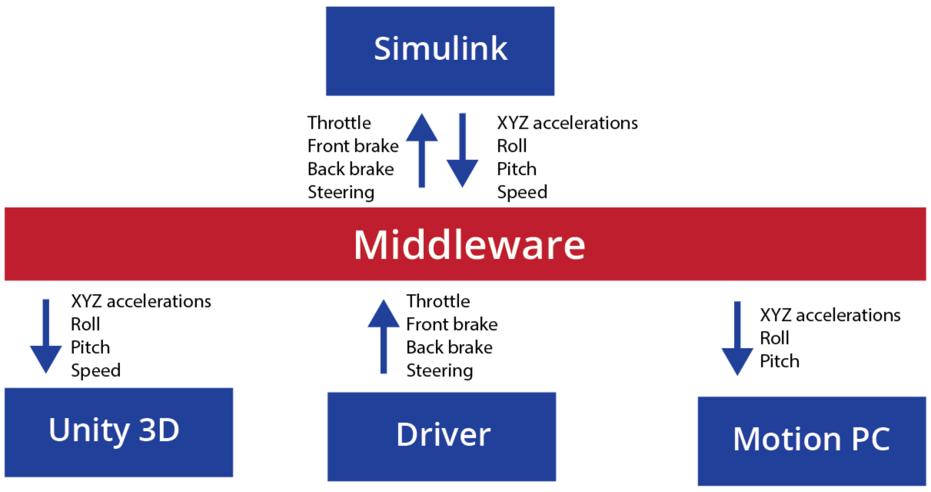


Figure 21: Illustration of system nodes and variables.

4.9.1 Publish-Subscribe Messaging Pattern

There are many different middleware technologies, each with both advantages and disadvantages. Publish-Subscribe is a messaging pattern often used in robotics systems, ROS (Robot Operating System) is such a system, a loosely coupled environment where one node can publish and/or subscribe with no direct communication to the other nodes [18]. Simplifying plug and play and many-to-many, meaning subscribing to many different publishers at the same time. In this project a publish/subscribe pattern will be used.

Robot Operating System

Robot Operating System (ROS), is an open-source middleware used in robotics development. Widely used in many cyber-physical applications, ROS has a large community with multiple libraries which makes it easy to implement. However, ROS is not a real-time system, in terms of deadlines, reliability and durability. ROS uses a master to collect all the nodes in the system, the master then initially advertises all the subscribers and publishers for a peer-to-peer connection illustrated in Figure 22.

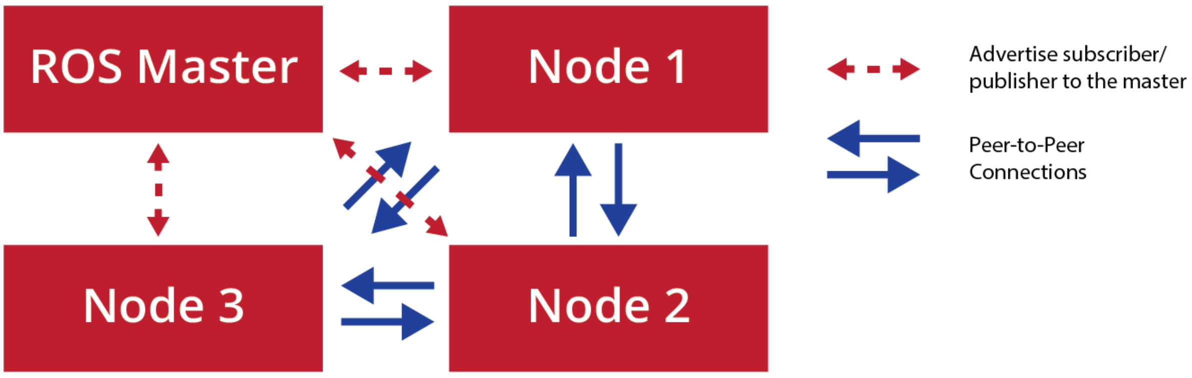


Figure 22: ROS Illustration.

Data Distributed Service

Data Distributed Service (DDS) also uses a publish/subscribe pattern just like ROS. The significant difference is that DDS is more suitable for real-time embedded systems in terms of the Quality of Service (QoS) it provides (deadlines, reliability and durability). DDS uses a Global Data Space (GDS) instead of a ROS master. The Global Data Space is required for DDS for subscribers and publishers to join and leave the GDS and be dynamically discovered. The GDS includes QoS policies as what should happen if something goes wrong. Figure 23 below illustrates a GDS in a DDS system [17].

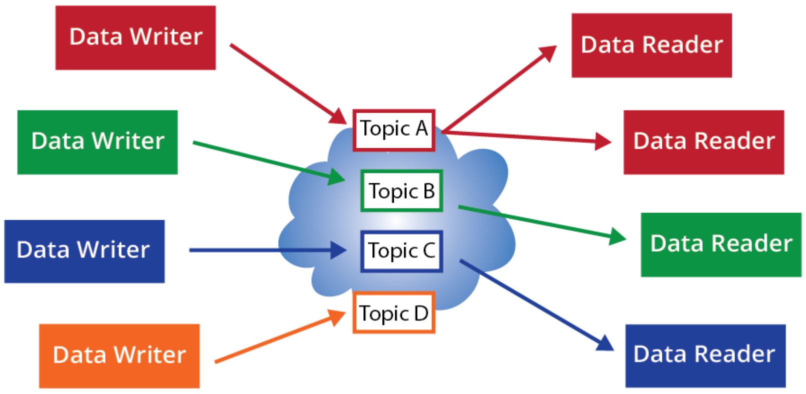


Figure 23: DDS Global Data Space.

In this project, ROS will be used due to its large community and library. DDS is a service that requires a third party application, and the more sophisticated programs are not open source such as VR-link, RTI Connext and Simware. These programs comes with many plug-ins and easy analytical functions, although this project will not need anything outside the scope of ROS. ROS2 is under development including DDS within the communication infrastructure. This will make it easier to implement the QoS within the system architecture making it a real-time system.

5 METHODS & IMPLEMENTATION

The section is dedicated to how different aspects of the project were implemented and which methods were used to do so.

5.1 Motorcycle Construction

To be able to reach full immersion the driver needed to sit on a realistic representation of a motorcycle with a seat and frame and such. An effective way to achieve this was by taking a real motorcycle and remove the drive-train and other unnecessary parts to reduce the weight. When fixating the motorcycle frame to the rig it was mounted at a distance from the plate of the rig. The distance was equal to the distance that the frame is from the ground when it had wheels. This was because the motorcycles natural rotational axises were then aligned with the rigs rotational axises. This enabled the rig to use its full potential without putting unnecessary strain on it.

5.1.1 Motorcycle Attachment to The Stewart Rig

A wood-construction between the Stewart rig and the motorcycle was firstly used to quickly start tests with the rig. This however, gave some noticeable damping between the rig and the motorcycle. This creates uncertain behavior and is not an optimal solution considering safety aspects. To meet requirement R 1 (Mechanical Structure), a steel construction had to be designed and constructed. 5 mm thick steel plates were water cut and welded together. The box was attached to the Stewart rig using M8 screws and bolts to made it possible to easily remove it if needed. Attachment points for the motor was used on the motorcycle to create a stiff connection between the frame and the steel box. The frame and the box was painted black to not distract the eye. Holes on the top and the bottom of the box made it easy to draw cables between the motorcycle and the micro-controllers. Figure 24 shows the final solution.

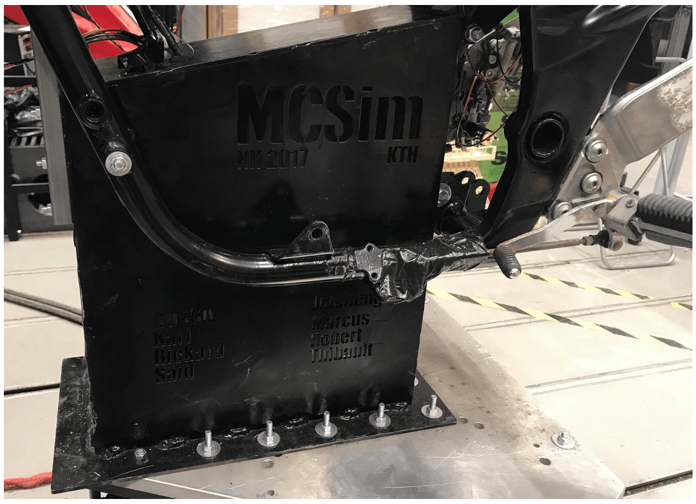


Figure 24: Final design of the steel box between the rig and the motorcycle.

5.1.2 The Motion Simulator

For this project a hydraulic Stewart rig , MicroMotion-600-6DOF-200-MK6, was used. The constraints of the motion simulator lies in the limited actuators and the relatively low stiffness. If the the motion simulator was to receive a signal out of bounds it will soften the motion after an adjustable threshold and stop at the maximum given value. These thresholds were kept according to recommendations in the documentation for the simulation platform but the inherit motion cueing filters were removed as they were not satisfactory.

The dynamic model and motion simulator share both rotate around the ground but differ in the Cartesian plane for translational accelerations. The translational accelerations from the dynamic model are extracted from the center of gravity of the motor cycle.The motorcycle's center of gravity is above the motion simulator platform and therefore the frame for motion simulator is adjusted with the following vector:

$$R_{MP} = \begin{bmatrix} 0 \\ 0 \\ 0.7 \end{bmatrix} \tag{34}$$

5.2 Communication With Motion PC

The physical connection is made by an Ethernet cable between the host and the motion computer's Real time Ethernet jack(RETH0), creating a 2 computer network. A S-function-builder is used to setup the UDP communications on the host. A 127-float array is created at the initialization of the S-function builder. Using the standard winsock2 protocol a socket is created which sends the array to the specific IP and port on the motion computer. To convert the integers into float a union of an int32 and a float is utilized. To interface the simulink model remains a trivial task of creating an input to which a signal in simulink can be dragged. All elements not utilized are set to zero before transmission.

In order to use the different winsock2 functionalities the win 32 library has to be linked to the S-function. This is done using the pragma macro, which is acceptable since the Microsoft Windows SDK v7.1 compiler was used (which allows for the use of macro in compiling)

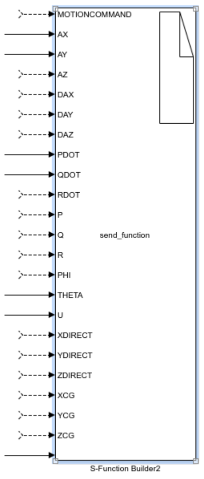
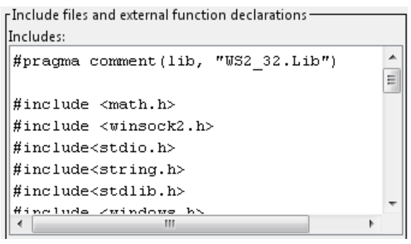


Figure 25: Simulink S-function builder interface.


Include files and external function declarations
Includes:

```
#pragma comment(lib, "WS2_32.Lib")

#include <math.h>
#include <winsock2.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include <windows.h>
```

Figure 26: Macro for linking to WS32.lib.

5.3 Simulation Model

Figure 27 shows the final version of the motorcycle model in Simulink. To fully fit the model into this project a few augmentations are made from the base version previously described:

- Added a stabilizing force to prevent motorcycle roll crashes, especially at lower speeds.

- Added framework for TMeasy.
- Added a regulator to keep speed above slow speeds.
- Made accelerations relative to motorcycle body.
- Added power train model.

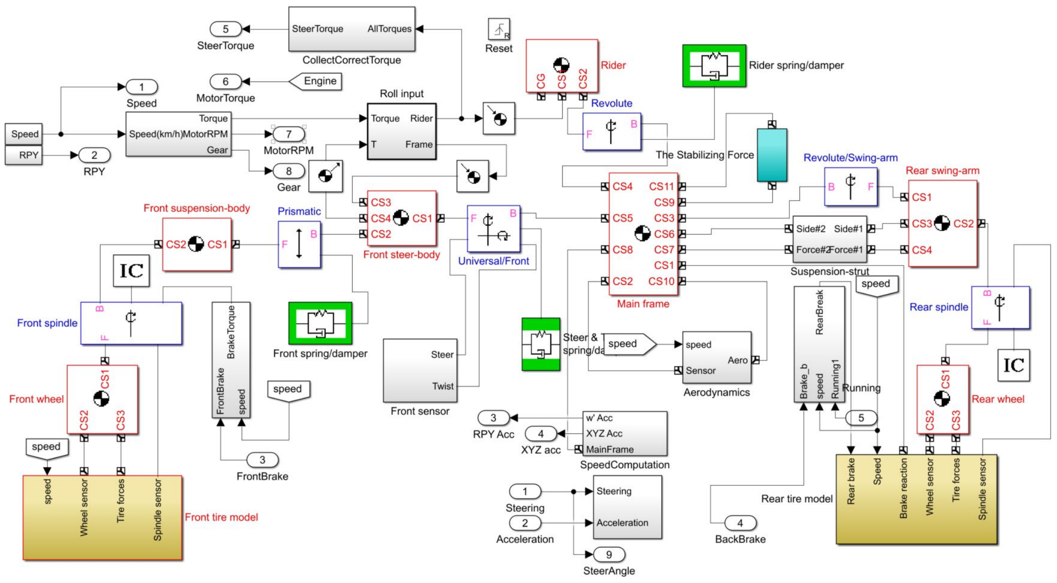


Figure 27: Motorcycle model in Simulink.

5.3.1 Preventing Roll Crashes

During validation, the model turned out to be unstable in the roll angle when taking a turn. The motorcycle model rolled too much, even at low steering angles. Figure 28 shows the roll angle of the motorcycle when driving in 60 km/h with a 3 degree turn on the handlebar and no acceleration. The roll angle continuous to grow during the large turn and when the motorcycle is tilted 70 degrees the tires presumably loses grip causing the model to crash after only a few seconds (40 meters travel length). This is not realistic and an unwanted behavior that must be prevented to meet requirement R 13 (System Stability).

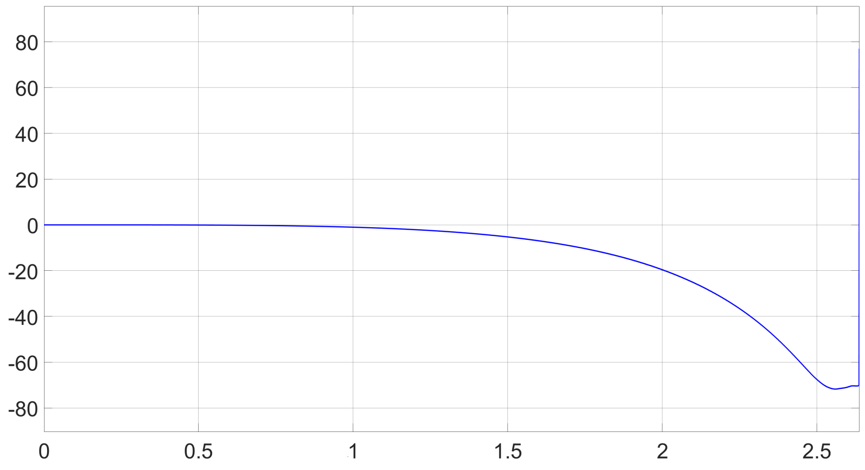


Figure 28: Roll crash, angle in degrees as a function of time.

To solve this problem a counteracting force was applied perpendicular on the main frame when the roll angle passes a certain threshold which is illustrated in Figure 29. This threshold is set to 17.3 degrees since this is the maximum roll angle on the Stewart rig (see Figure 3). A switch triggers the counteracting force if the roll angle value is more than ±17.3 degrees. Two PID blocks and a gain block were tuned and used to give a force sufficient to stop the motorcycle from surpassing the allowed roll angle while not destabilizing the motorcycle due to overshoot.

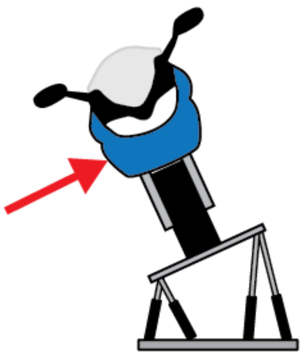


Figure 29: Roll crash solution with added force.

The output from Roll angle (PID regulators) was then fed directly into the mainframe. Th body actuator is a local body requiring no additional conversion before its application on the mainframe.

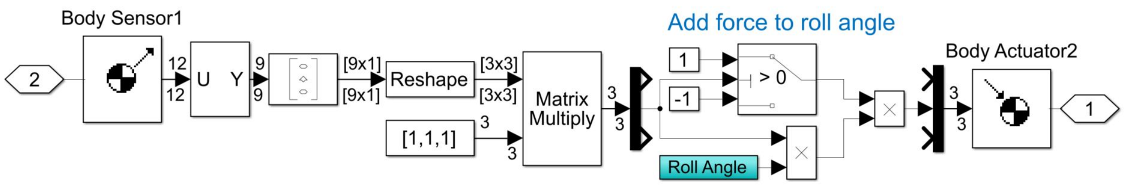


Figure 30: Roll crash solution, cyan blue sub-block in Figure 27 called "The Stabilizing Force".

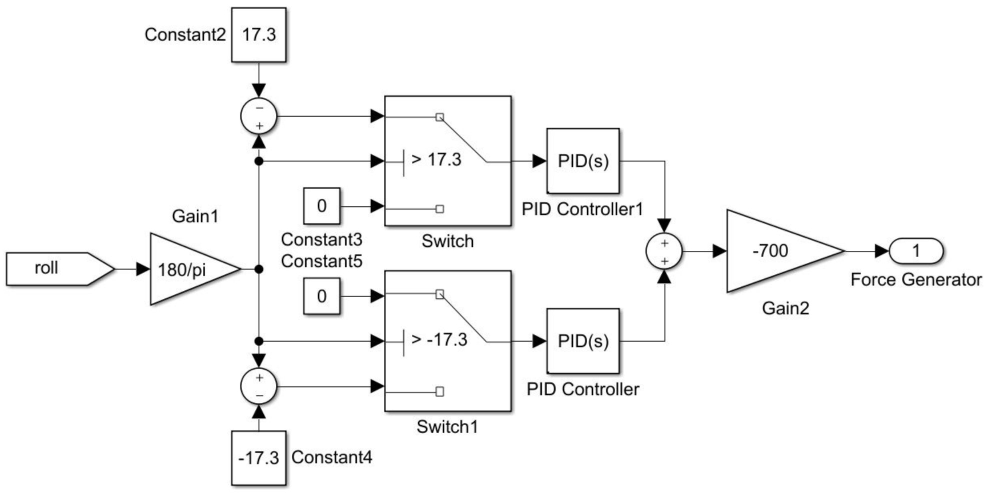


Figure 31: Roll crash solution, cyan blue sub-block in Figure 30 called "Roll Angle".

The PID-solution is seen in Figure 31 which is a sub-block called "Roll Angle" from Figure 30. The result with tuned PID regulators are displayed in Figure 32. (The same settings as for Figure 28 are used). As seen in the figure, the solution shows minor oscillations but remains stable over time and clearly prevents the motorcycle from having a larger roll angle than 17.3 degrees. This ensures the model stays stable in curves when tilted. The force however, only acts on the main frame and there is still an issue with the front wheel having unstable wobbling

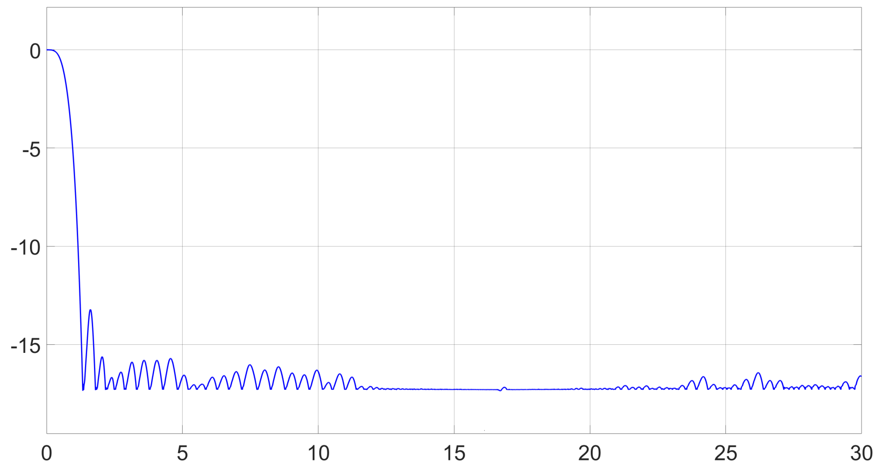


Figure 32: Roll angle as a function of time.

5.3.2 Framework for Tire Models

As described previously in the report the model utilizes Pacejkas magic formula. However, due to an issue with unstable pitch acceleration at low speeds an attempt to solve this issue with Tmeasy is made.
Using TMeasy it is possible to describe the forces applied on the tires while only using transversal speed in x and y direction as well as the rotational speed of the wheel, Ω, as inputs. The task then becomes to determine the generalized forces curve (utilizing the parameters S^M, $S*$, S^S F^M and dF^0) and perhaps the equations for the displacement of the contact point described previously. To verify the behavior of the tire model one can use the Pacejkas magic formula, run at higher speeds, and compare the two models. While this is not perfect it would give a good starting indication of the tire model.
The model is designed fully in Simulink and utilizes the same inputs as Pacejkas magic formula on the front tire while the rear tire has the same inputs except for engine torque which is applied to the rear spindle. The load on the tire was during testing taken from the magic formula as load is considered constant, however for TMeasy have the following inputs and outputs:

- **Inputs**
  - Wheel sensor
  - Spindle sensor

- **Outputs**
  - F_x
  - F_y

.

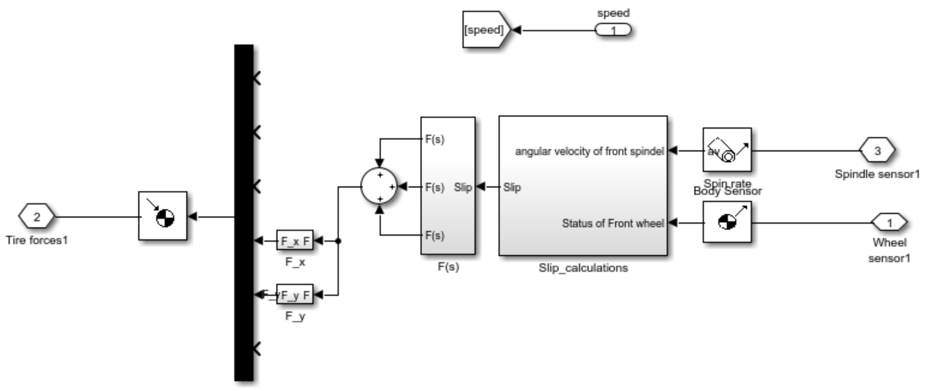


Figure 33: Front tire model.

First the slip calculations are performed.

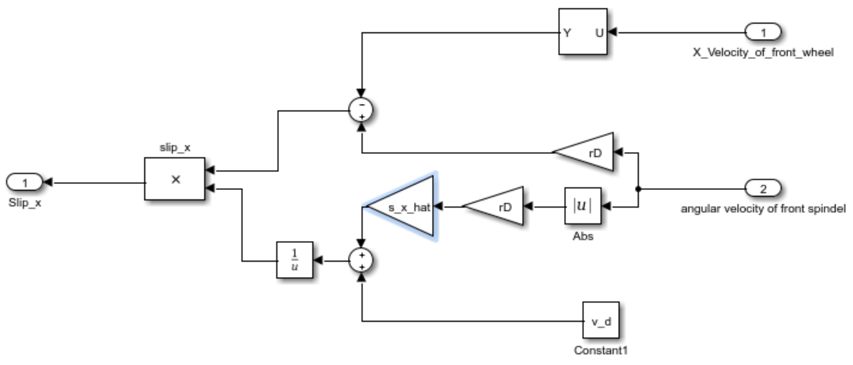
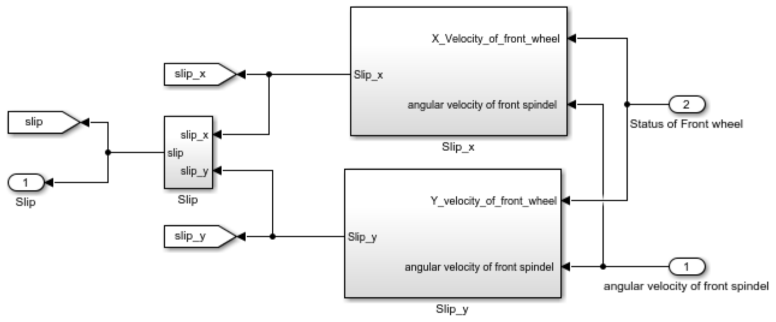


Figure 34: Slip calculations front tire x direction.

The two slip calculations are combined to the generalized slip. Due to issues with getting the model to work the displacement of the contact point is not currently used. If this was to be implemented the radius of the tire would obviously change and therefor the radius of the tire could not be considered a constant, IE the constant gain r_D needs to be changed.



slip_x
slip
slip_x
slip
slip_y
Slip
1
Slip
slip_y
Slip_x
X_Velocity_of_front_wheel
angular velocity of front spindel
Slip_x
Slip_y
Y_velocity_of_front_wheel
angular velocity of front spindel
Slip_y
2
Status of Front wheel
1
angular velocity of front spindel

Figure 35: Slip calculations front tire.

Next step is to determine the generalized force curve. In this steps 5 parameters need to be determined: S^M, $S*$, S^S, F^M and dF^0. These are key parameters that needs to be tuned for the model to correspond to the Pacejkas magic formula. More on this further down in the report.
As described by [34] the generalized forces can be described by

$$F(S) = \frac{dF^0 S}{1 + \frac{S}{S^M}(\frac{S}{S^M} + \frac{dF^0 s^M}{F^M} - 2)}, 0 \le S \le s^M \tag{35}$$

$$F(S) = F^M - a(S - S^M)^2, S^M < S \le S^* \tag{36}$$

$$F(S) = F^M + a(S^S - S)^2, S^* < S \le S^S \tag{37}$$

where

$$a = -\frac{dF^0}{s^M}(\frac{F^M}{dF^0 S^M})^2 \tag{38}$$

and this is done in simulink utilizing three different equations, two of which will always be equal to zero.

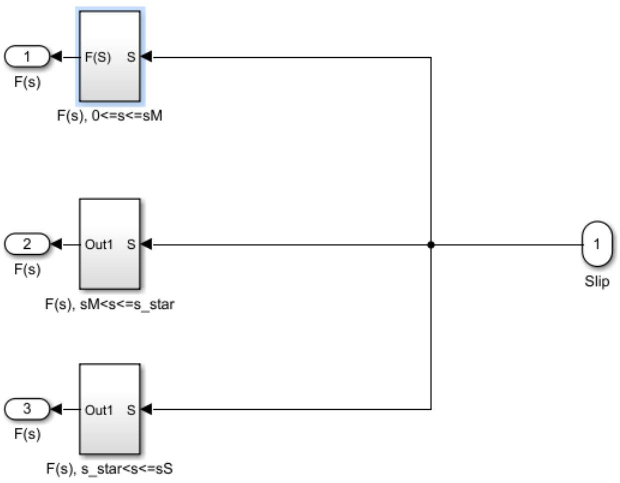


Figure 36: Generalized forces.

The image below shows how two of the equations will always be zero, the boolean will only be true for one interval.

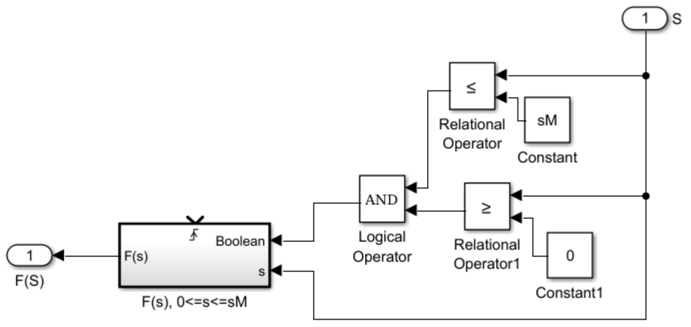


Figure 37: Generalized forces example, interval described by equation (35).

The block F(S), 0<=s<=sM computes *F(S)* as described by equation (35). The forces in the different directions can now be determined as TMeasy describes forces only with the generalized forces and the slip in a certain direction. What remains now is only to translate the tire forces upon the wheel. Unfortunately this project failed to verify good behavior for the TMeasy to justify the switch. This comes down to two aspects, the generalized force and the transference from the tire to the wheel in a satisfactory manner. The generalized forces would need to be properly tuned such that the forces are similar to the forces provided by Pacejkas magic formula. The transference of the forces to the wheel also needs to be investigated further.

5.3.3 Shifting of The Velocity

Since the model is unstable at lower speeds and stable at higher speeds and all attempts to stabilize lower speed behaviors failed a natural solution is to shift the velocity of the motorcycle. The shift must be done such that the motorcycle simply does not run at lower speeds. A solution where torque is applied to the back wheel is chosen. To make sure that the speeds never drop below a threshold lever a P regulator was implemented.

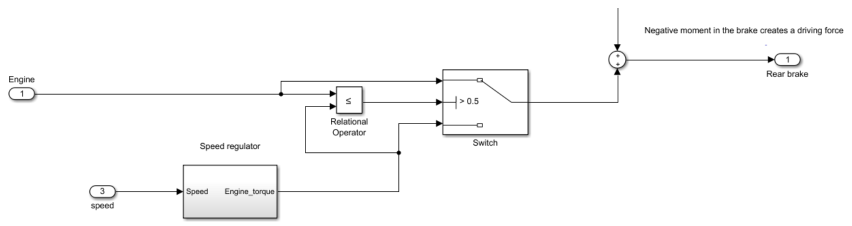


Figure 38: Switch determines which of the torques is the lowest.

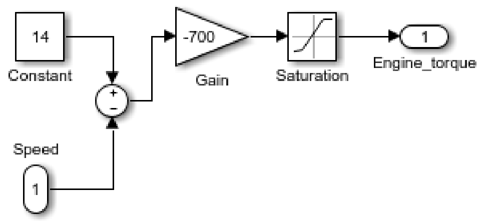


Figure 39: P-regulator, applies torque required to keep MC above 14m/s. Saturation set to 0.

The regulator works out which torque is smaller where negative torque means forward movement. The applied torque from the engine or th e torque required to hold the bike above the threshold value. If the torque of engine is not sufficient then the regulator kicks in and transfer enough torque to keep the bike going threshold speed. The braking torque of the motorcycle is then applied before the combined engine torque is applied to the back tire, and consequently the rest of the model. One could consider braking a disturbance. The P regulator is tuned so that the motorcycle is still above the low speeds even though both brakes are applied.
Aerodynamics is also changed to accommodated for this change in speed such that speeds are considered relative from the threshold speed, see Figure 40.

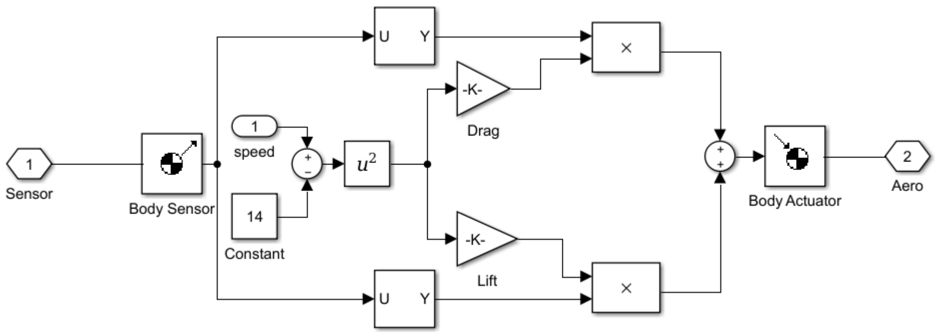


Figure 40: Aerodynamics set to compensate for the shift in velocity.

5.3.4 Velocities and Accelerations in Relation to The Motorcycle

As previously discussed the transversal velocities and accelerations are in relation to the absolute coordinate system. To make this useful for the Washout filter these need to be transformed into relation to the bike. As the accelerations in the x and x are known it is only a question of converting them into their components, assuming that you consider it a rotation in one plane.

$$a_{x'-mc} = a_x \cos(\gamma) - a_y \sin(\gamma) \tag{39}$$

$$a_{y'-mc} = a_x \sin(\gamma) + a_y \cos(\gamma) \tag{40}$$

where a_x, a_y are the accelerations in the x and y world-axis and γ is the yaw. While the $a_{x\text{'-}mc}$, $a_{y\text{'-}mc}$ are the acceleration rotated around the z axis, that is accelerations in regards to the bike.

5.3.5 Powertrain Model

The Simulink motorcycle model used a static gain combined with an input to generate a torque that was a percentage of a maximum torque of 750 Nm at all speeds. This meant that there was no simulation of how the motor behaved at different speeds and different gears. A motorcycle should be able to deliver different torque dependent on the rpm and gear to act realistically. Sound is also a major part of the simulation experience and the engine sound changes with different rpm which should change dependent on the current speed and what gear that is used. This meant that it was important have a functioning model of the powertrain to be able to deliver a more realistic experience. The model that was built is based on the same motorcycle that was used to build the physical part of the simulator, the Suzuki GSF 600 Bandit 2000[2].

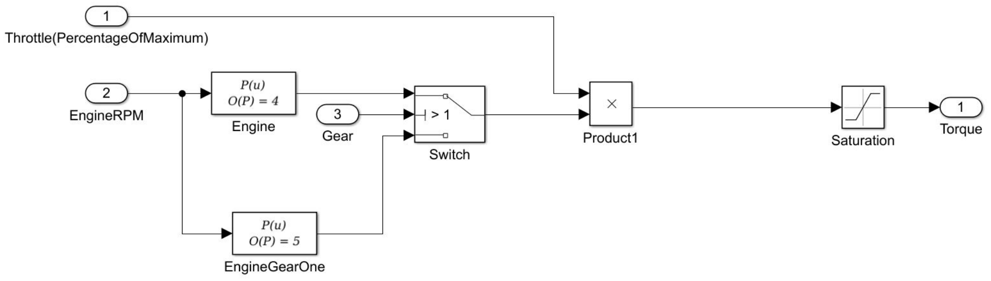


Figure 41: The developed engine model.

The engine was modeled by using the torque-rpm curve for the real engine. This was done by taking out torque values at different rpm and the doing a curve fitting using a 4th degree polynomial in Matlab, the degree was chosen based on how well the fitting corresponded with the expected behavior of the engine, but it also need to be able to generate torque at low rpm while running on gear 1. This created the need to design another engine model specifically for gear 1. That model was based on acceleration and torque data at low speeds. The model for the engine while running gear 1 became a curve fitted 5th degree polynomial that uses saturation to stay within the correct torque. The model then uses the current rpm of the motor and calculates the maximum torque it can deliver at this time. This value is then combined with what percentage of torque the driver wants by using data received from the sensor on the physical motorcycle. The resulting engine model can be seen in Figure 41.

The gearbox model was designed with an array of switches in Simulink and also uses a static primary and secondary gear using the gear ratios of the reference motorcycle. The primary gear is what connects an engine to the gear box and the secondary gear is what connects the output shaft of the gearbox to the back wheel of a motorcycle these two gears are constant and are in no way dependent on what gear that is used. The model was done as a heavily simplified model of a gearbox and does not take any spring or dampening behavior of a real gearbox. It also does not take any losses into account meaning that it became a very ideal model of a gearbox.

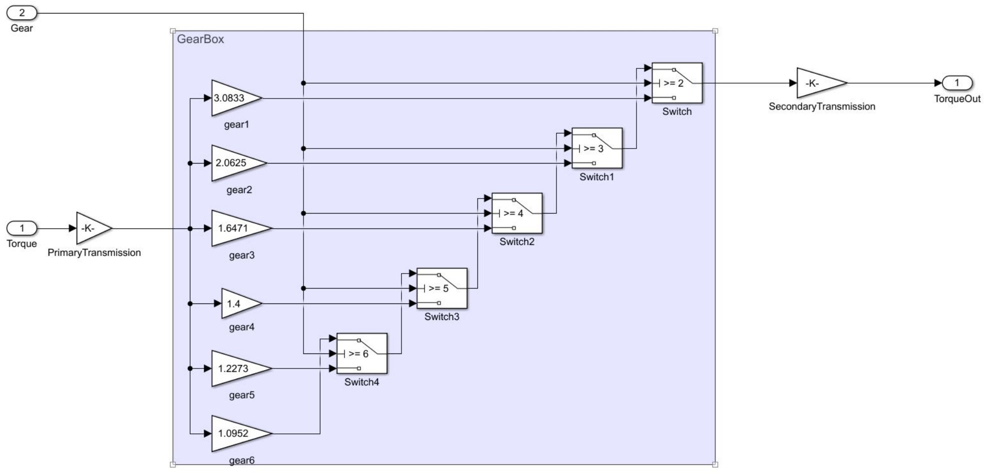


Figure 42: The developed gearbox model.

The gearbox, Figure 42, receives the current gear and then applies the correct gear ratio to the generated torque delivered by the engine model and the outputs the torque that should be applied to the back wheel of the model. The goal was to use a manual gearbox that uses inputs for the physical motorcycle to change gear but due to time constraints the physical system for changing gear was never implemented. This resulted in the development of a Matlab function that uses the current speed to control a state machine which sets the gear that should be used. The script was set up with some overlap between each gear to make sure that the gear does not jump back and forth when the speed oscillates around the limit of one state. The speed limits for each gear was decided upon based on the acceleration curves for the gears to generate optimal torque at all times during acceleration. The design with the Matlab function is shown in Figure 43.

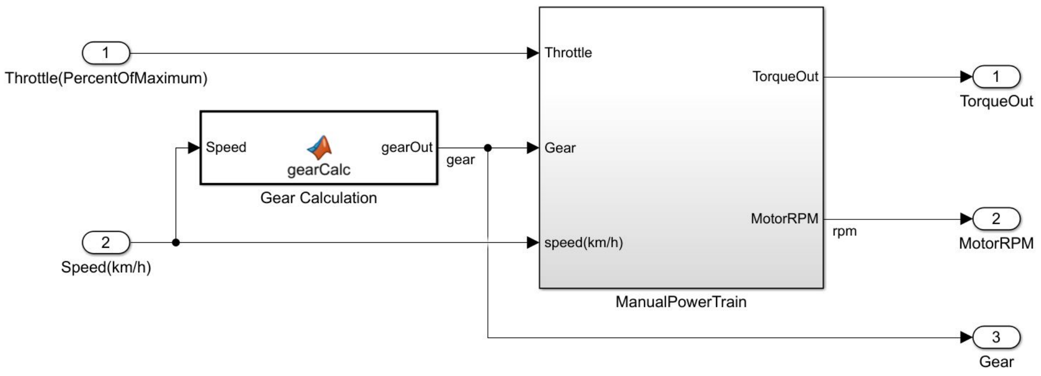


Figure 43: Top level of the powertrain model.

Since the engine model uses rpm to calculate its output torque the rpm needs to be calculated and then fed back into the model. This was done by using the motorcycle models current speed and dividing it into angular velocity on the back wheel by dividing with the radius of the back wheel. This angular velocity was the taken through the gearing and converted into rpm to be used as an input to the engine.

5.4 Motion Cueing

The motion cueing for this project was made as simple as possible. A washout filter was chosen and used to translate motions from the motorcycle model to the motion simulator platform. To increase immersion a crash function was implemented for physical feedback on the user when crashing in the virtual environment.

5.4.1 Building a washout filter

A classical washout filter was chosen for this project as with no accurate head tracking the other washout filters would not be feasible and it was the quickest to tune, test and verify. The scalars and time constants were tuned using a heuristic approach and changing values in real time. To avoid common motion cueing errors such as false motion cues and lag the guidelines in [8] were used. The washout filter was then tuned to stay within the motion simulators actuators limits and then the driver's experiences of the simulation.

5.4.2 Choosing Scalars and Limiters

Before the signals are passed through the washout filter they first need to limited and then scaled. The limits are used for safety reasons while scalars are used to create a realistic motion.
The limits on the inputs were based on real motor cycle accelerations. It takes less than 3.5 s for a fast motorcycle to go from 0 km/h to reach $100 km/h$. The limit on the acceleration should forward, a_x therefore be:

$$a_x = \frac{\Delta v}{\Delta t} \approx 7.9 m/s^2 \tag{41}$$

The acceleration a_y due to turning was first limited by a turn with a radius $r = 80m$ in the virtual reality environment with the speed of 100 km/h. The following max acceleration for a_y was then found:

$$a_y = \frac{v^2}{r} \approx 4.9m/s^2 \tag{42}$$

The limit on the acceleration a_x is larger than the motion platform can perform(See Figure 8) and was therefore limited to the max value 6 m/s^2. The limit on a_x and a_y were later limited even further as the actuators in the motion platform had issues performing with higher limits. The acceleration in z was limited to 3 m/s in case of crashing to allow for more movement in other directions.

A motorcycle can lean over 45° to the sides (roll). The dynamic motorcycle model cannot handle such rotations and is limited to leaning at a maximum of 20°. The roll was therefore limited to the maximum angle allowed for the motion simulator. The limit for the pitch angle was set by testing worst case scenarios including but not limited to crashing, accelerating from 0-100 and breaking violently.

Non-linear scalars in the form of fourth order polynomials were tested but proved to difficult to tune for all dimensions and were ultimately scrapped. The scalars (gains) for the transfer functions in the washout were found by iteratively increasing and decreasing their values for the accelerations and angular rates to be applicable on the motion platform. The scalars were than changed again to create a realistic sensation for the operator. If new time constants were changed the gains were altered if again. This process was repeated till satisfactory result for the gains were found.

Note that scaling a_z with the constant K also scales the gravitational acceleration. Consider the motorcycles acceleration in the z direction to be a_{zmc}. The acceleration a_z is then:

$$Ka_z = K(a_{zmc} - g) \tag{43}$$

To scale the acceleration a_z with a linear gain K the following strategy is recommended:

$$a_z = Ka_{zmc} - g \tag{44}$$

5.4.2.1 Translational Channel

High frequency responses such as increasing the throttle quickly, the vibrations from the dampening of the fork and breaking were used as starting point for designing the high-pass filters. The minimal order for the translational high-pass filter is $n = 3$ as mentioned in equation 21. The translational high-pass filters were then chosen to be of the second order with a first order return to zero for more design options:

$$W_{HP} = \frac{\omega_{HPT}^2}{\omega_{HPT}^2 + 2\zeta\omega_{HPT}s + \omega_{HPT}^2} \tag{45}$$

$$W_{RTZ} = \frac{s}{\omega_{HPT}s + 1} \tag{46}$$

$$W_{HPT} = W_{HP}W_{RTZ} = \frac{\omega_{HPT}^2}{\omega_{HPT}^2 + 2\zeta\omega_{HPT}s + \omega_{HPT}^2}\frac{s}{\omega_{HPT}s + 1} \tag{47}$$

When choosing time constants for W_{HPT} the time constant ω_{HPT} was incremented for a faster response until the users felt it was satisfactory. The problem then would be that the return to zero would be too fast or too slow. By using the RTZ as a W_{RTZ} buffer the motion back to neutral could be slower or faster. Unique time constants for each direction in x, y and z were tested and found.

5.4.2.2 Rotational Channel

In the rotational channel only the pitch angle θ is filtered. The yaw angle could not be implemented in the motorcycle as it would interfere with the camera in the visual environment. In a motorcycle the roll angle does not return to zero for longer periods of time. Rather than adding a low-pass filter for roll (this would generate another dimension to tune; the gap between high frequency rolls and low frequency rolls need to be very small) the true roll from the motorcycle model was used. The roll in the motorcycle is forced to stay stable and cannot tilt more than 17.3 degrees. By utilizing this fact the need for a high-pass filter for the roll angle ϕ was not needed.
The rotational channel was kept at the lowest order possible to make it easier to tune the washout filter:

$$W_{RC} = \frac{s}{\omega_{RC} + s} \tag{48}$$

The time constant were found by finding a trade off between responsiveness and a smooth returning to zero.

5.4.2.3 Co-ordination Channel

The co-ordination channel uses low-frequency translational accelerations to simulate continuous accelerations. A second order filter was used for this purpose:

$$W_{CC} = \frac{\omega_{cc}^2}{\omega_{cc}s^2 + 2\zeta_{cc}\omega_{cc}s + \omega_{cc}^2} \tag{49}$$

The time constants for W_{CC} were chosen so to be slow. This was done as to not create large "jumps" for roll and pitch angles on the motion simulator as they would prove to be jarring for the user. The low frequency accelerations were than transformed Euler angles with equation 26.

For the acceleration forward to be sensed by user and not the rotation a limit on the rotational speed was used. By reducing the rotational speed to a maximum $0.2 rad/s$ the rotation cannot be felt [26].

5.4.3 Crashing

The crash function uses input directly from the virtual environment in Unity. When the motorcycle in the visual environment fatally collides with a virtual object a crash signal is sent to Simulink. A direct movement backwards is then made to simulate a crash by ordering the motion simulator platform to change its position. Note that this is a forced changed of position and not an acceleration that is fed to the motion platform. The crash function can be found in appendix D, Figure D83.

5.4.3.1 The Complete Washout Filter

Each channel and each direction in the channels were first tuned separately in real time with input from the user. After usable values for each direction in every channel were found the whole washout system would then be tested and then tuned. The time responses were satisfactory but issues arrived when more than all dimension was used. The motion platforms actuators would saturate to produce a motion effectively making the user feel no motion at all at certain times. To remedy the this the scalars were changed and the whole tuning process repeated.

5.5 Driver Interface

To be able to create an immersive experience the input from the driver to the system needed to be as close to the real input devices as possible. The handlebar is however not only an input to the system but also an output that the driver uses to get a good feeling of how the motorcycle behaves and while taking a curve this becomes extra important.

5.5.1 Inputs From Driver

To get as realistic input devices as possible, the original handlebar and pedals were kept as intact as possible and then the original movement were read using sensors. The throttle had two wires where one retracts and one extends when throttling. The one that retracts when throttling was chosen as the one to be read as this enabled a spring to be added to mimic the original behavior of the throttle returning to zero due to springs on the original motor. This construction is seen in fig 44. The torque in the throttle could then be tuned by tension of the spring by moving the screw connector to the right to create a realistic behavior. The construction was also done for the clutch for future implementation of the gearing system.

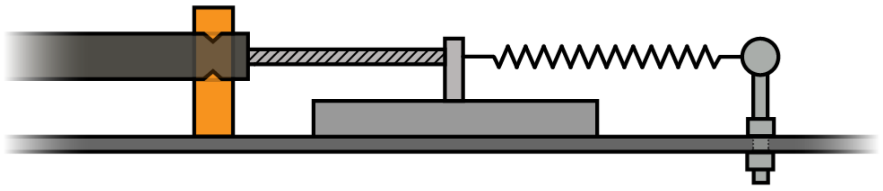


Figure 44: Wire-reading sensor.

The measurement of the hand and foot break were done by fixating a potentiometer to an axis of rotation for the movement and reading the rotation of the pedal related to the axis. It is seen in fig 45 were the movement of the potentiometer matches the movement of the brake.

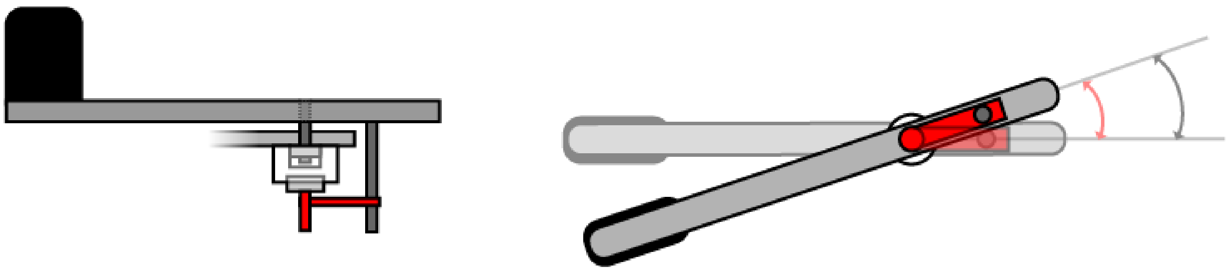


Figure 45: Brake-reading construction, potentiometer is marked with red.

The steering was read with a potentiometer fixated to the motor axis and the frame. This gave high enough accuracy at most situations as it used the gearing to have a larger change of angle from a small change in the steering.

For safety reasons a start and stop button was assigned to a toggle on the handlebar. Using this the user can tell the system to enable or disable the rig. This enabled the user to stop the hydraulic rig if the user felt even the slightest discomfort.

5.5.2 Sensor Box Design

The part of sensors, especially the throttle and clutch sensor, Arduino micro controller, cables and experiment board which used to connect sensors and Arduino need to be contained and organized properly below the seat. In order to simulate the feelings of returning to zero throttle features, the spring fixation attachments are added to the sensor box as well. The returning force of the throttle could be changed by adjust the position of spring fixation attachments in the slots with the same spring. As Figure 46 shows, the Arduino is fixed with bolts between the two slide potential meters (the dotted area) and the experiment board with cables is covered on the Arduino by inserted pins. The board is attached to the motorcycle frame through the 2 bolts on the each sides. In addition to this, the outer layer of the wire needs to be fixed to prevent the wire shaking and only enable the movement of the inner wire which is connected to the spring, as seen in Figure 44. The sensor box material is chosen as wood based on the consideration of prevent Electro-Magnetic Interference (EMI). All the end of the cables or wires are covered with electric tape to improve robustness in the system.

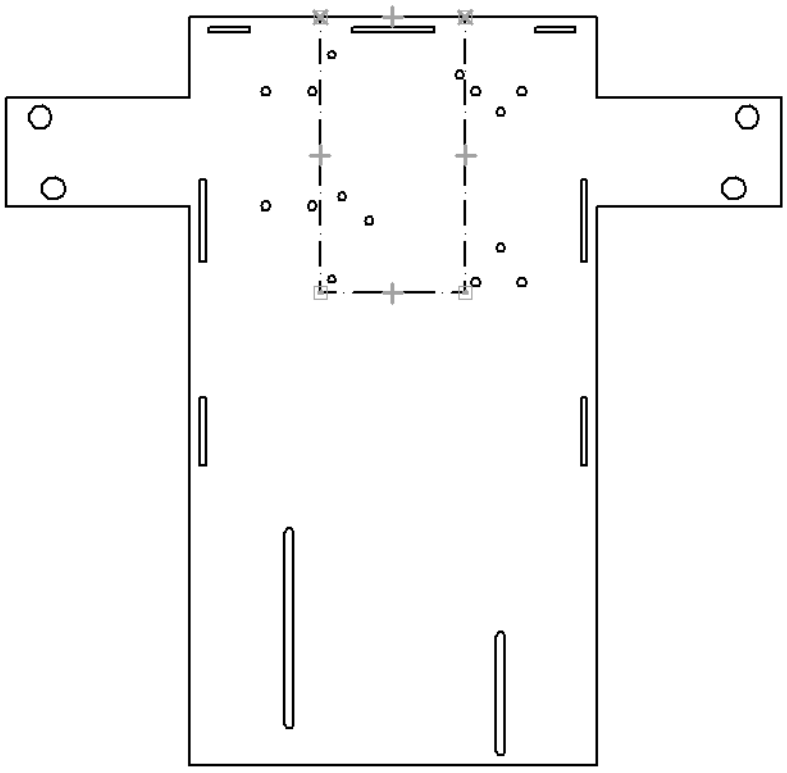


Figure 46: The sensor box base board.

5.5.3 Hardware Handlebar Torque Feedback

The final solution regarding the hardware for the handlebar torque feedback consist of three parts which will be explained in this section. Those three parts are the motor, the transmission and the motor driver. The chosen motor was a Dunkermotoren DC-motor GR63x55 245 watt combined with the PLG52 gear box. Dunkermotoren is a well-known motor brand that sells reliable products for a reasonable price which is why it was chosen. Table E5 shows specifications for the motor B. The transmission is a combination of a gear box and a cable transmission. Otherwise the ratio

Table 1: Torque Feedback parameters.

Nominal torque	0.27 Nm
Torque constant	0.064 Nm/A
Nominal current	4.9 A
Max current	33 A
Nominal voltage	24 V
Nominal speed	3000 RPM

of the cable transmission had to be very large to get enough torque to the handlebar. A 1 mm stainless metal wire was used for the cable transmission. It was not pre-stretched so one of the end points had to be adjustable to keep the wire tight over time. The capstan was made using PLA plastic and the drum was a combination of plastic and steel. The capstan had grooves on its surface to increase the friction coefficient in equation 33 since this gives a larger surface area touching the wire. The grooves were made in a v-shape causing the wire to press itself into the capstan during load which further increases the friction. The wire was wrapped 7 times around the capstan to decrease the risk of slipping according to equation 1. The final solution is shown in Figure 47.

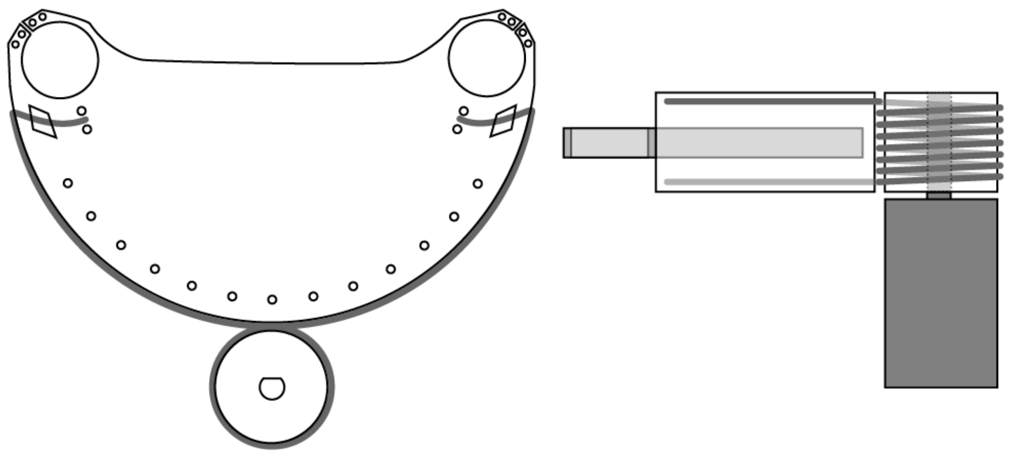


Figure 47: Final transmission solution, above- and side view.

The gearbox had a ratio of 6.25 and the cable transmission had a ratio of 5.12 giving a total ratio of

$$6.25 \cdot 5.12 = 31.98. \tag{50}$$

This gives a nominal torque of

$$0.27Nm \cdot 31.98 = 8.63Nm, \tag{51}$$

and a peak torque of

$$0.064Nm/A \cdot 33A \cdot 31.98 = 67.5Nm. \tag{52}$$

A full turn from left to right corresponds to a 70 degree angle which is close to one fifth of a full turn. Considering the nominal speed and the total gear ratio, the motor can turn from left to right in

$$\frac{3550}{60 \cdot 5.32} \cdot \frac{70}{360} = 0.11 seconds. \tag{53}$$

The Sabertooth 32A motor driver was used to control the dc motor. The switches on it was set in Serial Mode to control it with an Arduino Mega. Using the DEScribe software, the current was limited to 15 amperes, current limit smoothing was set to zero as well as ramping to minimize delay in the system. The integer value from Arduino to Sabertooth were between -127 and 127 where 0 is straight forward and minus sign gives reverse direction. The higher the absolute value, the higher the voltage causing higher torque.

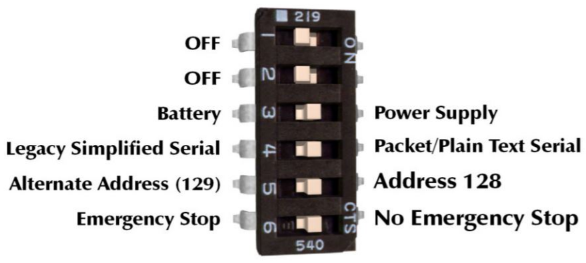


Figure 48: Sabertooth motor driver setting.

5.5.4 Torque Control

The handlebar torque feedback controller receives its reference from the motorcycle model and then controls the DC-motor that is connected to the handlebar to deliver accurate torque to be felt by the user. For the controller angular velocity was chosen as feedback. This decision was made based on what kinds of sensors that were available and due to time constraints. There was already an incremental encoder mounted on the motor shaft which reduced the time to implement the controller and thus the whole system. The goal was to use a current sensor for feedback but since such a sensor could not be found in time that would work well in this system. The only other option was to build the circuit, which would have resulted in a noisy sensor. The angular velocity approach was chosen instead.

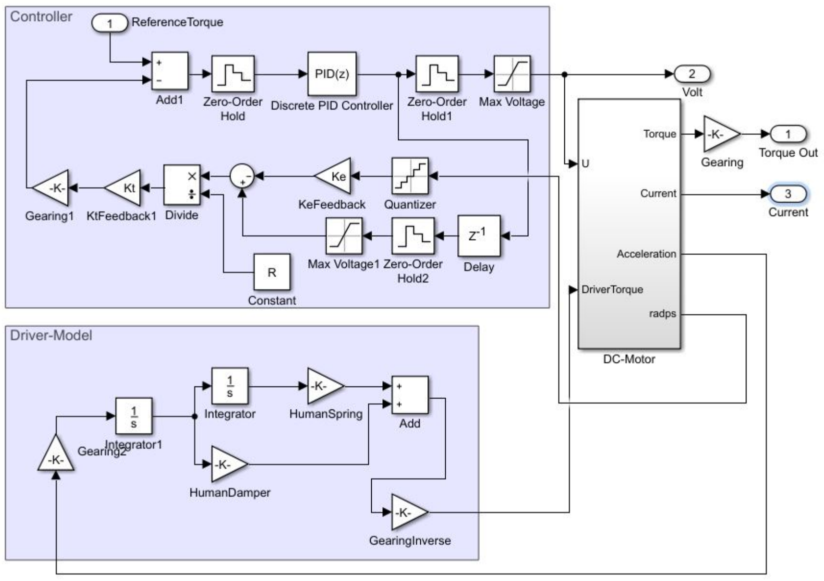


Figure 49: Development model used for tuning of the torque feedback controller.

For development of the feedback controller a model-based approach was chosen. The DC-motor, the physical transmission and the driver's arms and reactions were modeled and then connected to the motorcycle model for development and tuning of the model. For tuning an experimental method close to the Ziegler-Nichols method was used. The controller was first tuned in real time simulations, then the simulation model was changed to evaluate the controller in discrete time, see Figure 49. This delivered very good results when the controller was implemented on the physical system with no additional tuning being required. The resulting control functions are shown in equations 54 and 55.

Error calculation, where current is calculated using the last output voltage and multiplied with the torque constant and the gearing in the system [11].

$$e(k) = r(k) - 31.98 \cdot K_T \cdot \frac{(u(k-1) - K_E \cdot \omega)}{R} \tag{54}$$

Control signal calculation, where $a = K_p \cdot \frac{K_i * T_s}{2}$, $b = -K_p \cdot \frac{K_i * T_s}{2}$, $K_p = 0.3$, $K_i = 100$ and $T_s = \frac{1}{54}$.

$$u(k) = u(k-1) + a \cdot e(k) + b \cdot e(k-1) \tag{55}$$

The main limitation of the controller as it was implemented was the frequency at which it can run. Due to stability problems with the ROS-node located on the micro-controller and the interrupts generated by the encoder it was not possible to run it faster than 54 Hz which is to low to get the optimal performance out of the controller.

5.6 3D-Graphics and Virtual Reality

This chapter will explain how and why the whole 3D environment has been built up and what features that where implemented, including everything from 3D maps to the animated human avatar. Project files and code is found on our GitHub repository [47].

5.6.1 Unity3D

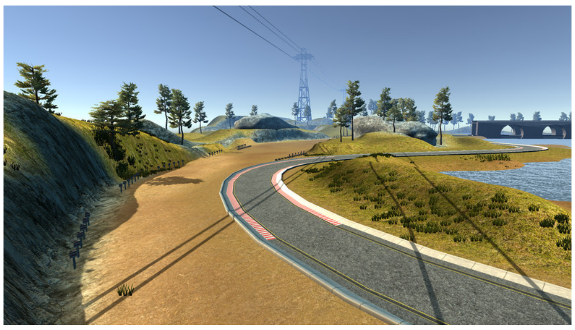
For developing the entire 3D-world that would visualize our motorcycle in the environment and take user inputs, the application Unity3D was used. Unlike the competitive game engine *Unreal Engine*, Unity3D is a very easy and intuitive program to start developing a game from scratch as well for their very rich manual [45]. As our team did not have any knowledge of 3D-game development, choosing Unity3D was an obvious choice. It also included all the features that we needed for the virtual reality.

5.6.2 Virtual Reality Environment

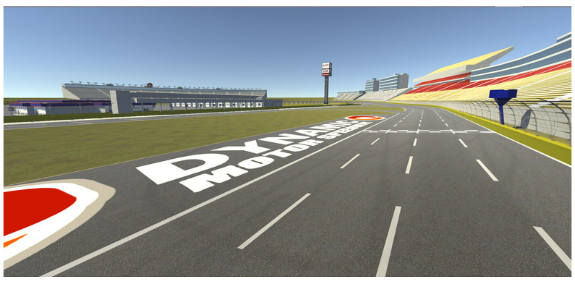
To make this motorcycle simulator as realistic as possible, virtual reality is going to be implemented with a visual environment. Virtual reality uses head-mounted goggles to generate realistic images and other sensations that simulate a user's physical presence in a virtual or imaginary environment. In this project, the VR equipment called Oculus Rift was used so that the person sitting on the motorcycle could "look around" the artificial world and interact with virtual features. With Unity3D's built-in VR-plugin[44] the Oculus Rift headset could be used directly as a display by plugging in the USB and HDMI cable, and settings some basic settings in the project settings.

Figure 50: Virtual Reality Headset.

The visual environment consists of two different pre-modeled maps. The first map added, *Lake Race Track* [27] has a flat asphalt circuit with some severe curves and various rocks, lakes and other objects. As this map was difficult to use for beginners, we had to add a second easier map, Speedway Oval Race Track [7] which was easier to use at high speeds.



(a) Lake Race Track.



(b) Speedway Oval Race Track.

Figure 51: 3D maps.

When starting the game, a simple GUI canvas [43] with buttons was implemented as a selection menu. The menu consist of a set of buttons and an image, as shown in Figure 52 below, where the user can switch between maps, choose day/night mode, calibrate VR tracking and start the simulator.

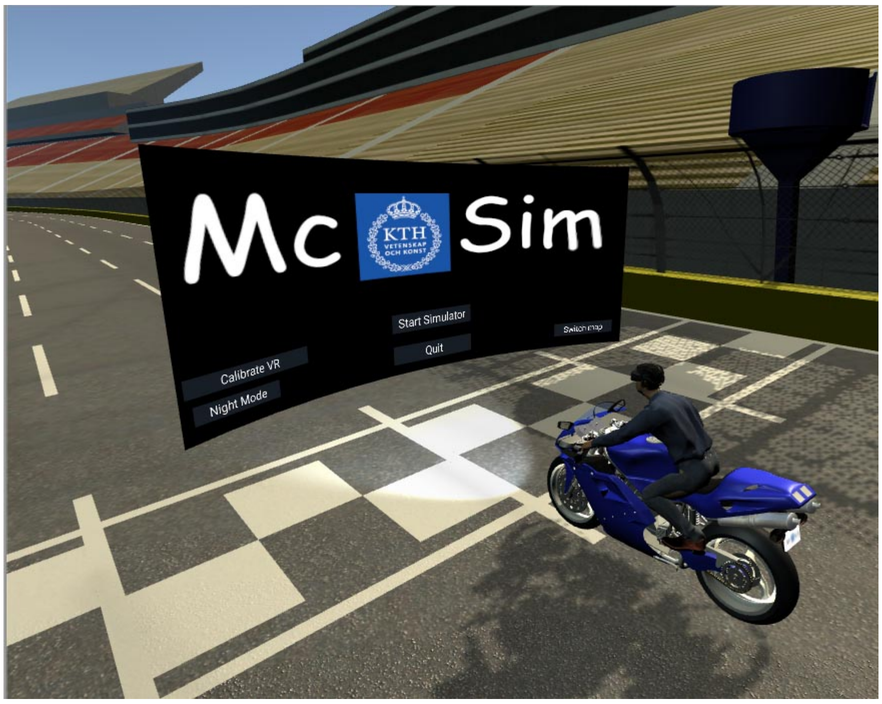
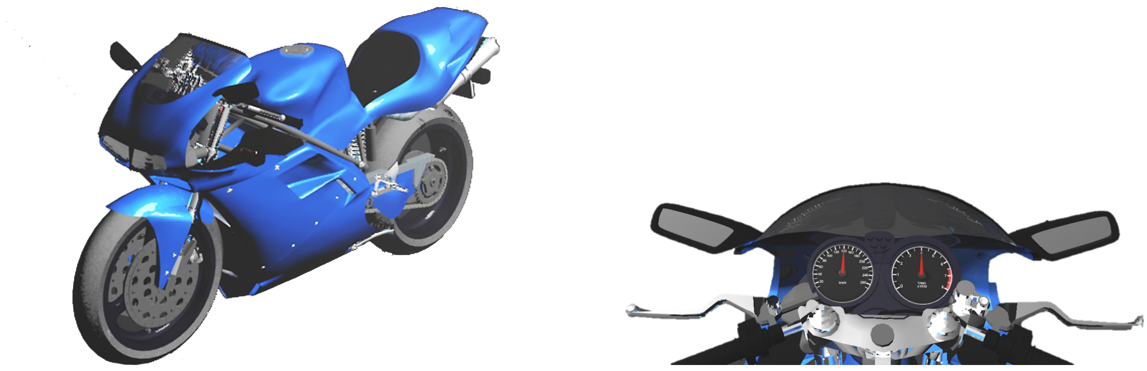


Figure 52: Main menu user interface at simulator startup.

5.6.3 Motorcycle Model

A raw motorcycle 3D-model of a Ducati [51] was downloaded and then painted and parts modified in the modeling application Cinema4D [22]. They where modified to match the real motorcycle mounted on the physical rig as much as possible, as well as separating important sub-parts so that they could easily be animated in Unity3D. Then the 3D-model was added to the map as a game-object in Unity3D and could be dynamically moved both in global state and it's internal state, such as rotating handlebar, speed/rpm indicators etc. All this behavior is controlled by the C#-script *modelcontroller.cs* where all the variables of the motorcycle state are stored and updated every frame-update, e.g. it's local pose, speed, rpm etc. These variables are updated from external scripts which handles the incoming data from the dynamic Simulink model. Some feedback variables such as user-leaning, crash-state and off-road state are instead being fetched from external scripts and sent to other nodes in the system.



(a) 3D-model representation of motorcycle. (b) First person view.

Figure 53: 3D Motorcycle model

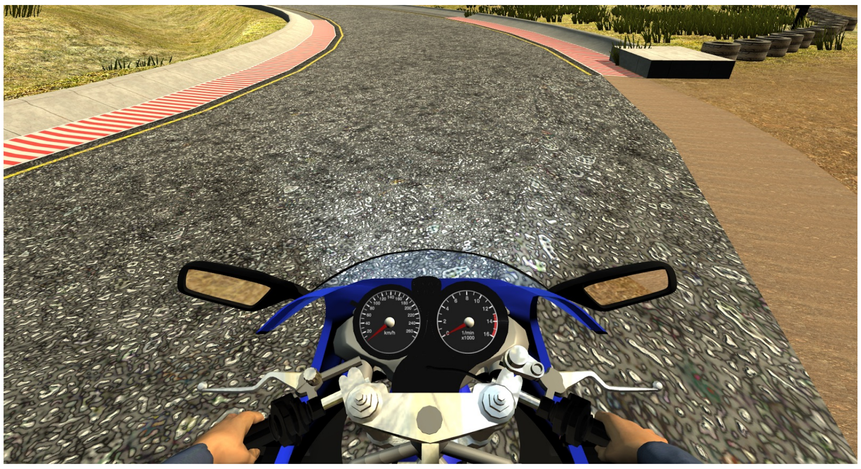


Figure 54: First person view in VR-environment.

5.6.4 In-game Dynamics

The motorcycle is set up with so-called "sphere-colliders" [42] for both front and rear tire as seen in Figure 55 below. They are used as collision and road detectors, where data such as crash, off-road detection and road-inclination could be extracted. It is possible to set spring, damper, friction and slip parameters for the wheels but as these dynamics will primarily be handled by the Simulink model, this feature is not used in Unity3D. However the vertical gravity feature is used for keeping the motorcycle on the ground, as our Simulink model cannot handle those mapping dynamics.
The crash detection system is defined so that if the motorcycle runs at a speed higher than 50 km/h and collides into something at the front-most point on the front wheel, the game stops and Unity3D sends a "fatal crash" signal to other nodes in the system. At lower speeds, Unity3D sends a "soft crash" signal to other nodes which would handle that event. The crash signal consist of an Integer between 0-2, where 0 is no crash and 2 fatal crash. The inclination of the road is calculated by taking the difference in angle of the two wheels. By having a collision detector on the asphalted road we can determine if both wheel are off or on the road. This data is then sent to the Simulink node for feedback dynamics.

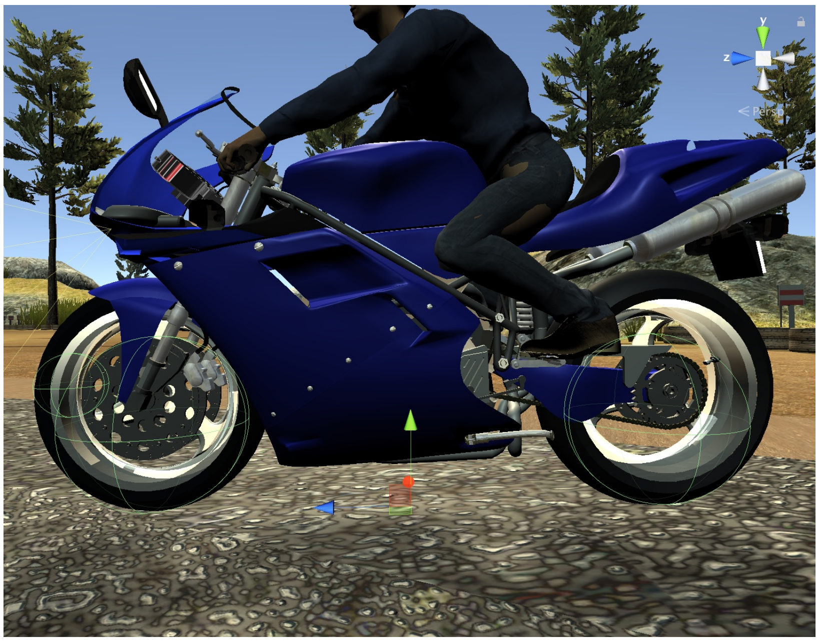


Figure 55: Road and crash sphere-collides on the 3D-motorcycle's front/back-wheels.

5.6.5 User Inputs

As the Oculus Rift headset had positional tracking 3D-space using its camera sensors, we could use that for some user inputs. The user does not have access to any mouse or keyboard for pressing buttons in-game, but instead with the provided pose of the user's head a simulated mouse pointer could be created. This interaction method is quite common in mobile VR application where the user can't access the touch screen, which is called "Gaze" input [40]. So by using your head's orientation to point and hover over a button for a specific amount of time, it would simulate a click.

5.6.6 Human Avatar

For a better looking visualization an humanoid avatar was added on the motorcycle (Figure 56) with his hands holding the handlebar, just like the user would. The hands are set as children to the handlebar object so that if the user would turn it, the hands would follow its motion. Using the motion tracking of the user's head, the head motion is copied to the avatar and the avatar's upper body could also be dynamically rotated if the user would lean forward/backwards or left/right. The rest of the human avatar could not be moved as we did not have sensors for that.

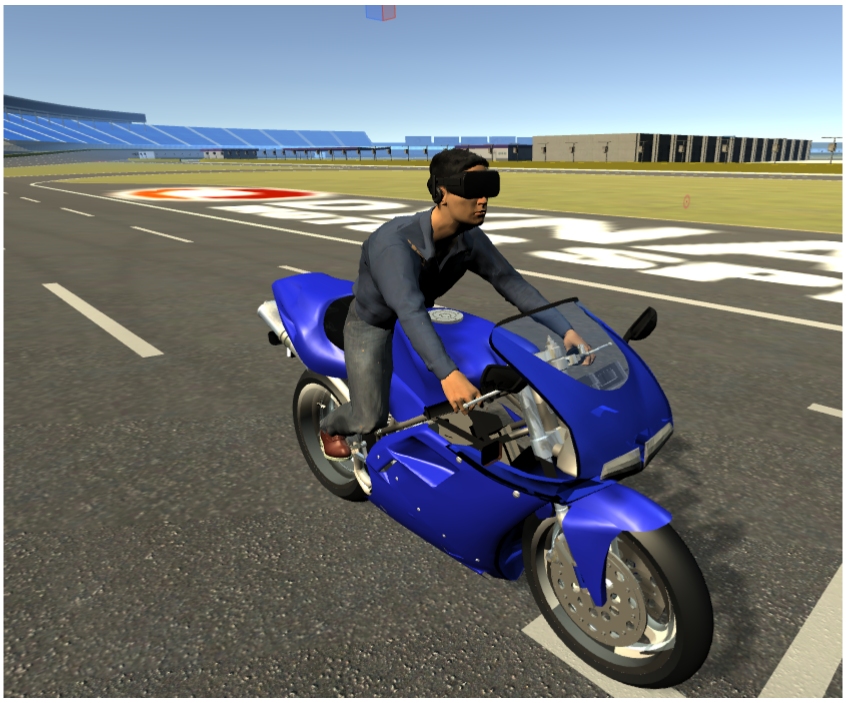
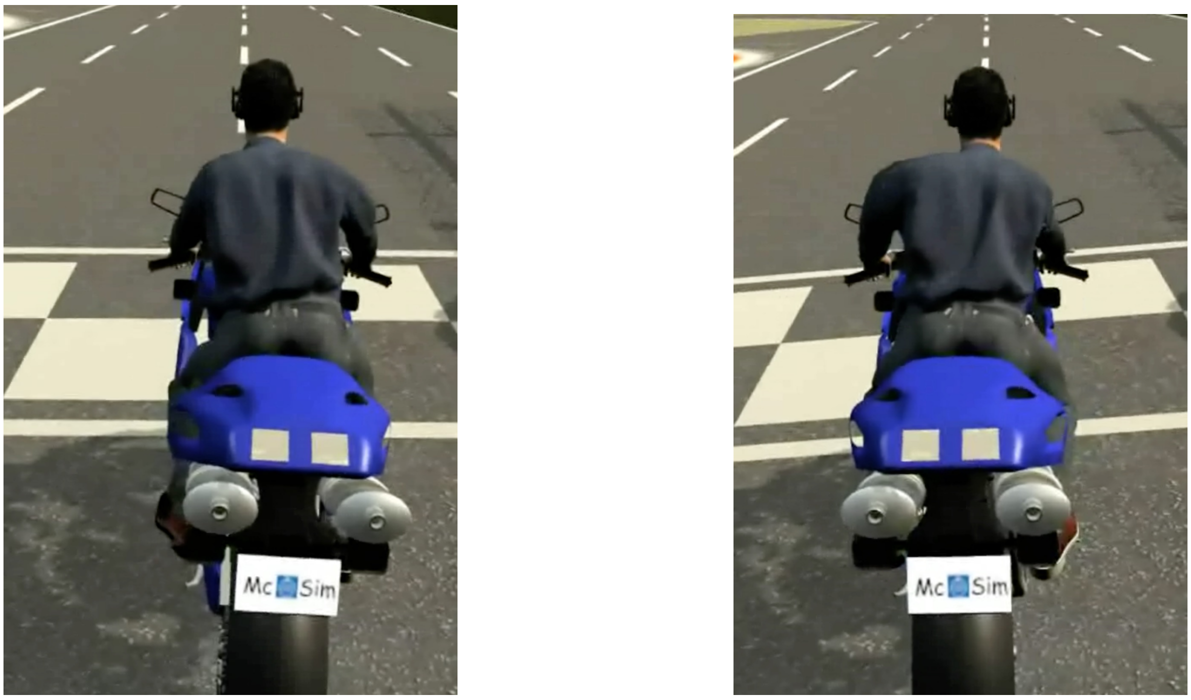


Figure 56: Representation of user as a human avatar.

5.6.7 User Leaning

The user's body-leaning pose could be estimated by measuring the horizontal right or left translational offset of the head from the center-origin of the camera. This leaning offset can then be used for additional dynamic calculations in the Simulink node, e.g. added forces on the motorcycle while turning. However due to time constraints, this feature was not used in Simulink.



(a) User leaning approximately 20cm to his left. (b) User leaning approximately 20cm to his right.

Figure 57: User leaning input

5.6.8 Motion Rig Filter

One problem that arises when using VR and motion platforms is that the motions from the rig can disturb the 3D-view of the user. E.g. if the motorcycle makes a movement by pitching upwards to simulate accelerating forces on the body, the user's distance to the 3D motorcycle model in VR would increase as the model does not pitch in VR. This also occurs while the rig is rolling. This problem can infer VR-sickness and had to be solved to meet the requirements, R 12 (Cyber-Sickness).

Therefor, the pose of the user's view (VR-camera in-game) is moved and/or rotated depending on the rig's rotation as illustrated in Figure 58 and 59. Note that the the VR-view is rotating around the same point as the rig's center of rotation as shown in Figure 6. The following equations used in the Unity code, calculates the new pose of the VR-view from the offset. P and R is the pitch and roll of the rig in degrees respectively.

$$Camera_{Position} = \begin{bmatrix} x \\ y \\ z - \sin\frac{P\pi}{180} \end{bmatrix} \tag{56}$$

$$Camera_{Rotation} = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z - R \end{bmatrix} \tag{57}$$

where $x, y, z, \theta_x, \theta_y, \theta_z$ represents the current pose of the camera.

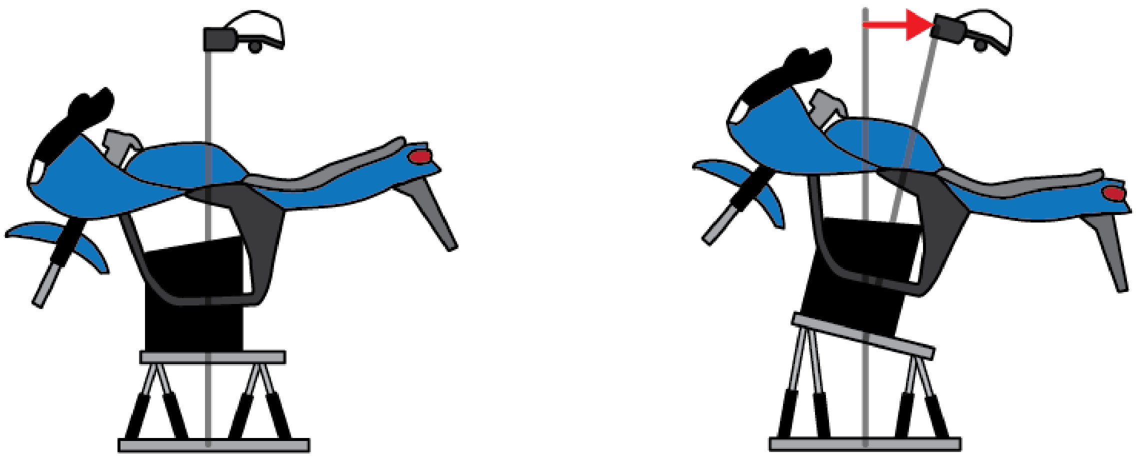


Figure 58: The rig in normal- (left) and pitch- (right) position.

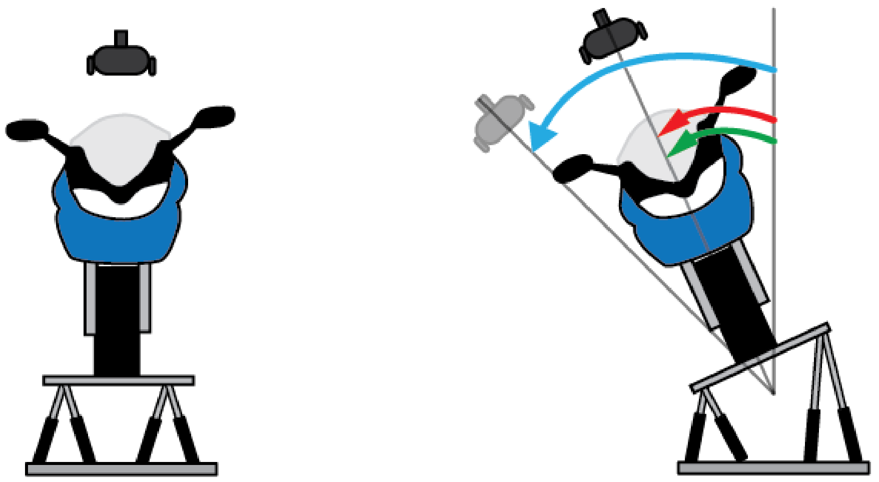


Figure 59: The rig in normal- (left) and roll- (right) position. Blue arrow shows angle of virtual headset due to physical and virtual rolls additive behavior.

5.6.9 Communication and Integration with Simulink

This Unity node is only communicating with the Simulink node in the system, using ROS. Everything that will make the VR-environment change is mainly controlled by Simulink. Both inputs from the user such as throttle/brake and the motorcycle dynamic behaviors such as roll/pitch/yaw are received by Unity. Data is also sent back to Simulink, such as crash-detection, user-leaning, off-road detection etc. These input and output subscribing/publishing are handled in arraysubcsribe.cs and publisher.cs by using the ROSBridge library explained in chapter 5.8.8. To make the system as complete and integrated with Unity as possible, some features and error handling where added. For example, if the motion-rig is disarmed by the user's red switch or by the Simulink model, the VR-environment will restart the scene and go back to it's main menu. Also, if the Simulink model crashes for any reason, the VR-environment will pause and display a "Game Over" interface.

5.7 Sound Engine

To increase immersion, sound is a must for the user to experience speed, gearing, accelerations and brakes. By using the same tool as for the visual development Unity3D, the sound engine could easily be implemented in the *EngineSound.cs* C#-code by studying Unity's sound manual[41]. The audio clips (.wav, .mp3) are attached to objects called *audio-sources* which are placed near the source of the audio in the 3D-environment.

5.7.1 Motor and Gearing Sound

By knowing the torque of the motor and looping a constant audio clip of a motorcycle motor at 4000 rpm and then pitching the sound, a motor sound was generated. For lower rpm:s, the pitch would decrease and the opposite for higher values. To get as realistic sound as possible the pitch-gain interval was found after some testing to be best at [0.4-3.4]. The following equation shows the relationships between the audio clip's pitch level and the current motor rpm denoted ω,

$$pitch_{motor} = \frac{3\omega}{\omega_{max}} + 0.4, \tag{58}$$

where ω_{max} is the maximum torque that the MC's motor can deliver, in our case $14000 rpm$.
If the current torque is zero, then the motor audio source is stopped. If the torque goes from zero to a higher value, a single additional motor ignition sound is started. When gearing up/down, two simple clutch sounds are initiated.

5.7.2 Braking Sound

The braking sound is done by using the value of the user's front-braking input state [0.0 - 1.0] where 1 is full brake, and a sound clip of a motorcycle braking and skidding on asphalt. If the user brakes at 30% or more, the sound will be initiated and also pitched depending on the current braking-state by the user input. The back brake state is however not used/implemented for this sound because of time constraints, but should be added for a more realistic sound. The pitch-level of the brake-sound is set to equal the braking-value while the user is holding the front-brake, thus at the interval [0.3 - 1.0].

5.7.3 Other Sound Effects

By knowing if the motorcycle's tires are currently on the main road or outside, a simple looping vehicle-on-gravel audio clip is played during the off-road state. If the 3D-model crashes into some other game-object in the virtual environment, a vehicle-crash sound effect is initiated. Some audio sources and clips such as birds, city noise and music are also added to the maps and main menu.

5.8 ROS as a Communication Infrastructure

All systems working on multiple machines requires robust communication. A real time infrastructure that can provide low latency, distribution of information and synchronization. Building the highway between all the entities. The publishers decides what frequency their corresponding topics will have, as the subscriber will run its callback function every time it receives a message. ROS uses both TCP and UDP for the communication, in this project mostly TCP will be used as the protocols used to communicate with external software only supports TCP.

5.8.1 ROS Master

The ROS master handles all the naming and addresses for the rest of the nodes. The main purpose of the master is to pair the nodes that wish to communicate, once this is done, the nodes will establish a peer-to-peer connection. A RPI (Raspberry PI) will be used to setup a ROS master, the RPI running Linux Ubuntu Mate 16.04 with ROS Kinetic installed.

ROS Nodes & Topics

A node is a participant in the ROS network that can share its information with other nodes. The information flowing in the ROS network are shared between the nodes using defined Topics. A node can publish and subscribe to multiple topics in a system.

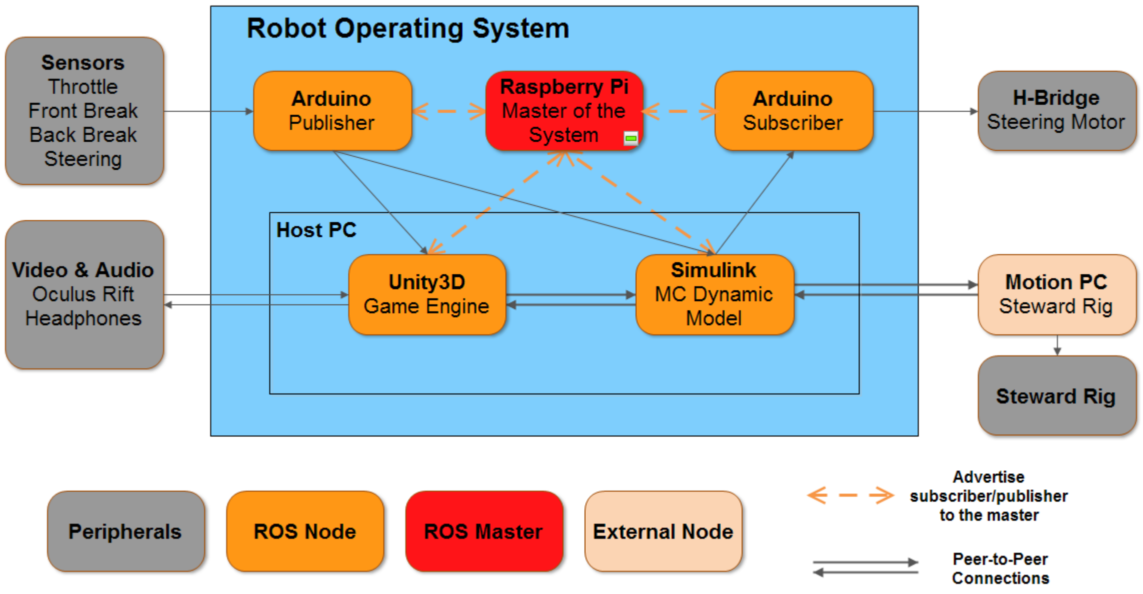


Figure 60: System Architecture.

The system architecture can be seen in Figure 60. Every ROS node that sets up a peer-to-peer connection with another node have a preset topic, the preset topic defines the message structure and the custom messages needs to be configured on every node that will participate in that specific topic.
The only part of the entire system not entailed by the ROS-network is the motion computer. This is due to a desire not to tamper with the motion computer, since any accidental tampering with core feature could potentially results in weeks of repairs. Therefore the decision was made to work with the interface already in place on the motion computer.

5.8.2 Custom Messages

Custom messages are used to setup a specific structure for the system demands. The custom messages are created in a Linux environment using ROS, the ROS master in this case. Then the custom messages are sent to its corresponding application (Matlab/Simulink, Arduino and Unity3D environment). More details for the specific application are explained in each sub-system chapter.

5.8.3 Declaring Environmental Variables

As ROS will be run on different machines in a local network, network variables needs to be addressed. This is to make the ROS master visible to the rest of the network. Setting up the ROS_MASTER_URI will direct all the nodes to the master on the network. Moreover, every node needs to declare their own address, either ROS_HOSTNAME or ROS_IP will do, ROS_HOSTNAME will take precedence if both are used. This procedure is similar in Simulink and Unity3D.

5.8.4 Wall Clock & ROS Time

Wall clock server is needed to synchronize the time between all the machines in the network. This is not necessary to get the system running, although the wall clock makes it possible to accurately measure the delays between nodes on different machines on the network. ROS also have a built in header that includes a time stamp, if these stamps are setup correctly, the delays can easily be displayed over the whole ROS network on any ROS node. ROS also have a built in clock that runs on every ROS node, the ROS time can be used as a stamp on a published message, the subscriber then compares the stamp with its internal ROS clock and the delay can be extracted.

5.8.5 Data Centric Overview

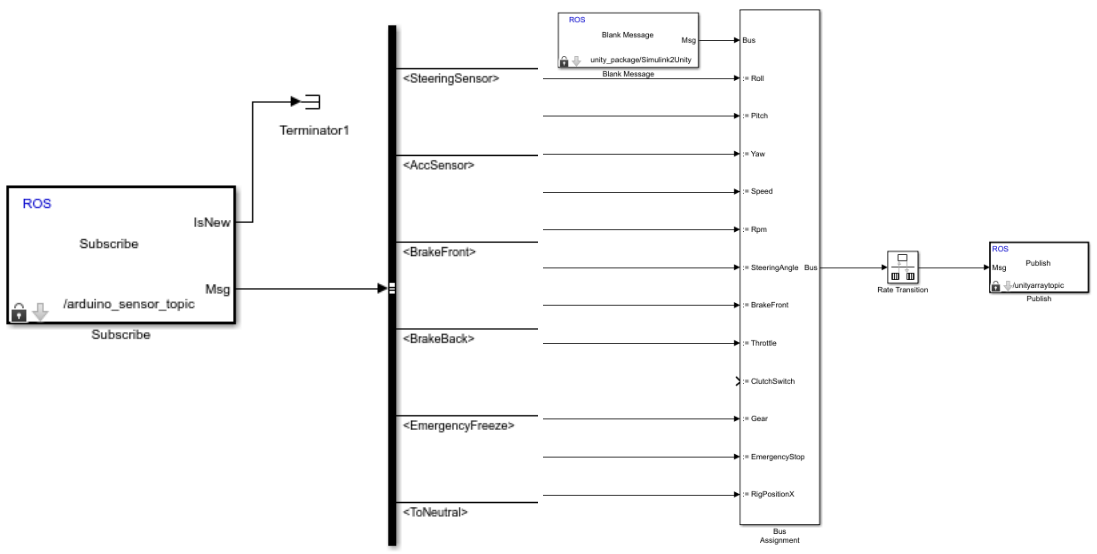
There are multiple features in ROS that facilitates debugging and observability. Commands that can be used on any native ROS node that displays the nodes connected, the topics advertised, their associated message type and the amount of publishers and subscribers. It is also possible to display the bandwidth and frequency used over a topic in real time, furthermore delays can be displayed between two nodes over a topic, however this requires that headers with time stamps are included in the message and that they are set up correctly. It is also possible to observe what protocol is used over all topics. This will be used to verify the requirement R 8 (Data Centric Infrastructure).

5.8.6 ROS Recordings & Visualization

ROS have a built in function called ROSbag which can record and store all the flowing data in the network inside a data lake. This function can be very helpful for debugging purposes as all the system data included in the ROS network can be gathered in one place. Rqt is a software framework that offers various plug-ins to visualize either recorded data or near real-time data as the system is running. This will be used to verify the requirement R 8 (Data Centric Infrastructure).

5.8.7 ROS Node on Simulink & Matlab

In Simulink there is a toolbox called Robotics System toolbox, this toolbox makes it possible to start a ROS master or connect to an existing master on the network. [20] This toolbox only supports TPC communication and is therefore slower than UDP communication. However, for the purpose of this project, TCP is proven to be fast enough. Publishers and subscribers are created on the Simulink model and all the environmental variables are setup in a Matlab script. One way to create custom messages is to downloaded the custom messages from a native ROS node and then compile them using ROS Custom Message Support [21]. The Figure 61 below shows custom messages in Simulink as blocks.



(a) Simulink Subscriber (b) Simulink Publisher

Figure 61: Simulink & ROS.

The subscriber block in Figure 61a defines what topic to listen to which is the arduino_sensor_topic. The structure of the message are preset and must match the publishing structure, the message then connects to a bus selector where it can be distributed to the rest of the model. Figure 61b works about the same way as the publisher, first a Blank Message is set where the message type is defined.

Connecting it to a bus assignment then displays the corresponding message structure, publishing block is then used for the merged message where the publishing topic is set. Simulink usually runs the simulation as fast as it can. As the model is designed to run in real-time, setting the pace of the system is necessary for it to run in real-time.

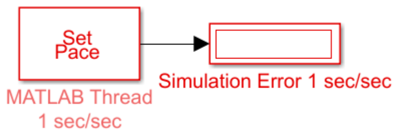


Figure 62: Simulation Pace.

Figure 62 shows the block in Simulink making the model run in real-time. Although this block can only slow the system down if it runs to fast, if the error starts to add up, this means that the system requires more processing power than available.
The Simulink model requires 64 bits float and integers, to satisfy the system, the custom messages around Simulink used 64 bit variables. Adding more variable to the custom message, it increased in size and at one point the message size jammed the publisher. As a 32 bit variable would give the same accuracy as a 64 bit variable, changing the variables to 32bit floats and integers solved the jamming problem. The variables was then converted in Simulink to the size preferred by the model.

Standalone Node On a Linux System

This option requires the Simulink Coder package and Robotics System Toolbox. [19]. Generating c++ code using the Catkin compiler, this makes it possible to run the whole Simulink model on an external ROS device. However in the case of this project, Simscape v1 is not supported with the c++ catkin compiler. Upgrading the Simcape blocks would make it possible to run the whole dynamic model as a standalone node, as a standalone node would require less processing power than a Simulink simulation. Furthermore, Simulink have frequency restrictions unlike a standalone ROS node.

5.8.8 ROS Node on Unity3D using ROSBridge

Unity3D operates in the realm of Csharp which is not directly compatible with ROS. ROSbridge is used to integrate ROS functionality with Unity. ROSbridge provides a JSON (JavaScript Object Notation) API which is used to exchange data with non-ROS programs. ROSbridge is easily installed on a ROS master, using already developed open source APIs [23] for Unity3D to communicate with the ROSbridge. However, Csharp scripts for custom messages, subscribers and publishers needs to be written and compiled for Unity, furthermore how to communicate with the ROSbridge and the ROS network. In this project a TCP websocket is launched that Unity3D can connect to for a full duplex communication.

5.8.9 ROS Node on Arduino using ROSserial

ROSserial is a protocol that makes it possible for an embedded controller for example an Arduino to be connected to the ROS network via a serial cable or a network socket. As the Arduino have both digital and analog pins, these pins can be directly mapped to the ROS infrastructure. The Arduino IDE is used on the RPI to manage the Arduino.

To create custom messages for the Arduino environment, ROSserial have a command for generation of custom messages. Creating the custom messages on a native ROS node, the messages can then be generated using ROSserial for the corresponding system, in this case the Arduino. [5]
Two Arduinos will be used in this project, as one will handle the sensor readings and publish the data to a topic and the second Arduino will handle the H-bridge and subscribe to a topic on the

network to provide the H-bridge with the latest values.

The Arduino baudrate over the serial bus was set to 57600 as standard, as the publishing Arduino exceeded a certain packet size, it easily placed itself out of sync, increasing the baudrate to 115200 made it run smoothly. The reason was that it needed to send two separate packages for one message. As the Arduino run cyclic execution, the frequency of the embedded controller depends on the time it take to execute one cycle. To test the frequency of the subscribing Arduino, one simple publisher was used on the same node. One could argue that the publisher would slow down the Arduino cycle time and produce an inaccurate frequency, but tests showed that a simple publisher or subscriber using a light topic sending and receiving nothing but header files have almost no impact on the cycle time.

One Arduino alone can handle both publish and subscribe, however, using interrupt handler inside the Arduino controlling the H-Bridge puts it out of sync when running high frequencies. As the interrupts seems to take up to much of the processing power on the Arduino, the ROS callback function runs out of time and the Arduino reboots. As the frequency was decreased the Arduino could handle both the publishing and subscribing on one Arduino around 50Hz. As this frequency was to low for the sensor inputs, one more Arduino was installed to handle the interrupt together with the control function of the torque motor.

5.9 Complete System Startup and Project Files

This section explains how to use the motorcycle simulator and were all the necessary files for the project is stored.

5.9.1 System Startup Guide

To start the motorcycle simulator, a PDF-guide is available in appendix A.

5.9.2 Project Files

All project files are stored on our public GitHub Repository called *MC-Simulator-with-VR* [47], where all files are sorted in four main folders. The Arduino folder contains code that are implemented in the two Ardiuno Megas. The Raspberry Pi folder contains ROS files for custom messages, launch files, ROS packages. The Simulink folder contains the Matlab and Simulink files. The Unity folder contains all unity project files as code, assets and plugins. The Raspberry Pi, Simulink and Unity folders also contains documentation PDF:s to assist future modifications.

6 VERIFICATION & VALIDATION

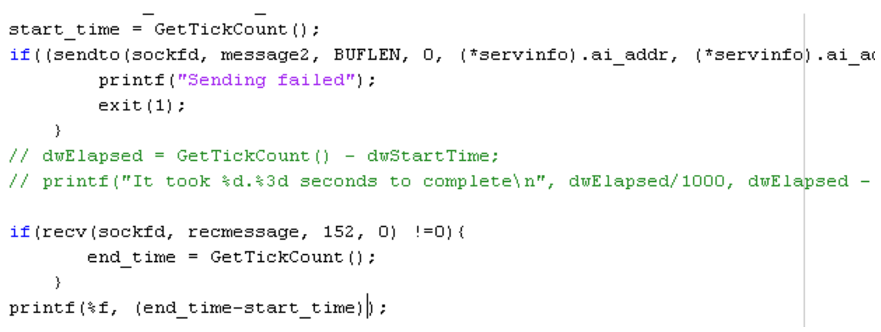
To verify and validate the system against the requirements defined in chapter 3.3, subsystems had to be tested individually as well as the whole system together.

6.1 Motorcycle Construction

To be able to validate that the system complied to the requirement R 1 (Mechanical Structure) the mechanical structure needed to be tested by applying large forces to it to test the stability. Both in weight in form of multiple people and in the way of rotational and translational forces. This was done by applying both large forces over a longer period and also applying fast impulses. By applying larger forces than expected from normal use it can be verified that it can handle normal use. The rig was also under heavy load during normal testing and during the fair.

6.2 Communication with Motion PC

There is no perfect way to determine the execution time of the s function builder. The best way to determine how long the transmission takes is to measure how long it takes to send and receive a message. For this end the a receive part was added and a sort of "ping" was setup. This is done by measuring the time for a transmission and successful reception, using the windows function GetTickCount(); . GetTickCount() is included in windows.h.



```
start_time = GetTickCount();
if((sendto(sockfd, message2, BUFLEN, 0, (*servinfo).ai_addr, (*servinfo).ai_a
        printf("Sending failed");
        exit(1);
    }
// dwElapsed = GetTickCount() - dwStartTime;
// printf("It took %d.%3d seconds to complete\n", dwElapsed/1000, dwElapsed -

if(recv(sockfd, recmessage, 152, 0) !=0){
       end_time = GetTickCount();
    }
printf(%f, (end_time-start_time));
```

Figure 63: Code for investigating run time execution of UDP send & receive.

6.3 Simulation Model

To validate that the simulation model worked as required by the rider, extensive testing was needed. These tests were at first focused on checking that the inputs were working as they should(throttle, front brake, back brake and steering angle). As development continued the test were more and more focused on the stability of the model according to R 13 (System Stability). To test the stability the tests were designed to check the speed interval that the model could handle without crashing as well as testing at what rates braking could be applied at the back and front wheel. This was done since it became clear during test runs that braking at to high rates would get the model outside the safe speed interval. The roll behavior also needed to be tested and limited due to the Stewart Platform and to increase stability.

The simulation model behavior also needed to be verified in terms of R 12 (Cyber-Sickness) since if the motion generated by the platform is not accurate enough they will add to the problem with motion sickness. These tests were done by having external users testing the simulator and give feedback on what felt wrong and what made the feel bad during the test-drive. Both people with experience driving a motorcycle and people without tested the simulator. This made it possible to get feedback about realistic motorcycle behavior from motorcyclists and more feedback about what caused motion sickness from those without motorcycling experience. This was done to validate the user needs of the simulation model as well as for the full simulator, with the goal being that the user could drive the motorcycle in a virtual environment using physical inputs and have a realistic simulation.The work to verify and validate the simulation model naturally went hand in hand with verifying and validating the washout filter.

6.3.1 Powertrain Model

During development the powertrain model was tested with different driving condition. Both the engine model and the full powertrain model were verified in Simulink by comparing their delivered torque at different speeds and then comparing that data to the performance data of the reference motorcycle, the Suzuki GSF 600 [2]. The goal was of course to get as close to the real life performance as possible.

6.4 Motion Cueing

6.4.1 Verification of Washout

As there were no reference signal there is nothing to compare the output of the washout filter to. To verify the washout filter the operator had to logically deduce if the motion responses felt correct. Tests were done to make sure R 11 (Motion Cueing - false cues) and R 10 (Motion Cueing constant accelerations) were met. While they could be verified finding the exact tilt for R 10 (Motion Cueing constant accelerations) could not be found. No way to verify R 11 (Motion Cueing - false cues) besides user feedback was found except reading the signal. Closer analysis needs to be made for human sense of acceleration for a conclusion to be made. To verify the need for the dimensions in R 9 (Motion Cueing degrees) all dimensions except the dimension analysed were used in motion simulation. If the operator did not sense a difference the dimension deemed superfluous.

6.4.2 Validation of Washout Filter

To validate the need to create a washout filter for all dimensions stated in R 9 (Motion Cueing degrees) each dimension was tested. Validation of R 10 (Motion Cueing constant accelerations) and R 11 (Motion Cueing - false cues) was done by researching the subject of motion cueing and tests on the the motion simulator operator. This tests were done to ensure that the user wold not feel motion sickness. If the motions of the user are not in line with what is represented visually or physically from controlling a motor cycle the user experience could result motion sickness.

6.4.3 Validation and Verification of Motion Cueing Functions

The user would often feel hitting a wall with no effect felt and no response when changing ground surface felt jarring. Therefore a crash function and vibration function were added to fulfill the requirement R 12 (Cyber-Sickness).

To verify that the crash and vibration function helped reduce cyber-sickness the user had to test the simulation with and without these functions.

6.5 Driver Interface

As the driver interface is what the driver uses to interact with the rest of the system it was important to validate that it worked in a satisfactory way so the user wouldn't be annoyed or limited by them, as well to verify our requirements. This were done by letting a lot of people test the interface and listen to their feedback. By letting both beginners and experienced motorcycle driver test it the level of realism could be understood and thereby verify requirement R 14 (Driver Interface) and R 15 (Driver Interface Realism).

For the handlebar torque feedback, values in Figure 19 compared to equation 52 verifies that the motor generating the torque on the handlebar is sufficient. Equation equ:turnSpeed verifies that the turnings speed is sufficient. It is not realistic when driving to turn the handlebar from left to right in less than 0.11 seconds. The mechanical design is therefor, fast and strong enough to give torque feedback to the handlebar which satisfies requirement R 16 (Torque Handlebar Feedback).

During validation, external, experienced motorcycle drivers gave feedback that the produced torque is too little in small angles when driving straight forward.

6.6 Virtual Environment

6.6.1 Verification of Unity node

To verify the visual/sound node (Unity), requirements R 3 (Visual Delay), R 4 (Sound Delay), R 12 (Cyber-Sickness) and R 5 (VR Immersion) had to be fulfilled by doing different tests. For R 3 and R 4, the delays can be measured using ROS tools and with a test-users (not from the team) that would actuate random inputs and then see/hear the corresponding output. The test-users would then tell their experienced output from the subsystem and the resulting delays would be verified. To verify R 12 and R 5, test-users would try the complete system for at least 5 minutes and then tell how they felt and what they experienced during and after the simulation. For VR immersion, the users would tell if they where driving the motorcycle differently than from a regular computer game. Those results would then be evaluated for verification.

6.6.2 Validation of Unity Node

The reason why all the verification-tests mentioned in the previous chapter was done, is because without having all those requirements met, the user would cancel the simulation directly and/or would most likely not want to use it again. No one wants to use a system where he/she would feel delays or sickness, as well as if it would not increase the immersion by the VR-equipment. In other words, if all requirements where fulfilled we could validate that the user would experience an immerse MC simulator, which is one of the main needs of the user.

6.7 ROS as a Communication Infrastructure

Tests on the ROS network will be done to verify the connections between nodes separately and then verify the whole infrastructure with integration of all the subsystems. Table 2 shows some usual ROS commands used to verify the system.

Table 2: ROS Commands.

Commands	Explanation
rostopic list -v	List of all the topics including subscribers and publishers
rostopic bw "topic name"	Bandwidth of the topic (B/s)
rostopic hz "topic name"	Frequency of topic (Hz)
rosnode list	Lists all the nodes in network
rostopic echo "topic name"	Listens to the topic and prints the flow of messages

6.7.1 Data Centric Overview

One major perk with using a middleware as an extra layer is centralizing the flow of data, even though there is a peer-to-peer connection between all the nodes, it is easy to display valuable data from any ROS node. Table 2 will be used to verify the requirement R 8 (Data Centric Infrastructure), moreover, that the ROS network is behaving as constructed. Furthermore, these commands will be very useful when debugging the network, facilitating the search of errors in the system.

6.7.2 Communication Connectivity

To verify the requirement R 6 (Communication Connectivity), different protocols and plug-ins will be used. As mentioned earlier, Mathworks provides an easy way to couple Simulink/Matlab together with ROS. ROSSerial will used to verify the Arduino connection, connecting the Arduino with a serial cable to the RPI. Unity3D will be using ROSbridge to connect to the ROS environment, opening a web-socket that will provide a full duplex communication between the entities, using Ethernet as the physical layer. ROSbridge and ROSSerial will need some further installations on the ROS master to establish a connection, the Robotics System Toolbox does not need any modification on the ROS master.

6.7.3 System Modularity

ROS enables a public/subscribe messaging pattern where the nodes can be easily plugged in or plugged which will be used to fulfill the requirement R 7 (System Modularity). To verify the requirement, nodes will be added and removed to an existing functional system, testing the simplicity and modularity.

6.7.4 Communication Delays

To verify the requirement R 2 (Motion Delay). Time stamps will be used and compared between nodes, these tests will be conducted between two nodes peer-to-peer and will be added on the existing system. Moreover, this will help verify the modularity of the system. A wall clock will be used to ensure synchronization of time between different machines. ROS time will be used as the common denominator, where these times will be compared between the nodes.

7 RESULTS

This section is dedicated to the results of the different subsystems and of the system as a whole.

7.1 Motorcycle Construction

The construction proved to be successful as it could hold three persons without failing during runtime so it can hold a heavy person. The construction also proved sturdy when stress tested during normal testing and during a 2 hour demo session and it didn't break or move. The system thereby fulfills requirement R 1 (Mechanical Structure).

7.2 Communication with Motion PC

Using the "ping" does not give exact figure on the transmission/reception time but rather it relies on windows ticks, which is typically between 10-16ms. Running this functionality revealed that a "ping" took no longer than 1 tick. One can therefore say that a transmission takes at most 16ms but one can not say anything more than that.

7.3 Simulation Model

With the final simulation model there were barely any system crashes and the model was stable within the possible speed interval. The crashes that still happened were limited to over-steering, meaning having a very large steering angle, at high speeds and locking the brakes at high speed. This could make sense since you would crash this was done on a real bike. The model was also shown to behave poorly at 0 kilometers per hour, especially when braking and low levels of torque were switched rapidly. Which makes R 13 (System Stability) fulfilled for this subsystem. The model is technically going forward while it is standing still in the VR which makes the behavior not match up completely and the washout filter is based on accelerations, for a constant speed there is none. But this disappears when the motorcycle picks up some more speed.

Feedback from motorcyclists made it very clear that the model does not act realistically while turning. The input in terms of steering seems to be the direction the motorcycle should go in, not the actual angle of the handlebar. This means that the simulation model simulates counter-steering inside of the model but the user does not need to take this into account while driving, if the user turns the handle bar to the right the motorcycle will go to the right and roll to the right. And if the user tries to counter-steer the model will just turn in the wrong direction. This makes User needs, section 3.3.2, not fulfilled since the lack of realism and more work needs to be done to get the right behavior out of the simulation model.

The feedback from non-motorcyclist was mostly that the simulator was fun and they wanted to use it again with only complaints on small amounts of dizziness after the test-run. This shows that R 12 (Cyber-Sickness) is mostly fulfilled.

7.3.1 Powertrain model

The performance of the powertrain model was naturally highly dependent on the performance of the engine model. So this part of the system was the focus during testing. The results is shown in Figure 64 and 65 below.

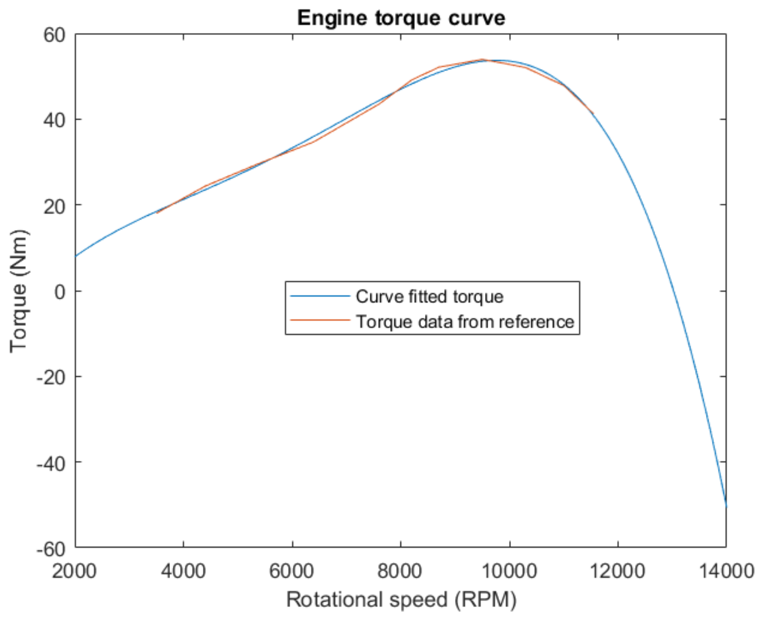


Figure 64: Torque curve for the engine at gear 2 through 6.

The resulting torque curves closely corresponds to the performance data meaning that the engine models performance is very close the the reference motorcycles performance.

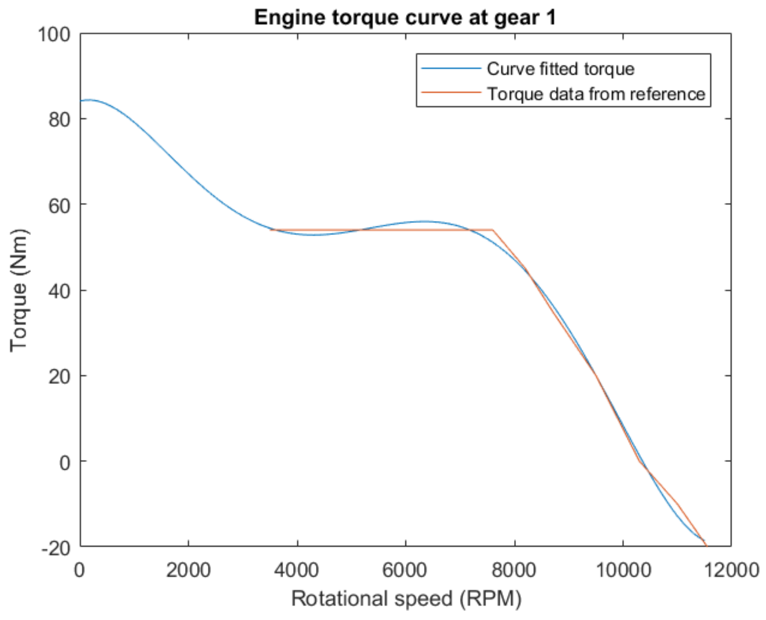


Figure 65: Torque curve for the engine at gear 1.

7.4 Motion Cueing

The response sent to the motion simulator in each direction in each channel was tested. The high frequency responses for step accelerations in in x, y, z are found below.

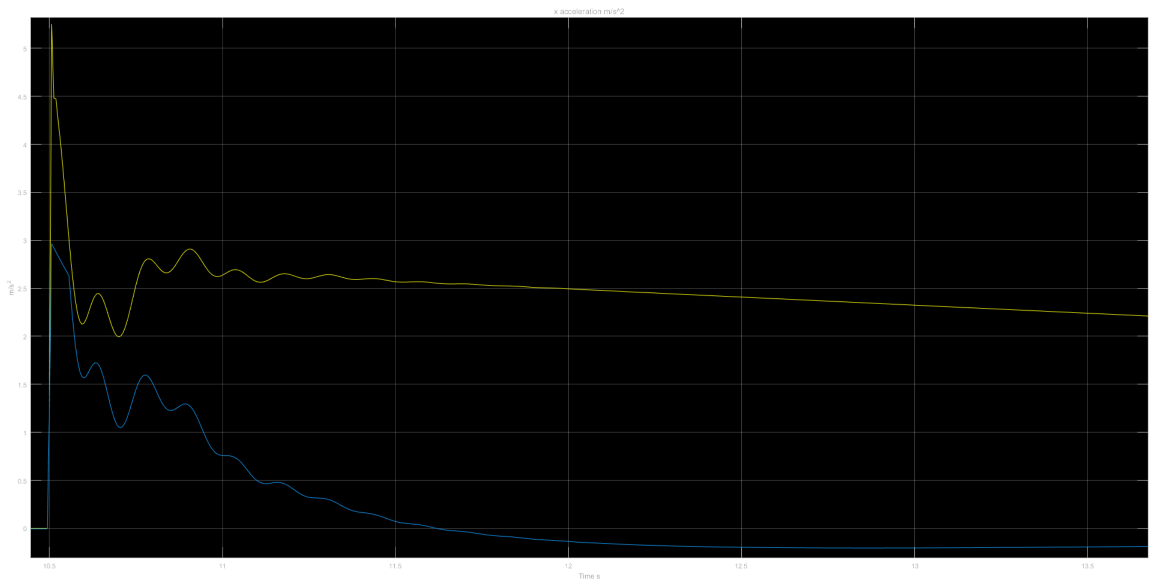


Figure 66: Step response for throttle increase: Scaled motorcycle acceleration (yellow) and washed acceleration (blue) in m/s^2 in x

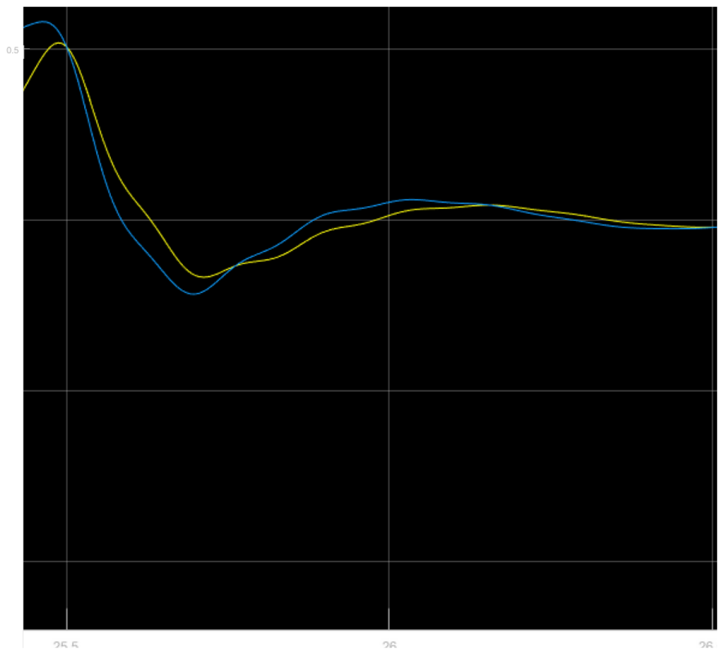


Figure 67: Step response for turning: scaled motorcycle acceleration (yellow) and washed acceleration (blue) in m/s^2 in x dependent on time s.

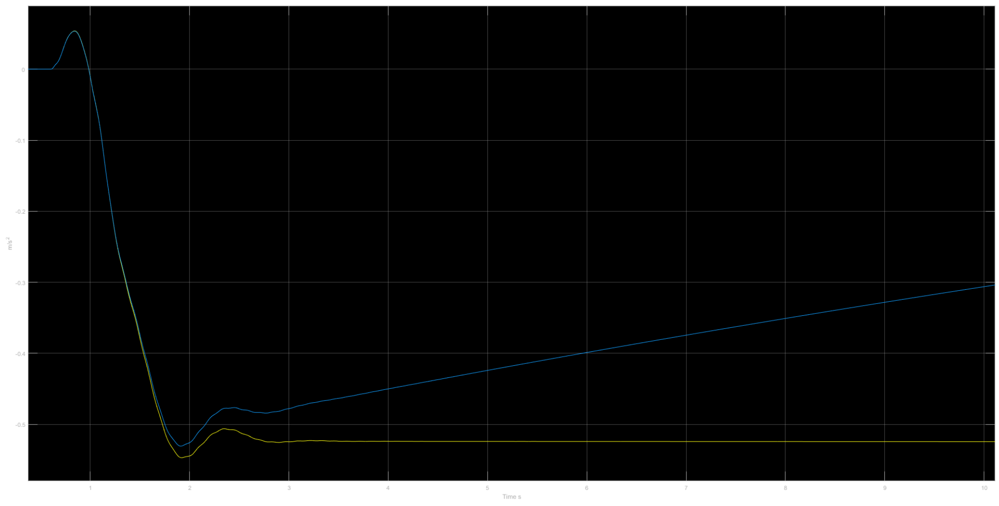


Figure 68: Scaled motorcycle acceleration (0.3[same as filter]) (yellow) and washed acceleration (blue) in m/s^2 in x.

The signals for the high response return to zero at a reasonable speed and follow the behavior of the motorcycle. No false motion cues are found. The requirement R 11 (Motion Cueing - false cues) is met. The angular displacement for pitch is tested with another step signal.

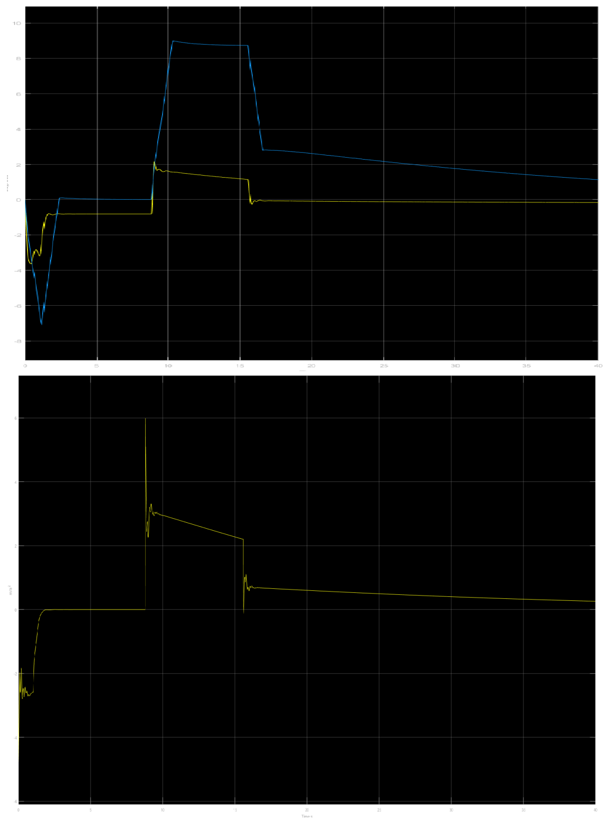


Figure 69: Pitch displacement in degrees for motorcycle (yellow) and motion washout (blue) with accompanying acceleration in x m/s^2.

There are multiple movements happening on the pitch graph. The motor cycle is pitch decreased due to breaking and then kept constant, the pitch acceleration decreases and than returns to zero. Another step is introduced at 9 s when the acceleration is zero. The high-pass filter passes the pitch increase till approximately 10.5. The pitch starts returning to zero but because of a positive sustained acceleration through the co-ordination channel the the motion platform is tilted to simulate a sustained acceleration. In the remainder of the graph the pitch is following the decrease of acceleration in x for both high and low frequencies. Note that the time to find a constant tilt for the second low frequency acceleration is substantially longer than the first. The pitch follow the requirements in R 11 (Motion Cueing - false cues) and R 10 (Motion Cueing constant accelerations).The angular displacement in roll is tested for a sharp turn.

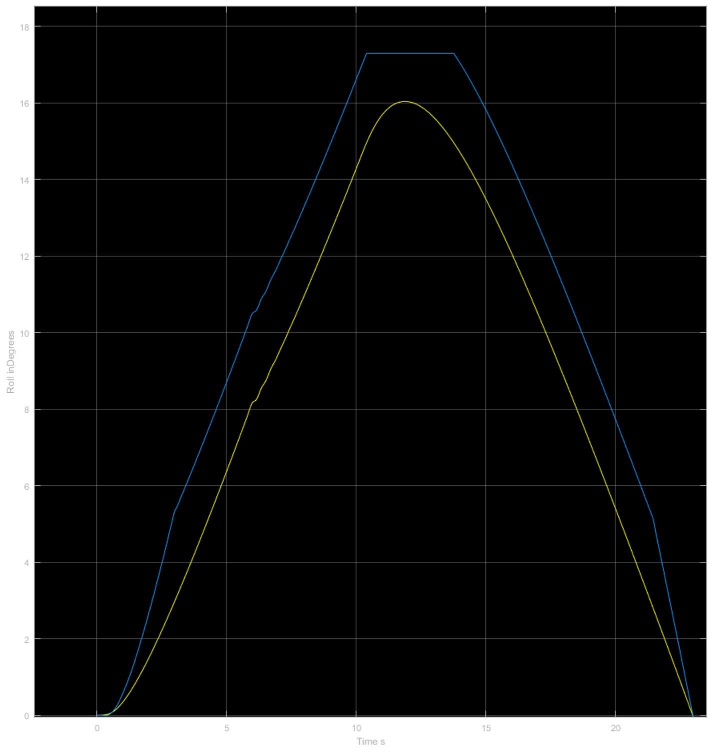


Figure 70: Angular displacement for roll in degrees for right and left turn.

To simulate a large roll the steering angle used is 25 degrees with a speed of 120 km/h. The roll angle was increased was increased due to the constant acceleration in the y direction, verifying R 10 (Motion Cueing constant accelerations). Max value was reached due to the violent turn and the high speed.

The crash function was tested with clear results. The jerk backwards made crashing the motorcycle experience more immersive and reduced feelings of motion sickness.

Signals that seemed of little significance were acceleration in z and the yaw angle displacement. During testing yaw was only felt in extreme turns and the acceleration in z was only noticeable with a considerable scaling up.

7.4.1 Frequency Response

Below are the bode plots for the filters used in the washout filter.

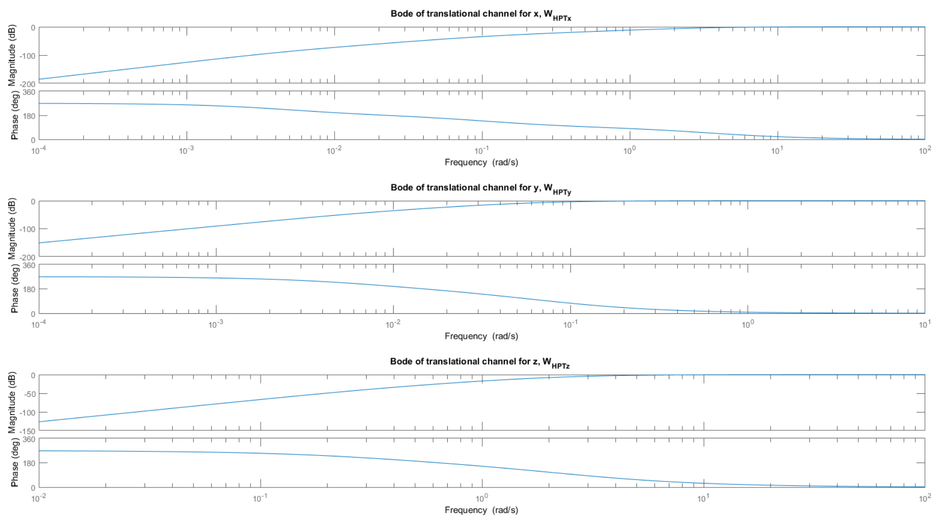


Figure 71: Frequency responses in the translational channel

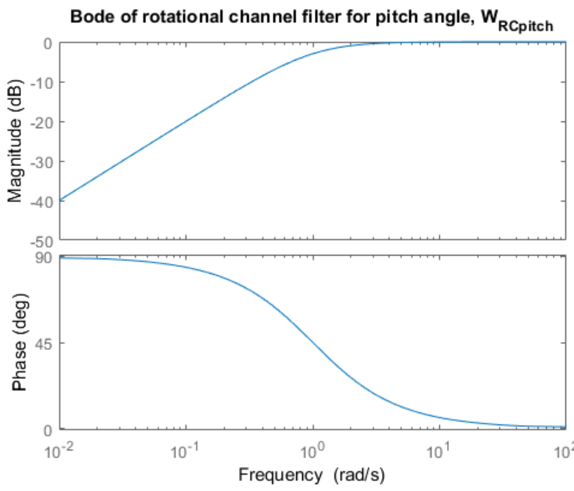


Figure 72: Frequency response in rotational channel

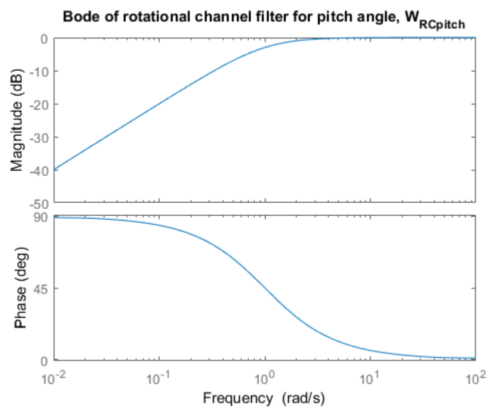


Figure 73: Frequency response in rotational channel

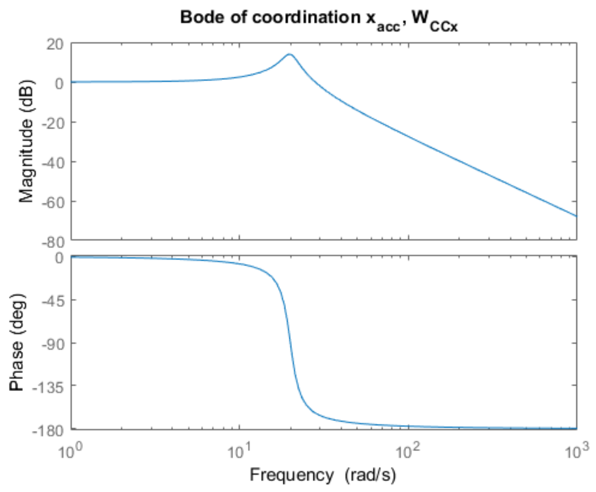


Figure 74: Frequency response in the coordination channel, passed accelerations are later transformed to pitch

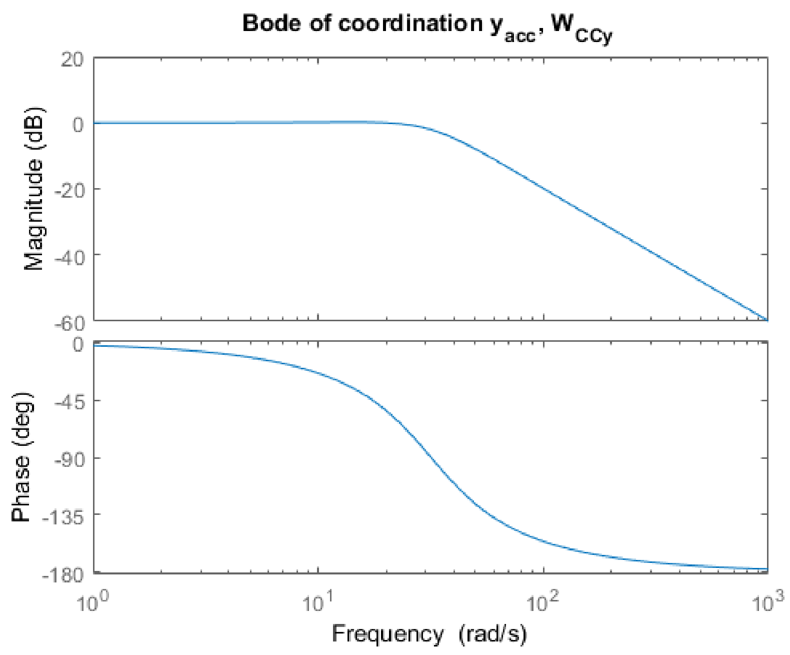


Figure 75: Frequency response in the coordination channel, passed accelerations are later transformed to roll

The motion delay caused by the washout filter was not pinpointed accur

7.5 Driver Interface

The driving experience was satisfactory for new users and they said that they weren't restricted by the driver interface so requirement R 14 (Driver Interface) was fulfilled. Experienced drivers did however feel that the driver interface did lack a few parts. These were the ability to input leaning to the dynamic model, too low stiffness in the front and back brake, and throttle and a steering torque that better corresponded to the feeling of driving a real motorcycle. The system did thereby not fulfill the requirement R 15 (Driver Interface Realism) and left some parts up for improvements in future work.

7.5.1 Handlebar Torque Feedback

The torque feedback system was mainly evaluated using the development model that was used to develop it. Figure 76 shows the resulting step response of the system. The main feedback from users was that the magnitude of the torque was to low which was solved by increasing the magnitude of the reference signal that was sent to the controller.

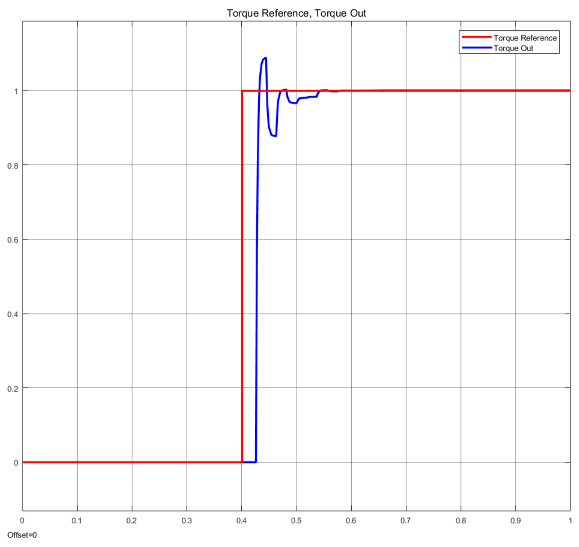


Figure 76: Step response for the simulated controller with the modeled system.

The limiting factor for the controller was the frequency at which it could be run. The low frequency caused the controller to be unstable, which effected the response time of the controller since the gains within the controller couldn't be set to high. There is also an delay in the response since the controller can't sample or actuate any faster. The controller could also be interrupted by the while calculating which will effect the performance and stability of the controller even more.

7.6 Virtual Environment

As table 4 showed in chapter 7.7, requirement R 3 (Visual Delay) and R 4 (Sound Delay) was fulfilled as the measured delay of the Unity3D node was smaller than 20ms. Full system tests carried out by 50 external test-users during development, showed that there where no cyber-sickness introduced to the users to 90% of all cases. In 10% of the cases, the user could feel sick after using the complete simulator, if some other node where failing, which did happen in the early test-stages. Thus, requirement R 12 (Cyber-Sickness) was considered fulfilled. During the same tests, the users would also tell how they felt that the immersion of the VR-environment impacted the way of driving the motorcycle, compared to a regular game. In every case, the result was that they where afraid of driving into obstacles or out of the main road, one example of an user getting scared of crashing is illustrated in Figure 77. One citation of an user, *"It felt like being in another world...!"*. This is definitely a confirmation of a fulfilled requirement, R 5 (VR Immersion).

Figure 77: Scared external user during the demonstration day.

7.7 Communication Infrastructure

Here results will be displayed in what bandwidth, frequency and delays the messages have between different nodes and machines. Furthermore, the protocol in use, the publishing and subscribing nodes, and the topics in use. As all the applications proved to work with one another the requirement R 6 (Communication Connectivity) is fulfilled.

Table 3: ROS Parameters.

Topic	Publishing Node	Subscribing Node	Bandwidth	Size	Protocol	Frequency
Arduino Sensor	Arduino Sensor	Simulink	3.65 KB/s	24 B	TCP	151 Hz
Unity Array	Simulink	Unity3D	7.14 KB/s	48 B	TCP	148 Hz
Torque Control	Simulink	Arduino Torque	215 B/s	4 B	TCP	54 Hz
Game Mode	Unity3D	Simulink	990 B/s	12 B	TCP	83 Hz

The result in table 3 are all extracted from the ROS master, with the commands mentioned in Table 2. Fulfilling the requirement R 8 (Data Centric Infrastructure), verifying a data centric infrastructure. The frequencies depends on the publishing node , in Unity3D the frequency depends on the volatile frame-rate of the game. However when driving as usual the frequency were stable around 80-90 Hz.

Table 4: ROS Delays.

Publishing Node	Subscribing Node	Size	Delay (mean value)
Arduino	Simulink	16 B	22 ms
Unity3D	Simulink	16 B	11 ms
Simulink	Unity3D	16 B	12 ms
Simulink	Arduino	16 B	23 ms

Table 4 shows the delays between different nodes, note that these delays are calculated using header messages with a size of 16 Byte, including the time stamp. The ROS time was used to stamp the published messages, the subscriber then subtracted the stamp time with its internal ROS time, displaying the delays. To include the ROS time with the transmission, a standard message in ROS named Header was used. Test topics were integrated with the functional system and not with the existing topics, resulting in additional topics in the system running simultaneously with the system topics in table 3. The wall clock was also setup on the RPI to ensure no difference in time between the machines. For Unity3D a custom message was designed to include the header. As the delays

cover the transmission for all the nodes in the ROS network, the requirement R 2 (Motion Delay) cannot be fulfilled due to the lack of information about the Simulation delays and the actuation delays on the motion rig. Although the delays can be added to display the communication delays on the system inside the ROS network.
By doing these tests, the requirement R 7 (System Modularity) was verified as the system proved itself to be modular, furthermore nodes crashing are easily isolated and less affecting on the rest of the system.

7.8 Full System Results

As shown in figure 78 from the final demonstration day at KTH, the full system with all connected nodes where able to function and maintain stable for a 2 hours of usage. Some individual nodes required a restart if the user would do a fatal crash with the motorcycle model, but did not occur very often. This fulfills the requirement, R 13 (System Stability). The total cost of the project can be seen in E.

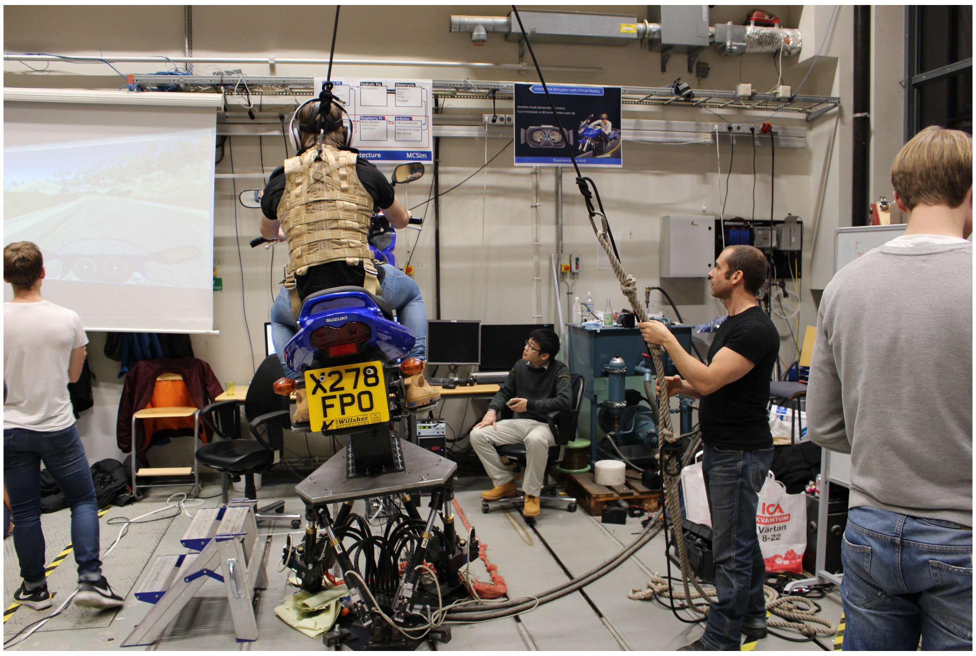


Figure 78: Complete MC Simulator running with VR and safety-harness attached to the user.

8 DISCUSSIONS & CONCLUSIONS

This part of the report is dedicated to discussions and conclusions drawn during the project.

8.1 Motorcycle Model

The work to stabilize the model was somewhat successful. The added stability due to the use of a PID-controller applying a stabilizing force to limit roll should be considered a success. Having the motorcycle model behave slightly different from reality while greatly stabilizing the model and the limiting factor being what the hydraulic rig is able to achieve is an acceptable trade off.
The failure to incorporate the TMeasy model into the model and the consequential hotfix of shifting the speed should not be taken lightly. The shift of the velocity resulted in a motorcycle simulator that *worked* at low speeds but the performance improved with increasing velocity. This can be compared to the unaltered model which was *unusable* for the purpose of the simulator at lower speeds. Had the switch between tire models happened this issue could have been solved, meaning that the possible positive outcome of this venture made it worthwhile (even in retrospect). Or perhaps it is naive to think that TMeasy can be utilized without the calculation of the contact point.

8.2 Counter Steering

To be able to make a turn with a motorcycle it needs to be leaning into the curve. So to be able to make a right turn, the driver first needs to turn to the left to make the motorcycle lean to the right. Then after it is leaning the driver can turn to the right to take the curve while leaning into it. This behavior is called counter steering and is something that all motorcyclist needs to learn, some are even unaware that they do it but still do it naturally when they drive. The dynamic model that the whole project is built on had this behavior implemented into it. As it was created to be able to handle steering input so that it is given the amount of turning left or right it will handle the counter-steering by itself. To be able to have the driver preform the counter-steering this behavior should have been bypassed. This proved much harder than expected and wasn't done due to lack of time. This makes the system lack a degree of accuracy so it could not be used to train drivers to counter-steer. It however creates other feelings that an untrained driver could enjoy without the risk of crashing a real motorcycle and get hurt.

8.3 Motion Cueing

The motion cueing was successful. It generated enough accelerations and rotations to simulate driving a motorcycle with the use of a small space. No motion sickness was induced due to the washout filter and the crash function and users felt as if they were riding an actual motorcycle. Tuning each of the channels and direction by themselves helped opened up a lot of design space for the washout filter that helped making it work. The tuning of the washout filter could however been done better. The tuning process was costly in time with many the iterative steps and reliant on the users understanding of riding a motorcycle. This led to a simulating to what the user tuning the washout filter thought riding a motorcycle felt. While not wrong the motions simulated are not completely correct either. The results compared to mathematically optimized classical are far from ideal. The time response are faster while still retaining smooth returns to the neutral position. In summary the rough behaviour of a motorcycle is present but the finer dynamics are not which does make the simulator lacking for motorcycle development in its current form but has potential if further refined. The delay caused by the washout filter should be looked in to. The phase lag and it's effects on multiple dimension was too complex to analyse for the full system but was deemed to be negligible.

8.4 Virtual Reality

By having a Virtual Reality headset, there was no doubt that the user would feel a complete different driving experience than just using a regular big screen in front of the user. One consequence of having VR is that there is a risk of cyber-sickness either if something goes wrong in the system or if it is the first time for a user to experience VR. But this problem is not related to our project as it can occur in every kind of VR application.

8.5 Communication Design

As ROS cannot guarantee real-time and lack performance as a service (QoS). ROS was still the winning choice of middleware due to its large libraries and communities. The main area that set the requirements of the middleware was the API:s to the applications in use. A way to connect Unity3D, Simulink and Arduino to a common infrastructure. ROS proved to satisfying the requirement of system modularity, communication connectivity and data centric infrastructure.

To calculate the delays on the network between the nodes, extra topics were integrated with the rest of the system. These topics was only activated when calculating delays. This could be slowing down the system and increasing the delays. Moreover the time of the delays could differ depending on the placement of the calculations in the code. A few parameters to consider is delay between the creation of the timestamp and the publishing command, the delay between the subscription, the subtraction of the local timestamp and received timestamp. One possible cause of the increased Arduino delays compared to the rest of the system could be the structure of the code used in the Arduino.
The end-to-end delay regarding the motion in the system could not be extracted, however, testing the system, one could not detect any kind of delay from input to motion. Furthermore no complaints were made about delays when tests were conducted by external users.

9 FUTURE WORK

This part of the report focus on potential future work the project team feels would be advantageous to look into.

9.1 Safety Aspects

Including the Motion PC withing the ROS network would introduce a safer environment as it still can be controlled if the Simulink model crashes.

9.2 Mechanical Design

To be able to get great immersion other sources of inputs needs to be added. This includes, but limited to, wind and vibrations. Implementation of large fans to simulate the wind passing by the driver at high speeds is something that really would increase the immersion. It was found in the pre-study that these kinds of extra inputs to the user really elevates the simulation. Unfortunately due to time limitations it could not be implemented during this project but is something that is believed to be essential to do if someone would choose to continue to work on this project.
The vibrations from the motor is something that the hydraulic rig was not able to recreate due to too high frequencies. This should instead be implemented with some sort of vibration device that could either be mounted on the gas tank or the frame depending on what gives the mot realistic feeling of motor and road vibrations. The rig is able to simulate road rumble but not high frequency vibrations from gravel.
The clutch and gear shifting was not implemented but should be to enable full control of the motorcycle as this is crucial for full immersion. The throttle and brakes tension should be changed to be harder as some experienced drivers mentioned that they felt too lose, also any kinks in the throttle movement should be removed as it can sometimes be uneven as the springs get stuck on edges on the frame.

9.3 Simulation Model

The possibilities to improve the motorcycle model are plentiful. The completion of the switch between Pacejkas magic formula to TMeasy is a natural place to start. This would decrease the necessity to shift the speeds of the motorcycle and generally improve model stability.
It should also be possible to exchange the motorcycle model into only a Simulink model. This would require some VERY good understanding of vehicle dynamics. Another approach would be to fully build the motorcycle model in unity. This would ultimately decrease system latency even further since it would effectively eliminate the need to communicate with Simulink.

Another important part to improve in the dynamic model is to implement counter-steering as the model used removed this behavior that is crucial for full immersion for experienced drivers. This is also something that could be solved by completely replacing the dynamic model used in the project.

9.4 Motion Cueing

There are two great improvements that can be made to this project. One is adding the model for the simulator rig and the other is integrating the vestibular system to the washout filter[26]. For the integration of the vestibular system to be fruitful precise head tracking relative to the motion platform frame is needed.

The yaw motions cannot in the projects current state be used as yaw is used to position the motorcycle in the virtual environment and slips are therefore not felt. To improve the scalars non-linear scaling could be used. This would ensure a large gain on big movements while not increasing the gain on smaller movements.

The classical washout filter is functional but lacking. The problem lies within the classical washout filter being tuned for the worst-case maneuverer for the motion simulator to not go out of bounds and the need for human experience to parameterize. These points are addressed by adding the vestibular model and a model for the motion simulator. Optimizing the washout filter with an optimal washout filter will then yield better results but taking the step further and using an adaptive washout filter will be preferable as its bound to perform well on relevant frequencies and not all frequencies.

Another issue with washout filters in general is the trade between space for motion and motion. To have realistic jerks or slower motions there is a need for larger space, motions are therefore often tuned down. Ways to increase the use of the given space could be made by moving the operator to a more desirable position without the operator sensing the motion to then have more space for a larger motion to be felt. This can be done by using a fuzzy control based algorithm [39].

9.5 Electrical Design

To read the steering with a potentiometer was sufficient in the beginning of the project to be able to send the steering signal through the system to test it. The intention was however to eventually change it to a good encoder to get higher accuracy that is needed while driving in high speeds as then the amount of steering to alter the trajectory is very small. So to be able to reach the next level of accuracy in the control an encoder would need to be implemented. To be able to read the encoder a separate unit should be used as the Arduino was put under heavy load when using interrupts. Using a separate unit would allow higher pulse-per-rotation count and give the position, then use an Arduino or preferably something stronger to do the calculations for the controller. A current sensor should be used to be able to know the torque that the motor is applying to the handlebar and at what position it is.
The implemented solution for stopping the system was a good idea that worked fairly well but to increase safety, kill switches should be implemented that works even if Simulink or ROD crashes as the current solution does not.

9.6 Virtual Environment

For improved visual immersion, more tracking sensors could have been implemented for tracking and visualizing the user's arms, hands and fingers in a more realistic way. In Unity3D the possibility to animate bones with inverse kinematics for different actions could have been used also. The virtual environment could also have improved by having different kind of motorcycle models.

9.7 Communication Environment

To improve the communication system there are some changes to do. Include the Motion PC in the ROS network, as the computer uses Linux Fedora it would be an easy implementation.

However, the software installed on the motion PC is provided by the company Bosch and therefore should be contacted before any adjustments. Including the motion PC with the rest of the network would make it easier to measure delays and would not make the whole system so dependent on the Simulink model, as the Simulink model is the only node in direct connection with the Motion PC. Change to Simscape V2 in the Simulink model would make it possible to compile the whole Simulink model to a standalone ROS node. This would make the node lighter, more stable and possibly faster. One major change to the system would be to upgrade to a DDS system, as this would introduce real-time characteristics to the communication, however, that would mean to change the whole architecture of the system. As DDS still uses a publish/subscribe pattern, ROS version 1 does not support it. For debugging purposes, the ROS master hardware can be upgraded, as the master records and visualizes the flow data of the ROS network, it can slow down the whole system. Alternately having a more stable ROS node using Linux in the system. Integrate header files in all the custom messages, this is to have a continuous measurement of the delays in the whole system, although this could introduce unnecessary bandwidth to the system. Examine the options for the ROSbridge websocket that is used to connect the game engine Unity3D, possible to change the websocket from a TCP to a UDP communication to increase speed.

9.8 User Leaning

One of the most asked question during demos where the possibility to lean on the motorcycle while turning. This input where successfully implemented using the VR - tracking system as described in chapter 5.6.7, however due to time constraints we didn't have time to implement those forces on the motorcycle.

9.9 Motion Platform

The motion simulator could be improved with an electrical model to attain a higher stiffness for more precise motions. To improve the motion cueing architecture by changing the control algorithm for the motion platform itself and modeling it more access to documentation is needed. Using the position of upper gimbals, lower gimbals and the actuator maximum length a relatively simple model could be made for the motion platforms position and actuator position [26].

Other missing functions includes the position of the actuators from the motion simulators sensors. The information from these sensor could be used to create a better washout filter with real time feedback, verify the results of testing and to develop digital filters for the signals being sent after motion cueing to the motion platform [26].

The whole startup process of the system could have been made simpler with some automated launch process. This was not very easy to implement as the system is very modular and has different nodes on different machines.

References

[1] D. Allerton. Principles of flight simulation. Aerospace Series., 2009. [Hoboken: Wiley. p434].

[2] Motorcycle Performance Analyzer. Suzuki GSF 600 Bandit Performance Data. `http://motorcycleperformanceanalyzer.com/suzuki/gsf-600-bandit-2000/`, 2013. [Online; accessed 21-Dec-2017].

[3] V. Cossalter, R. Lot, M. Massaro, and M. Peretto. Investigation of motorcycle steering torque components. *AIP Conference Proceedings*, 2011.

[4] F. Barbagli C. A. Avizzano G. Di Pietro A. Brogni M. Vignoni M. Bergamasco D. Ferrazzin, F. Salsedo. The moris motorcycle simulator: An overview, 2001.

[5] Open Source Robotics Foundation. Ros wiki. `http://wiki.ros.org/rosserial/Tutorials/Adding%20Other%20Messages`.

[6] D Straumann G Bertolini. Moving in a moving world: A review on vestibular motion sickness. *Frontiers in Neurology*, 2016.

[7] Dynamic Games. Speedway Oval Race Track. `https://www.assetstore.unity3d.com/en/#!/content/34282`, 2015. [Online; accessed 13-Dec-2017].

[8] Lamri Nehaoua Hichem Arioui. Motion washout filter tuning: Rules and requirements. 2010.

[9] Chin-I Huang. Adaptive washout filter design with human visual-vestibular based (hvvb) for vr-based motion simulator. *Neural Networks (IJCNN), The 2010 International Joint Conference on*, 2010.

[10] Houck Jacob A. Cardullo Frank M. Telban Robert J. Motion cueing algorithm development: Human-centered linear and nonlinear approaches. 2005.

[11] Hans Johansson. *Elektroteknik*. KTH, 2013.

[12] B.J. Kemp. Reaction time of young and elderly subjects in relation to perceptual deprivation and signal on versus signal off conditions. *Developmental Psychology*, 8(2):268–272, 1973. cited By 12.

[13] Kitguru. Virtual reality motion sickness more of a problem for women. `https://www.kitguru.net/components/vr/jon-martindale/virtual-reality-motion-sickness-more-of-a-problem-for-women/`, 2015. [Online; accessed 17-Dec-2017].

[14] Kristen L Casto and John Casali. Effects of headset, flight workload, hearing ability, and communications message quality on pilot performance. 55:486–98, 06 2013.

[15] Pontus Larsson, Daniel Vastfjall, and Mendel Kleiner. Better presence and performance in virtual environments by improved binaural sound rendering. In *Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio*, Jun 2002.

[16] Qusay Mahmoud. *Middleware for communications*. John Wiley & Sons, 2005.

[17] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. In *Proceedings of the 13th International Conference on Embedded Software*, page 5. ACM, 2016.

[18] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Wip abstract: Preliminary evaluation of ros2. In *Cyber-Physical Systems (ICCPS), 2016 ACM/IEEE 7th International Conference on*, pages 1–1. IEEE, 2016.

[19] MathWorks. Generate ros standalone node. `https://se.mathworks.com/help/robotics/examples/generate-a-standalone-ros-node-from-simulink.html`.

[20] MathWorks. Robotics system toolbox. `https://se.mathworks.com/products/robotics.html`.

[21] MathWorks. Ros custom message support. https://se.mathworks.com/help/robotics/ug/ros-custom-message-support.html.

[22] MAXON. Cinema 4D application. https://www.maxon.net/en/products/cinema-4d/overview/, 2017. [Online; accessed 17-Dec-2017].

[23] Michaeljenkin. michaeljenkin/unityros. https://github.com/michaeljenkin/unityros, Jun 2015.

[24] Microsoft. UDP example. https://msdn.microsoft.com/de-de/library/bb979228.aspx/#ID0E3BAC, 2005. [Online; accessed 17-Dec-2017].

[25] N. Murgovski. Vehicle modelling and washout filter tuning for the chalmers vehicle simulator. Lund University, Iea, 2007. [Lund University, IEA].

[26] Nikolce Murgovski. Vehicle modelling and washout filter tuning for the chalmers vehicle simulator. 2007.

[27] NIANDREI. Lake Race Track. https://www.assetstore.unity3d.com/en/#!/content/55908, 2015. [Online; accessed 13-Dec-2017].

[28] LloydD.Reid PeterR.Grant. Driving simulation. 1997.

[29] Thiago Malheiros Porcino, Esteban Walter Gonzalez Clua, Cristina Nader Vasconcelos, Daniela Trevisan, and Luís Valente. Minimizing cyber sickness in head mounted display systems: design guidelines and applications. *CoRR*, abs/1611.06292, 2016.

[30] M.A Reid, L.D. · Nahon. A flight simulation motion-base drive algorithms: part1. 1985.

[31] Ron Reisman. A brief introduction to the art of flight simulation. *G. Hattinger et al.[6] pp*, pages 159–169, 1990.

[32] Bernhard E Riecke and Jörg Schulte-Pelkum. Perceptual and cognitive factors for self-motion simulation in virtual environments: how can self-motion illusions ("vection") be utilized? In *Human walking in virtual environments*, pages 27–54. Springer, 2013.

[33] Robert Riener and Matthias Harders. *Virtual reality in medicine*. Springer Science & Business Media, 2012.

[34] Prof. Dr. Georg Rill. *VEHICLE DYNAMICS*. ISTE Ltd., 2007.

[35] D. Limebeer R.Sharp. *A Motorcycle Model for Stability and Control Analysis*. Kluwer Academic Publishers, 2001.

[36] Erik Sikström, Amalia de Götzen, and Stefania Serafin. The role of sound in the sensation of ownership of a pair of virtual wings in immersive vr. In *Proceedings of the 9th Audio Mostly: A Conference on Interaction With Sound*, AM '14, pages 24:1–24:6, New York, NY, USA, 2014. ACM.

[37] Frank Steinicke, Yon Visell, Jennifer Campos, and Anatole Lécuyer. *Human Walking in Virtual Environments: Perception, Technology, Applications*. 03 2013.

[38] Li-Chen Fu Sung-Hua Chen. An optimal washout filter design for a motion platform with senseless and angular scaling maneuvers. *Proceedings of the 2010 American Control Conference*, 2010.

[39] Li-ChenFu Sung-HuaChen. An optimal washout filter design with fuzzy compensation for a motion platform. *18th IFAC World Congress*, 2011.

[40] Unity Technologies. Unity - Interaction in VR. https://unity3d.com/learn/tutorials/topics/virtual-reality/interaction-vr, 2017. [Online; accessed 17-Dec-2017].

[41] Unity Technologies. Unity - Sound Effects and Scripting. https://unity3d.com/learn/tutorials/topics/audio/sound-effects-scripting, 2017. [Online; accessed 17-Dec-2017].

[42] Unity Technologies. Unity - Sphere Collider Class. https://docs.unity3d.com/Manual/class-SphereCollider.html, 2017. [Online; accessed 17-Dec-2017].

[43] Unity Technologies. Unity - User Interfaces in VR. https://unity3d.com/learn/tutorials/topics/virtual-reality/user-interfaces-vr, 2017. [Online; accessed 17-Dec-2017].

[44] Unity Technologies. Unity Manual - VR. https://developer.oculus.com/documentation/unity/latest/concepts/book-unity-gsg/, 2017. [Online; accessed 17-Dec-2017].

[45] Unity Technologies. Unity user Manual. https://docs.unity3d.com/Manual/index.html, 2017. [Online; accessed 17-Dec-2017].

[46] Telegraph. Cybersickness: The new 'illness' sweeping the nation. http://www.telegraph.co.uk/news/health/news/12001743/Cybersickness-The-new-illness-sweeping-the-nation.html, 2015. [Online; accessed 17-Dec-2017].

[47] Rickard Svensson Marcus Olsson Gustav Sten Karl Hansen Jiacheng Cai Said Saleh. Thibault Helle, Robert Yousif. GitHub - MC Simulator Project Files and Code. https://github.com/GitTibbe/Mc-Simulator-with-VR, 2017. [Online; accessed 21-Dec-2017].

[48] Nehaoua L. Trioui, H. Driving simulation (focus series). John Wiley & Sons Content Provider., 2013. [Hoboken: Wiley. p87-92].

[49] Nehaoua L. Trioui, H. Driving simulation (focus series). John Wiley & Sons Content Provider., 2013. [Hoboken: Wiley. p90].

[50] Nehaoua L. Trioui, H. Driving simulation (focus series). John Wiley & Sons Content Provider., 2013. [Hoboken: Wiley. p85].

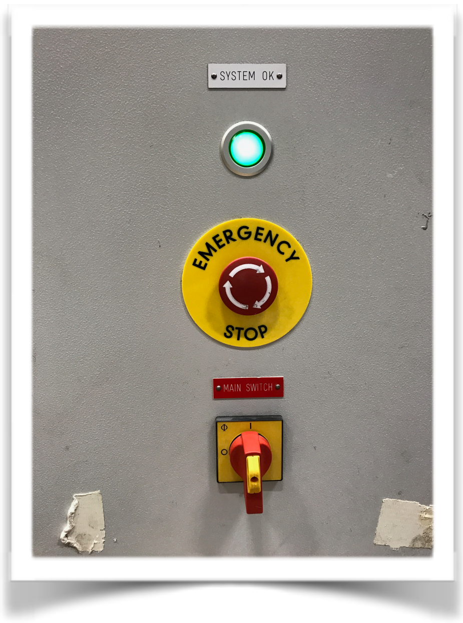
[51] ufukufuk. Ducati 916 3D-model. https://www.cgtrader.com/free-3d-models/vehicle/motorcycle/ducati-916-9c44c4ae-3397-450e-b496-657a88f96bc3, 2016. [Online; accessed 13-Dec-2017].

[52] Vection VR. Car driving school simulator with VR and motion cancellation fix. http://www.vectionvr.com/en/index.html, 2015. [Online; accessed 17-Dec-2017].

Appendix A Motorcycle Simulator - Full System Startup Guide

System startup guide

1. **Turn on** the motion platform rig by turning on the red **switch** so that the green light turns on.

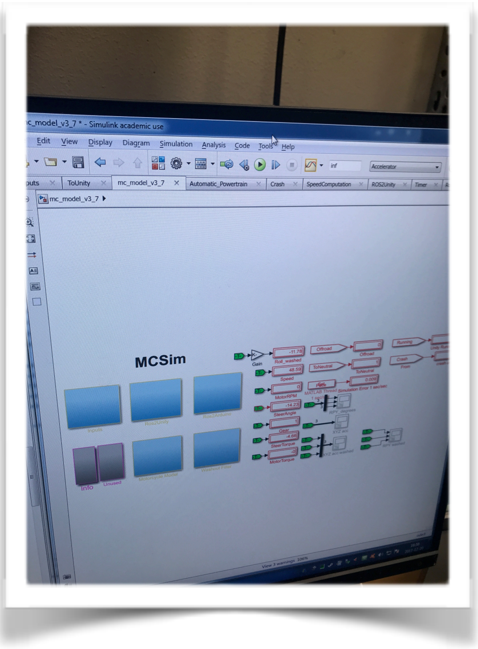


2. On the motion PC, make sure to set the Mode Of operation of HyPCoS UUclient to "**Normal mode**" and to **acknowledge** all errors in the Error Handler window.

3. Log in to the **Raspberry Pi** with password *New4you.* Start **ROS** communication by typing in these two commands in the bash:

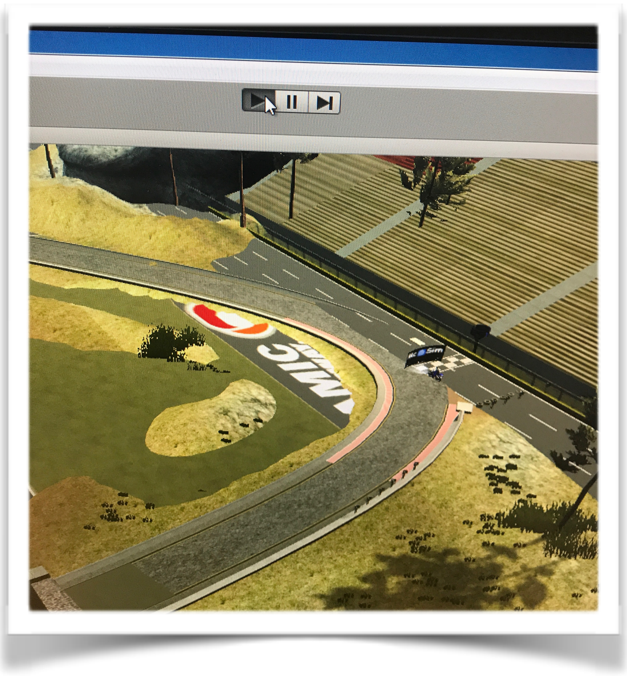
```
> roslaunch master system.launch
> rosrun rosserial_python serial_node.py _port:=/dev/ttyACM1
_baud:=115200 __name:=Arduino_SUB
```

4. **Open** the Matlab script **runModel.m** and run the code.
5. **Open** the Simulink model **mc_model_v3_7**.



6. **Run** the model by pressing the green “play button”.
7. **Open** the **Unity3D** application and open the project folder “*MCsim”, (*scene: *race_track_lane*). Make sure that the VR - headset is connected though USB and HDMI.

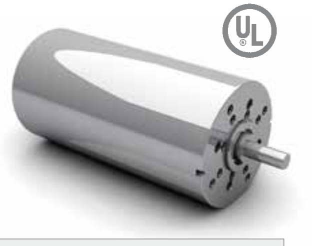
8. **Run** the simulator-game by clicking the “play button”.



9. Now that ROS, Simulink, Unity and the rig is running, the user should be able to start the Simulator with the VR Headset by looking at the button “**Start Simulator**” and turn on the Rig motion by switching on the **red switch** on the **right** side of the handlebar.

Appendix B DC Motor Datasheet

GR 63x55 | cont. 99 W, peak 286 W



» *Operation in both directions of rotation*
» *Ball bearing at motor output shaft is standard*
» *Optionally with deviant shaft length and diameter, shaft on both sides, special and high voltage winding, higher protection class up to IP 67*

» Drehrichtung Rechts-/ Linkslauf
» Motorwelle abtriebsseitig kugelgelagert ist Standard
» Optional abweichende Wellenlängen und -durchmesser, beidseitige Welle, Sonder- und Hochspannungswicklungen, höhere Schutzart bis IP 67

Data/ Technische Daten		GR 63x55			
Nominal voltage/ Nennspannung	VDC	12	24	40	60
Nominal current/ Nennstrom	A*)	8.7	4.9	3	2
Nominal torque/ Nennmoment	Ncm*)	24	27	27	28.3
Nominal speed/ Nenndrehzahl	rpm*)	3000	3350	3450	3350
Friction torque/ Reibungsmoment	Ncm*)	2	2	2	2
Stall torque/ Anhaltemoment	Ncm**)	190	257	301	200
No load speed/ Leerlaufdrehzahl	rpm*)	3500	3650	3600	3600
Nominal output power/ Dauerabgabeleistung	W*)	75.4	94.7	97.5	99.3
Maximum output power/ Maximale Abgabeleistung	W	174	245	282.7	285.6
Torque constant/ Drehmomentkonstante	Ncm A-1***)	3.2	6.4	10.5	15.4
Terminal Resistance/ Anschlußwiderstand	Ω	0.19	0.6	1.4	3.05
Terminal inductance/ Anschlußinduktivität	mH	0.5	1.5	3.5	7.6
Starting current/ Anlaufstrom	A**)	60	40	28.6	19.7
No load current/ Leerlaufstrom	A**)	0.8	0.4	0.28	0.2
Demagnetisation current/ Entmagnetisierungsstrom	A**)	66	33	20	13
Rotor inertia/ Rotor Trägheitsmoment	gcm²	750	750	750	750
Weight of motor/ Motorgewicht	kg	1.7	1.7	1.7	1.7

*) $\Delta\vartheta_w$ = 100 K; **) ϑ_R = 20°C ***) *at nominal point*/ im Nennpunkt

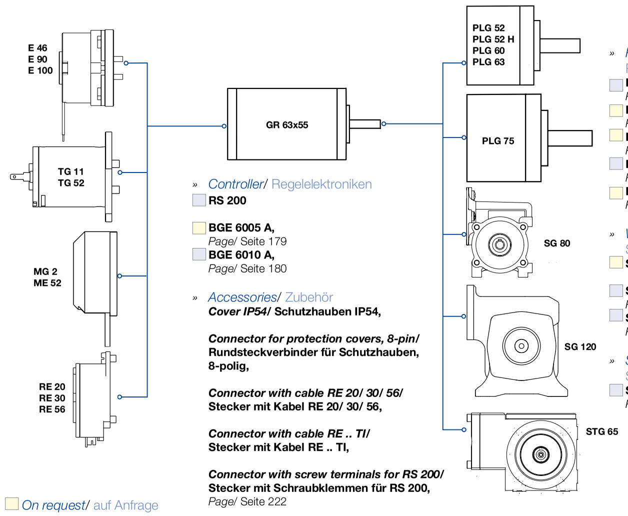
Modular System/ Modulares Baukastensystem

» *Brakes & Encoder*/ Bremsen & Anbauten
☐ **E 46,** *Page*/ Seite 214
☐ **E 90,** *Page*/ Seite 214
☐ **E 100,** *Page*/ Seite 214

☐ **TG 11,** *Page*/ Seite 219
☐ **TG 52,** *Page*/ Seite 219

☐ **ME 52,** *Page*/ Seite 220

☐ **RE 20,** *Page*/ Seite 216
☐ **RE 30 (TI),** *Page*/ Seite 216
☐ **RE 56 (TI),** *Page*/ Seite 216



» *Controller*/ Regelelektroniken
☐ **RS 200**

☐ **BGE 6005 A,** *Page*/ Seite 179
☐ **BGE 6010 A,** *Page*/ Seite 180

» *Accessories*/ Zubehör
***Cover IP54*/ Schutzhauben IP54,**

***Connector for protection covers, 8-pin*/ Rundsteckverbinder für Schutzhauben, 8-polig,**

***Connector with cable RE 20/ 30/ 56*/ Stecker mit Kabel RE 20/ 30/ 56,**

***Connector with cable RE .. TI*/ Stecker mit Kabel RE .. TI,**

***Connector with screw terminals for RS 200*/ Stecker mit Schraubklemmen für RS 200,** *Page*/ Seite 222

» *Planetary gearbox*/ Planetengetriebe
☐ **PLG 52,** *Page*/ Seite 194
☐ **PLG 52 H,** *Page*/ Seite 195
☐ **PLG 60,** *Page*/ Seite 196
☐ **PLG 63,** *Page*/ Seite 196
☐ **PLG 75,** *Page*/ Seite 200

» *Worm gearbox*/ Schneckengetriebe
☐ **SG 65**

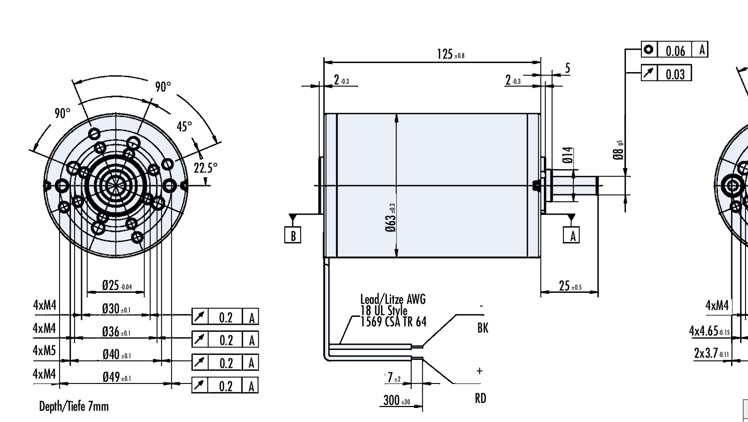
☐ **SG 80,** *Page*/ Seite 209
☐ **SG 120,** *Page*/ Seite 210

» *Spirotec gearbox*/ Spirotec Getriebe
☐ **STG 65,** *Page*/ Seite 206

☐ *Preference*/ Vorzugsreihe ☐ *On request*/ auf Anfrage

GR 63x55 | cont. 99 W, peak 286 W

Dimensions in mm/ Maßzeichnung in mm

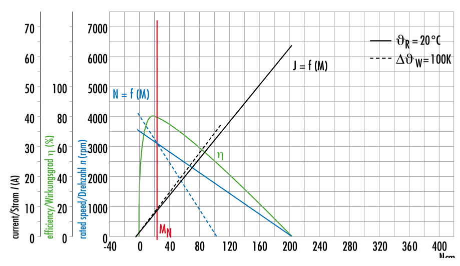


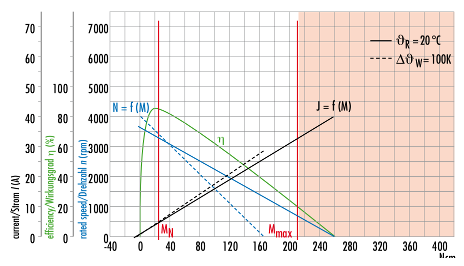
Shaft/ Welle	
front/ vorne	*back*/ hinten
8 x 25 mm	-
5 x 20 mm	-
8 x 55 mm	-
8 x 55 mm	8 x 55 mm
5 x 11 mm	-

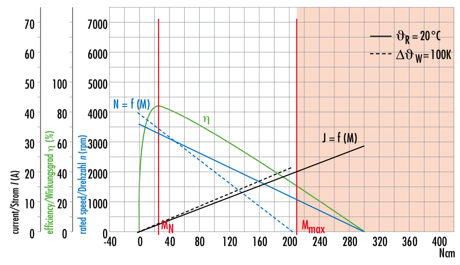
F_{axial} = max. 150N
F_{radial} = max. 150N

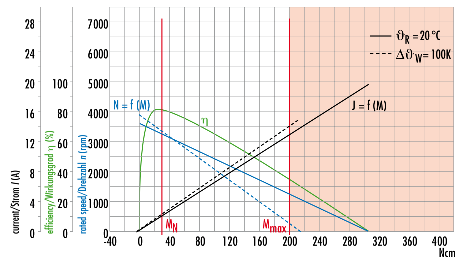
Characteristic diagram/ Belastungskennlinien

In accordance with/ Belastungskennlinien gezeichnet nach EN 60034


GR 63x55, 12V


GR 63x55, 24V


GR 63x55, 40V


GR 63x55, 60V

Appendix C Washout Filter and Crash Function

The design for the washout filter its time constants for the washout filter, scalers and dampening can be found below.

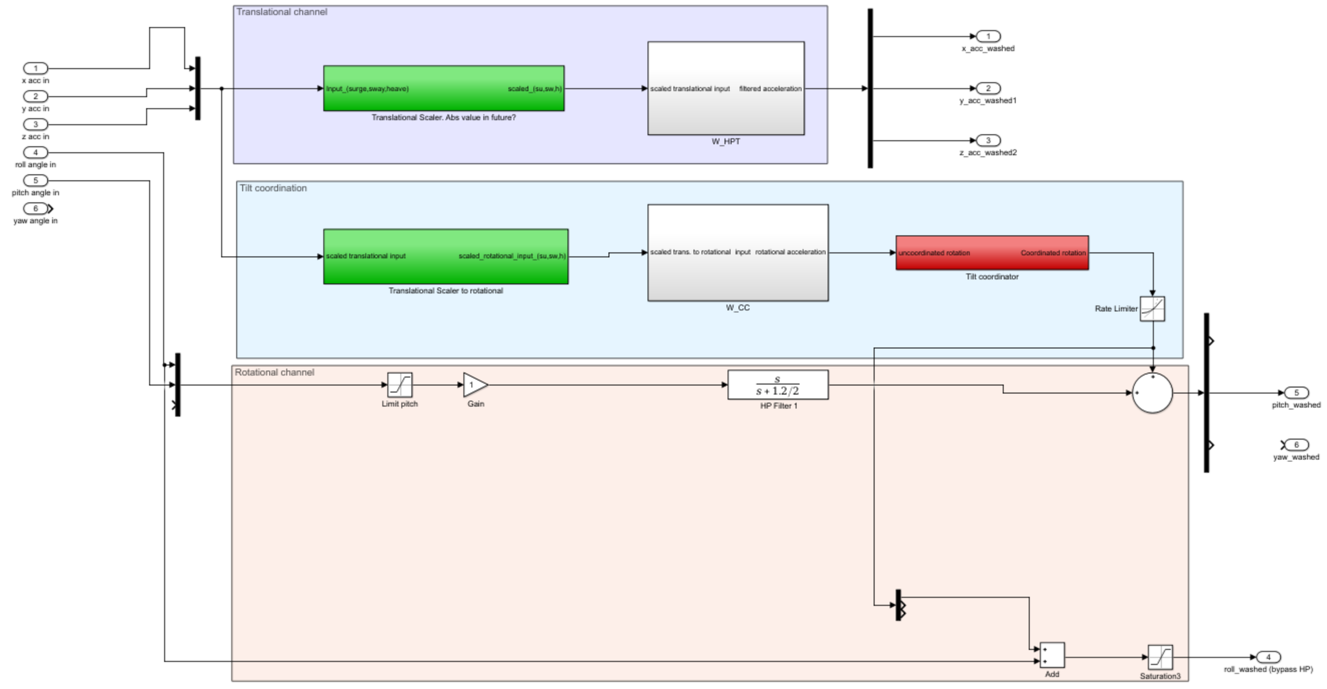

Translational channel
1 x acc in
2 y acc in
3 z acc in
4 roll angle in
5 pitch angle in
6 yaw angle in
Input_(surge,sway,heave) scaled_(su,sw,h)
Translational Scaler. Abs value in future?
scaled translational input filtered acceleration
W_HPT
1 x_acc_washed
2 y_acc_washed1
3 z_acc_washed2
Tilt coordination
scaled translational input scaled_rotational_input_(su,sw,h)
Translational Scaler to rotational
scaled trans. to rotational input rotational acceleration
W_CC
uncoordinated rotation Coordinated rotation
Tilt coordinator
Rate Limiter
Rotational channel
Limit pitch
1
Gain
$\frac{s}{s+1.2/2}$
HP Filter 1
5 pitch_washed
6 yaw_washed
Add
Saturation3
4 roll_washed (bypass HP)

Figure C79: Complete washout filter

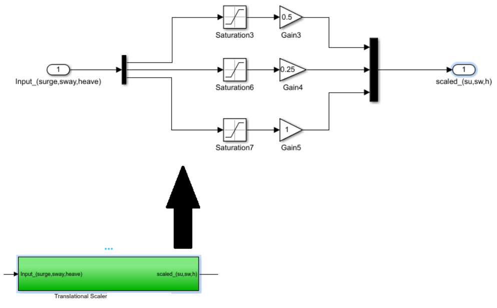


Figure C80: example of Limit and scaler

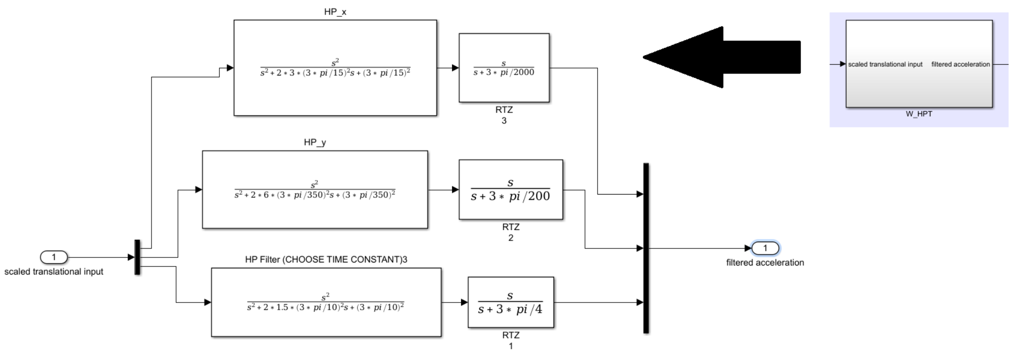

HP_x
s^2/(s^2+2*3*(3*pi/15)^2 s+(3*pi/15)^2)
s/(s+3*pi/2000)
RTZ 3
HP_y
s^2/(s^2+2*6*(3*pi/350)^2 s+(3*pi/350)^2)
s/(s+3*pi/200)
RTZ 2
HP Filter (CHOOSE TIME CONSTANT)3
s^2/(s^2+2*1.5*(3*pi/10)^2 s+(3*pi/10)^2)
s/(s+3*pi/4)
RTZ 1
1 scaled translational input
1 filtered acceleration
scaled translational input
filtered acceleration
W_HPT

Figure C81: Layout for transfer functions

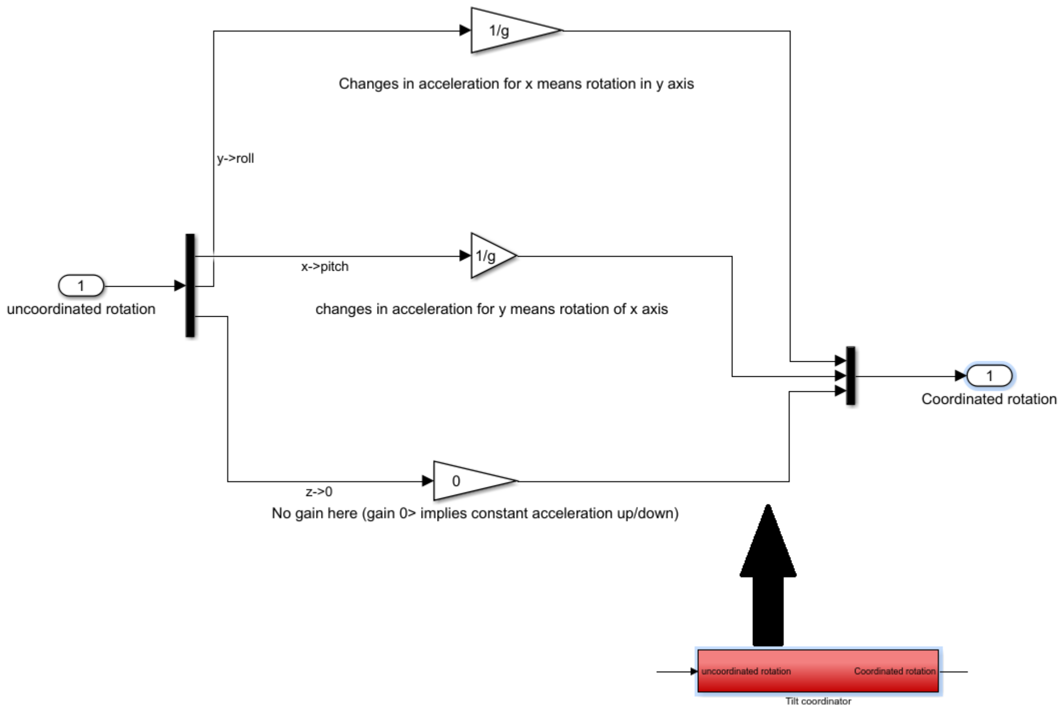

1/g
Changes in acceleration for x means rotation in y axis
y->roll
1/g
x->pitch
changes in acceleration for y means rotation of x axis
1 uncoordinated rotation
1 Coordinated rotation
0
z->0
No gain here (gain 0> implies constant acceleration up/down)
uncoordinated rotation
Coordinated rotation
Tilt coordinator

Figure C82: Tilt co-ordination block

Function	Limits and scalers	, Time constants and dampening
Translational channel		
W_{HPTx}	$Limit = \pm 6m^2/s,\ gain = 0.5$	$\omega_{HPx} = 0.6283$, $\zeta_{HPx} = 2$,$\omega_{RTZx} = 0.004$
W_{HPTy}	$Limit = \pm 6m^2/s,\ gain = 0.25$	$\omega_{HPTy} = 0.0269$, $\zeta_{HPx} = 6$, $\omega_{RTZy} = 0.047$
W_{HPz}	$Limit = \pm 3m^2/s$	$\omega_{HPz} = 0.9425$, $\zeta_{HPz} = 1.5$, $\omega_{RTZz} = 2.356$
Co-ordination channel		
W_{CCx}	$Limit = \pm 1m^2/s,\ gain = 4$	$\omega_{CCx} = 10$, $\zeta_{CCx} = 0.1$
$W_{CCy}3$	$Limit = \pm 1.8m^2/s,\ gain = 2.222$	$\omega_{CCy} = 10$, $\zeta_{CCz} = 2$
Rotational channel		
W_{RCp}	$Limit = 16.7°,\ gain = 1,\ Limitonoutput =°$	$\omega_{RCp} = 0.6$

Appendix D Crash Function

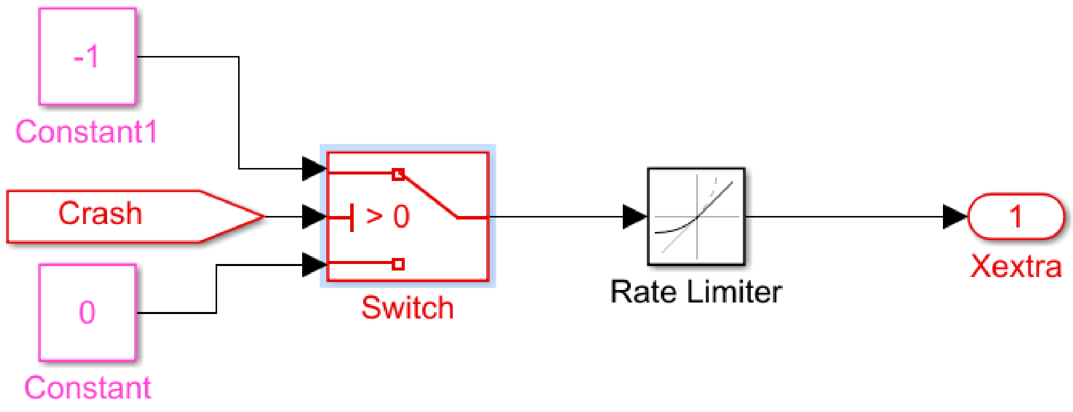


Figure D83: Simulink Crash function

The rate limiter has a raising rate limitation 0 and a falling rate limitation of -5. This allows for crashes to move the platform backwards in a quick fashion. With a set sampling time the falling rate in the rate limiter decides how quick the jerk back is.

Appendix E Project Cost

The budget for the project was 25000 SEK. The cost of this project was relatively low since the most valuable part of the project was the Steward platform and the motion PC that controls the platform which both were provided by KTH and were free to use. Other than that, items that needed to be purchased was a graphics card for the host PC to be able to use the VR-headset, Arduino micro-controllers, a Raspberry Pi, a H-bridge for DC-motor, etc. The specific costs are listed in the table below. The total cost of this project was 19077 SEK which was within the budget. All prices excludes taxes.

Table E5: List of costs for the project.

Item	Amount	Price in SEK
Oculus Rift	1	7000
Nvidia GTX 1060	1	2199
Arduino Mega 2560	2	399
Throttle sensor	2	65
Clutch sensor	2	19
Rotational potentiometer	3	59
Raspberry Pi 3	1	299
Raspberry Pi 3 charger	1	69
Sandisk SD-card 32GB	1	199
USB-B cable 5m	1	299
Unity map	1	300
H-bridge	1	1500
DC-motor	1	1000
Headphones	1	239
3D-printing material	0,35 Kg	53
Steel 5mm	1.2 m^2	50
Wood	0.02 m^2	10
Steel wire	1 m	89
Back light	1	29
Motorcycle parts	1	2600
Motorcycle frame	1	2600
Total		**19077**