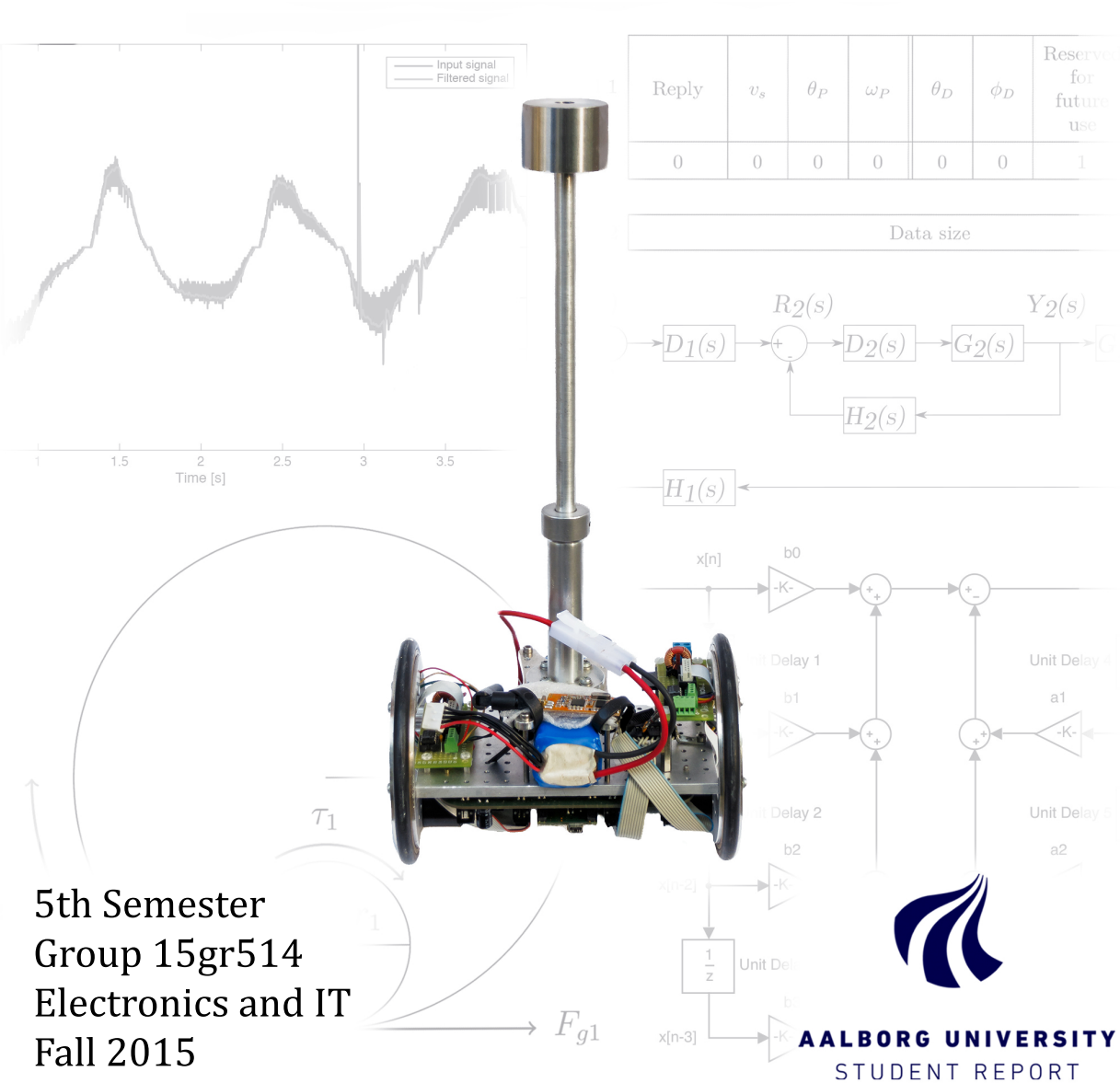
Remote Controlled Segway

Modelling and control of a motorized inverted pendulum

Andrea Tram Løvemærke Poul Hoang Ralf Ravgård Christensen
Rasmus Gundorff Sæderup Thomas Guyot Thomas Kær Jørgensen



5th Semester
Group 15gr514
Electronics and IT
Fall 2015

AALBORG UNIVERSITY
STUDENT REPORT

This report is compiled in LaTeX, originally developed by Leslie Lamport, based on Donald Knuth's TeX. The main text is written in *Computer Modern* pt 11, designed by Donald Knuth. Flowcharts and diagrams are made using Microsoft Visio, Inkscape and Tikz, a TeXpackage for generating graphics.

Institute of Electronic Systems
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Segway

Theme:
Digital and Analog Systems Interacting with the Surroundings

Project Period:
5. Semester

Project Group:
15gr514

Participants:
Andrea Victoria Tram Løvemærke
Poul Hoang
Ralf Victor Lømand Ravgård Christiansen
Rasmus Gundorff Sæderup
Thomas Paul Guyot
Thomas Kær Juel Jørgensen

Supervisor:
Christoffer Sloth

Co-supervisor:
Rasmus Pedersen

Copies: 9

Page Numbers: xx

Date of Completion:
December 17, 2015

Abstract:

In this project, a controller has been designed to make a segway stand in an upright position. A model of the segway has been derived, describing it as two motors moving the base of an inverted pendulum. Tests show a fit of 84% for the motor model and 70% for the inverted pendulum model. Based on these models, two controllers are designed. A P-controller for the motor and a PID-controller for the inverted pendulum, coupled in a cascade controller.
The controller is able to stabilize the model of the system, and fulfilling the requirements for the settling time and rise time, but with an overshoot of 23.5% and a steady-state error of 12.3%. Digital filters are designed to reduce the noise in the sensor measurements. The filters are low-pass Butterworth filters, which are transformed into the digital domain using bilinear transformation, but they are not implemented due to computational power constraints. A communication protocol is implemented, in which a package structure and package header has been designed, to facilitate the wireless communication between the segway and a remote controller.
After tuning the controller parameters, the segway is able to stand upright, and is also able to withstand a light push. It is also possible to drive the segway back and forth using the remote controller, as well as receive data from the segway wirelessly.

Contents

Preface

This report has been carried out during the fall of 2015 as a 5. semester Electronics and IT student project at Aalborg University by group 15gr514. The project concerns the development of a controller system that will make a given segway balance and allow a user to wirelessly remote control it.The report has the following structure: First, the segway platform is described and requirements to the controller are made. Hereafter, a model of the system is derived, and the controller can be made. Also, two digital filters are designed, together with a communication protocol between the segway and the wireless remote controller. Finally, an acceptance test and a discussion is made.
The reader of the report is assumed to have a basic understanding of physics, modelling electromechanical systems and basic controller design. The reader is also assumed having an understanding of electronic components, C-code, filters and methods for filter design, as well as a knowledge of the frequency domain. References to sources are of the APA-style, i.e. of the type [*source name/author's surname, year, optional page number*]. Sources are listed in a bibliography at the end of the report, and PDFs, datasheets etc. referred to throughout the report can be found on the attached CD. Figures without a source are made by the project group.

This project has been supervised by assistant professor Christoffer Eg Sloth as main supervisor with PhD fellow Rasmus Pedersen as co-supervisor. The project group would like to thank Aalborg University for providing the segway in the project. Also thanked is engineering assistant at Institute for Electronics Systems at AAU, Simon Jensen, for his help with the hardware used in the project. Machine worker Jesper Dejgaard Pedersen at Aalborg University is also thanked, for his help with the wheels and gears on the segway. Finally, group 15gr633 are thanked for their help with discussing the workings of the system, and providing the framework for the I²C driver functions.

Aalborg University, December 17, 2015

Andrea Victoria Tram Løvemærke
alavem13@student.aau.dk

Poul Hoang
phoang13@student.aau.dk

Ralf Victor Lømand Ravgård Christiansen
rvlr13@student.aau.dk

Rasmus Gundorff Sæderup
rsader13@student.aau.dk

Thomas Paul Guyot
tguyot15@student.aau.dk

Thomas Kær Juel Jørgensen
tkjj13@student.aau.dk

1 | Introduction

The inverted pendulum is a classical control system, aiming at getting a pendulum with its center of mass above the pivot point to balance, see Figure 1.1. The pendulum consists of a mass m_p on a rod with length l, located on a moving base. A force F_F is applied to the base, so the tilting angle of the pendulum, θ_p, is minimized.

Whereas a regular pendulum is a stable system with an equilibrium point, the inverted pendulum is inherently unstable. This means that unless some action is taken, the pendulum will fall over. To keep the pendulum balanced, it is necessary to measure the tilting angle θ_p, and, based on a model of the system, make the base move in the correct way by means of an applied force F_F.

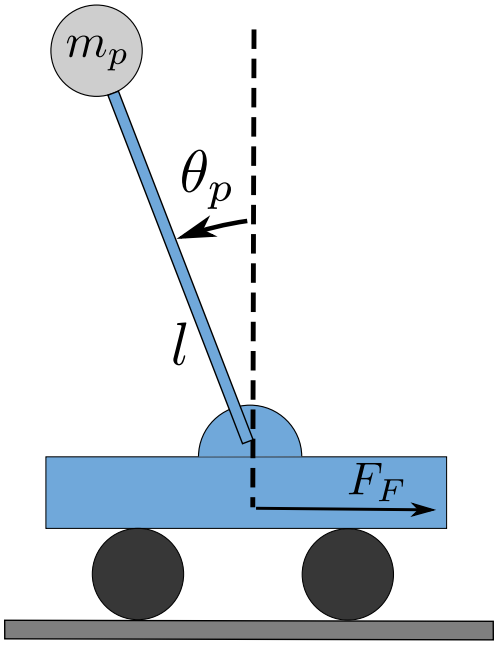


Figure 1.1: An inverted pendulum placed on a moving base

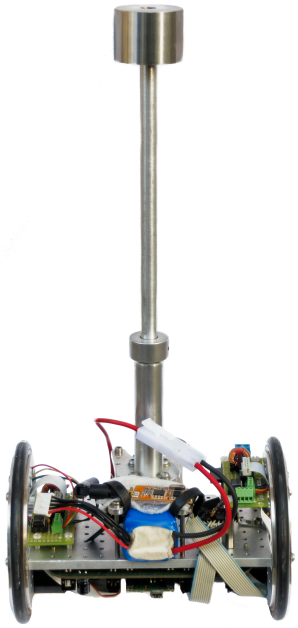


Figure 1.2: The segway to be used in the project (approx. height is 37 cm)

The inverted pendulum has found a new application in recent years, namely in the form of the segway. A segway is a small vehicle that can be used for human transportation where the user stands on a platform placed between two wheels.

This project concerns the development of a controller for a pre-manufactured segway provided by Aalborg University as can be seen in Figure 1.2. This controller is to stabilize the inverted pendulum in it's upright position, and control the segway's position. The segway is seen as toy to be driven around indoors, and not as a model of a full-size segway. Because of this, all mentions and analyses of a segway throughout the report is in regards to the segway, and not a full-size segway.
In the following chapter, a presentation of the general workings of segway is presented.

2 | Segway Presentation

In the following chapter, an overview of how a segway works is presented. This includes a block description of the system, and a description of the functionalities it is to have. The segway provided by Aalborg University is also described, in regards to the mechanical frame and the provided hardware.

2.1 Generalized Segway Description

A segway can be seen as a typical control system, as shown in Figure 2.1. Here, the controlled parameters are acquired using various transducers, processed by the system controller and then a control signal is outputted to the actuator(s) through. Apart from the control system, a Remote Controller (RC) is added to the system, which can also be seen in Figure 2.1. The reason for this will be described later on. In the case of the segway, the transducers are sensors, providing information on the movement of the segway, and the actuators are motors, which are driving the wheels.

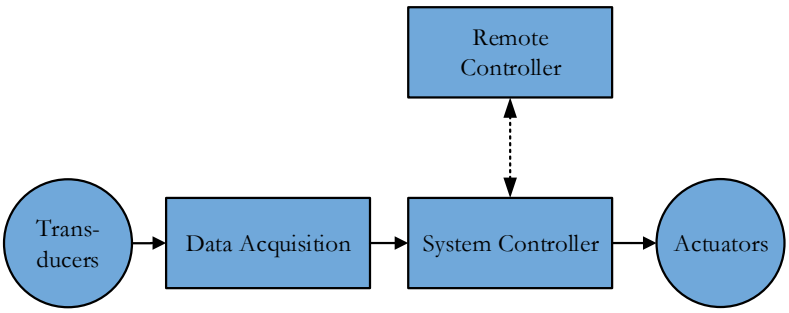


Figure 2.1: Generalized block diagram of a segway control system.

It is decided that the segway is to have four main functionalities to function as desired. These functionalities are *balancing, driving, turning and the ability to be remote controlled*. Each of these are described in further detail in the following section.

2.1.1 Functionalities of the System Controller

The controller unit has four main functionalities, namely balancing, driving, turning, and transceiving wireless signals between the RC and the system controller. Each of these functionalities are described in further detail in the following section.

One of the required functionalities of any segway is the ability to balance. The system controller must control the motors so the inverted pendulum is balancing vertically. Because the pendulum is unstable by nature, the controller has to make the system stable. This functionality relies

on input from the transducers, which is used to calculate an actuator signal to keep the segway balanced.

Through a wireless RC the segway is remote controlled. This means that the segway must be capable of receiving commands from the RC and make the segway move accordingly. To achieve this, the segway and RC must share a communication protocol. The wireless RC should also able to receive information regarding the movement of the segway like the speed, angular velocity and angle.

The driving and turning functionalities are responsible for controlling the segway's forward and backwards velocity as well as the direction of the segway. To ensure the segway does not fall over when driving or turning, the movement will of course have to happen in corporation with the balancing functionality, since balance is still to be kept when driving and turning.

2.2 Description of Provided Segway

Before the modelling and the design of the controller can begin, the segway provided by the university first has to be described, especially in regards to the electronic hardware used on the segway. This is done in the following section. A photo of the provided segway can be seen in Figure 2.2.

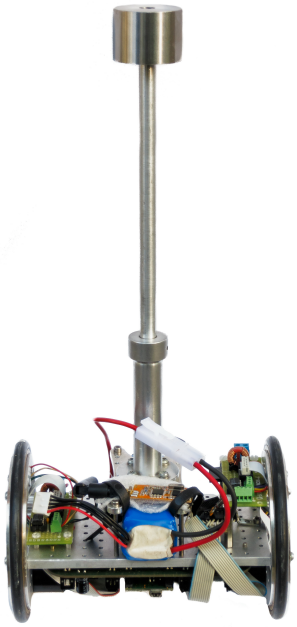


Figure 2.2: The segway to be used in the project.

The segway consists of several items, these items are divided into 3 main groups, namely **mechanical**, **power** and **electronics**. These groups and the items they include are described in the following.

2.2.1 Mechanicanics

Mechanical Frame

The provided segway's mechanical frame is made from aluminium. It consists of a baseplate onto which the wheels are mounted, together with a rod with a cylindrical mass on top. All electronics including batteries are mounted on the baseplate.

A blueprint of the segway can be seen in Figure 11.1 in Appendix A, and measurements of relevant parameters of the frame is listed in Table 11.2, also in Appendix A.

Wheels

The wheels mounted on the segway feature an offset motor, meaning the motor is not mounted at the center. Though, as seen from the blueprint in Figure 11.1 in Appendix A, the axis of the wheel is still in the center of the base. The wheel system is built as follows: The motor rotates a gear fitted onto the motor from the factory. The shaft from this gear is located on the inner side of the wheel, and mounted to a cogwheel. The inner side of the wheel itself has some small cogs making it the other part of the transmission system. For a better rotation and less friction, the wheel has three ball bearing guides. This can be seen in Figure 2.3. The cogwheel mounted on the motor wheel has 25 cogs and the inside of the wheel has 90 cogs.

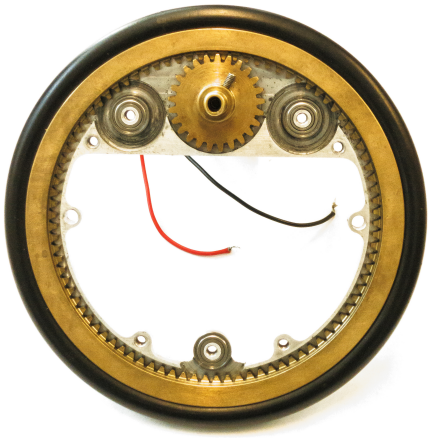


Figure 2.3: The wheel and motor showing the transmission. The motor is behind the small gear.

Motors

The segway uses two Maxon A-Max 110160 permanent DC-motors to rotate the wheels. Parameters of the motors that might be taken into account are the rotational speed and the torque. The specification of the motors are [**?**]

- Nominal torque at 6 V is 5.91 mNm.
- Nominal speed at 6 V is 6240 RPM.

The motors are brushed, meaning the stator corresponds to the magnets and the rotor is composed of the coils, receiving power through brushes. Brushed motors have important advantages: They have a low cost of construction, require only simple and inexpensive control and they operate in extreme environments due to lack of electronics. However, those motors also have drawbacks: They require periodic maintenance, their characteristics are moderately flat and at high speeds, brush friction increases, thus reducing useful torque. Also, the brush arcing will generate noise causing electrical magnetic interference (EMI) [**?**] [**?**].

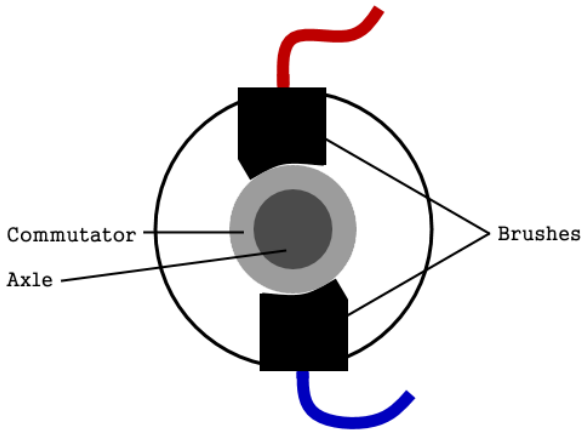


Figure 2.4: Sectional view of a brushed motors.

As a part of the motors, a gear (Maxon 110356) has been fitted on the motor. This gear was fitted from the factory, and the inner workings of it are therefore not known. From the datasheet, the gear ratio between the motor and shaft is known [**?**]:

$$N_{ms} = \frac{1}{19} \approx 0.053$$

2.2.2 Power

Batteries

The segway comes equipped with a battery which is the source of power of the system. The battery is a rechargeable 4S LiPo. LiPo batteries, or Lithium-ion Polymer batteries, consists of cells in series, xS meaning $\boldsymbol{x}$ cells in **S**eries. A LiPo cell has a nominal voltage of 3.7V. The battery used on the device has a nominal voltage of 14.8 V (3.7 Volts times 4 cells). LiPo batteries, compared to other types such as Nickel–metal hydride (NiMH), one of the most common battery types, are lighter and more compact. However, they are more expensive than "regular" batteries and require a specific charger to bring every cell of the pack to the same state of charge [**?**].

The Segway also has two Power Supply Units (PSU's). These are electronic components, whose role it is to regulate the unstable input voltage from the batteries, to a stable voltage to be used by the electronics. One has an output of 5V and the other has an output of 3.3V.

2.2.3 Electronics

Wireless Transceivers

To facilitate the wireless communication between the segway and a remote controller, a pair of APC220 radio communication modules is supplied. The modules have the following features [**?**]:

- Transmission distance of up to 1000 m (2400 baud)
- UART/TTL interface with 256 bytes data buffer
- -112 dBm sensitivity at 9600 baud
- 20 mW output power
- Adjustable frequency from 418 MHz to 455 MHz

Since the modules have a UART interface, they can be interfaced through a UART interface on a microcontroller, which makes the radio modules simple and easy to use.

Microcontroller

A microcontroller board is supplied with the segway. The printed circuit board (PCB) on which all the electrical components are mounted, is designed to fit the supplied microcontroller unit (MCU), and thus a new MCU is not chosen, since it might require a redesign of the PCB.

The microcontroller board supplied is a *CrumbX128A3* made by Chip45 [**?**]. The CrumbX128A3 features an ATxmega128A3U microcontroller, a 3.3V low dropout (LDO) voltage regulator, a USB UART converter, an RS485 transceiver and a micro-sd-card header/slot.

The ATxmega128A3U has the following specifications [**?**]:

- 32 MHz clock speed
- $1.6 - 3.6$ V power supply
- 128 Kb flash
- 8 KB SRAM
- 7 UARTs
- 7 16-bit counters
- 4-channel DMA controller

Accelerometer and Gyroscope

To measure the angle the segway is tilted as well as its relative position, an accelerometer and gyroscope is used. Here, the InvenSense MPU-6050[**?**] is used, a 6-axis gyroscope and accelerometer. The chip is implemented on a GY-521 breakout board. It features an I²C bus, from which the raw sensor values can be read. The gyroscope can be configured to having a full-scale range of either $\pm 250, \pm 500, \pm 1000$ or $\pm 2000°$/sec. The accelerometer can be programmed to have a full-scale range of $\pm 2g, \pm 4g, \pm 8g$ or $\pm 16g$ [**?**].

Motor Driver

To control the motors, a motor driver is used, in form of an H-bridge. The H-bridge allows a small controller-voltage to control a (often relatively large) current. This is practical, since the current needed to drive most motors is bigger than what can be drawn from the pins of a microcontroller. Here, two Freescale MC33926 5A H-bridges [**?**] are supplied - one for each motor. The H-bridge makes it possible for a motor to be run both forward and backwards, based on the input signals to the chip. It also features a feedback-pin, also known as a current sense, that can be used to measure the current drawn by the motor. The H-bridge is controlled by an input PWM signal and two direction pins. By setting these pins, the motor can be either locked, free-running or moving forwards or backwards.

Encoders

In order to keep the inverted pendulum in balance, the segway needs a way to quantify the amount of rotation on the wheels and, by extension, on the motors, hence the encoders. The ones currently used by the system are relative (or incremental) encoders placed directly on the motors. An incremental encoder delivers a certain number of pulses per revolution. This number of pulses measures the angular movement. Usually, relative encoders consist of a disc divided in transparent and opaque segments. Most of those discs are equipped with two rows of segments plus a top segment. Both segment tracks are slightly out of phase to indicate the direction of rotation and the additional top segment allows the count of revolutions. An incremental encoder's resolution corresponds to the maximum number of pulses sent per revolution and is symbolised by the unit "points

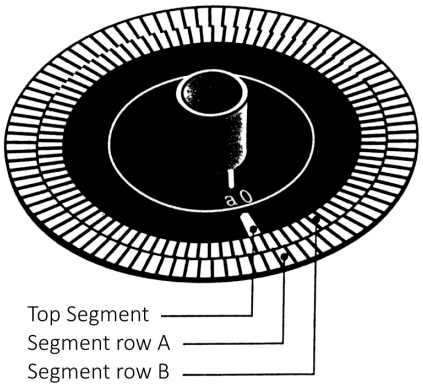


Figure 2.5: Detailed scheme of an incremental encoder [**?**].

2.2.4 Detailed Segway Overview

The general block diagram of the segway control system in Figure 2.1 can be extended based on the provided hardware described above. The detailed diagram can be seen in Figure 2.6.

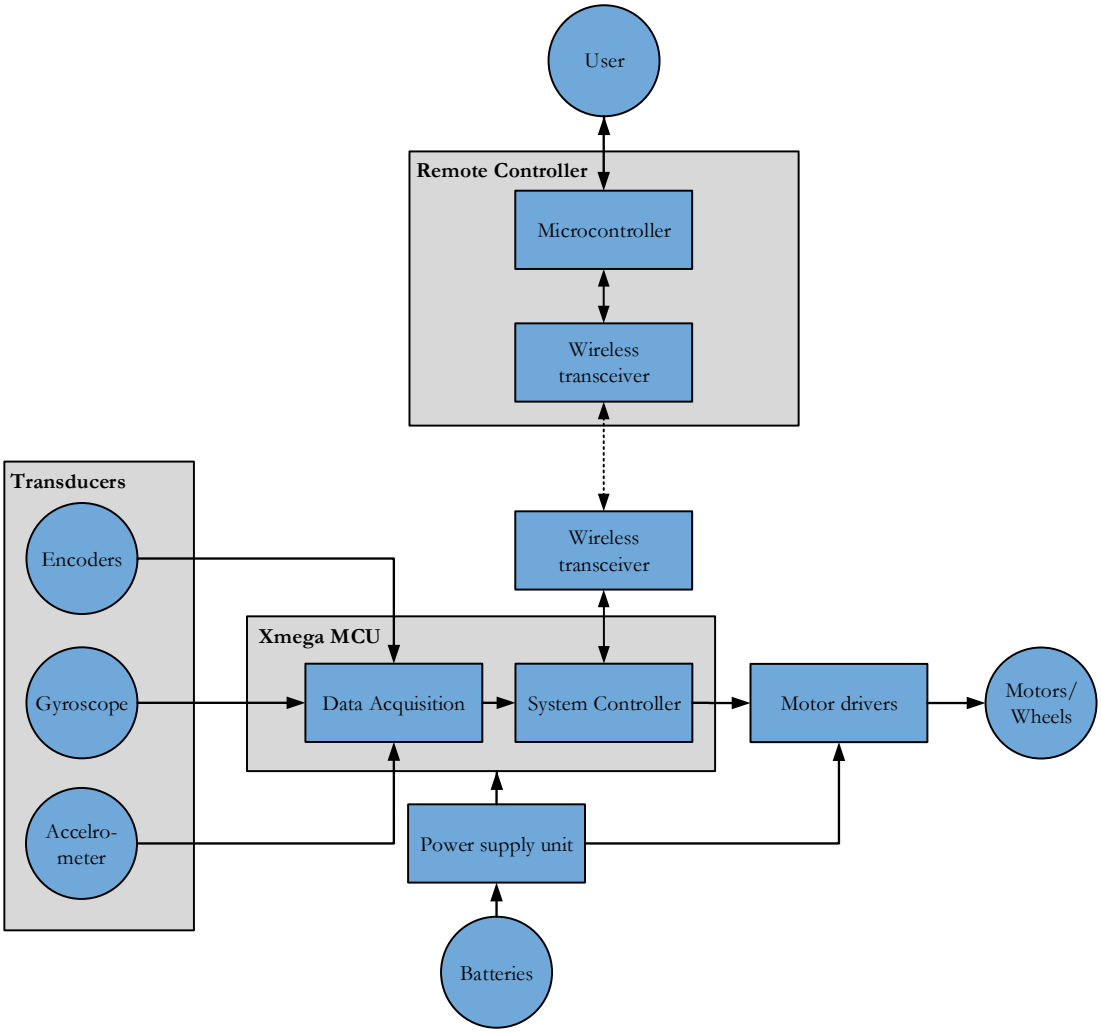


Figure 2.6: Detailed block diagram of the segway system.

In the detailed diagram, the different transducers have been added, and the functionalities the MCU is to facilitate have been grouped. Also, the actuator has been specified, in the form of motors. The workings of the remote controller are also described in further details, namely that it is to receive user input to a MCU and then transmit it wirelessly. It can also display data from the segway to the user. Also added to Figure 2.6 are the batteries and PSUs.

2.3 System Overview

Before the design of segway can begin, it has to be determined which subsystems are to be designed. In Figure 2.6, the detailed block diagram of the system was presented. As most of the hardware for the segway is already provided, the project will primary focus on designing the controllers in the system. This limitation can be seen in Figure 2.7. Here, the blue blocks are subsystem that are to be designed, while the grey blocks are not designed.

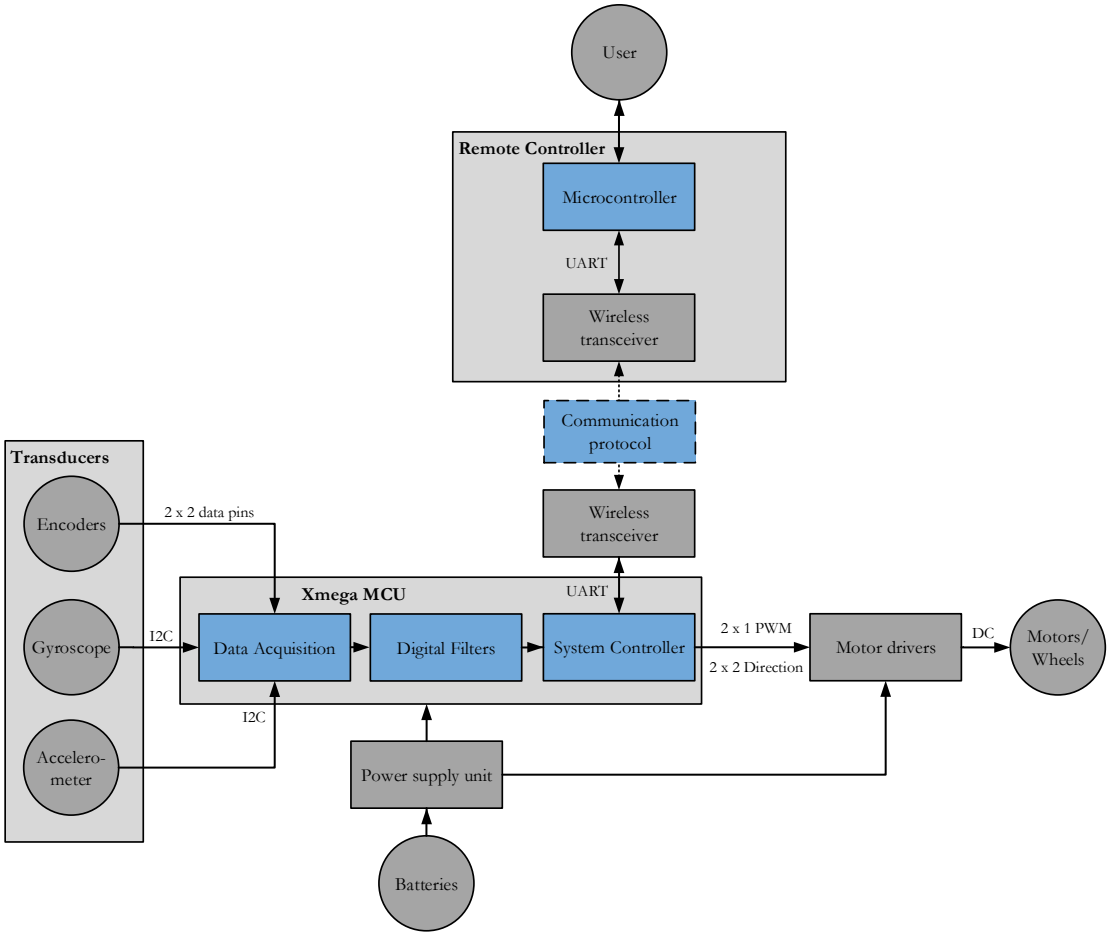


Figure 2.7: Overview of the system. The blue blocks are blocks the design primary deals with.

Note that a communication protocol has been inserted between the wireless transceivers. This is done because some "rules" has to be introduced in the communication between the remote controller and the MCU on the segway to ensure a proper data transfer without misunderstandings.

Also, a filter is inserted between the encoders and the MCU. To avoid breaking electrical interfaces and changing the electrical configuration of the segway, the filter is implemented as a digital filter, instead of an analog.

To ensure that subsystems can be designed parallel, the interfaces between subsystems have to be clarified. Data types transmitted between subsystems are shown in Figure 2.7. From left, the gyroscope and accelerometer transmits measured values through an I^2C communication protocol, and the encoders data through two data pins. For external communication to the remote controller, data is transferred through a UART interface which is supported by both the wireless transceiver and the microcontroller. To control the motor drivers, a PWM signal is used for each motor, together with two direction pins.

2.3.1 Control Loop Overview

The system that controls the balancing of the segway can be described as a closed loop system, as shown in Figure 2.8. The segway is balanced through the use of three subsystems, namely the controller (D), the plant (G) and the sensors (H). It is assumed that the sensors are ideal, meaning that their frequency response is equal to unity i.e. $H(s) = 1$, as this block won't include any amplification or filtering, but solely measure. This is done since it is decided not to determine the sensor's transfer function, and that it will make the calculations easier by assuming that the measued output data is identical to the actual output. The input, R, is the desired angle of the segway, which for the balancing functionality, will be zero, i.e. the steady state upright position. The output Y is the actual angle of the segway, denoted as θ_p.

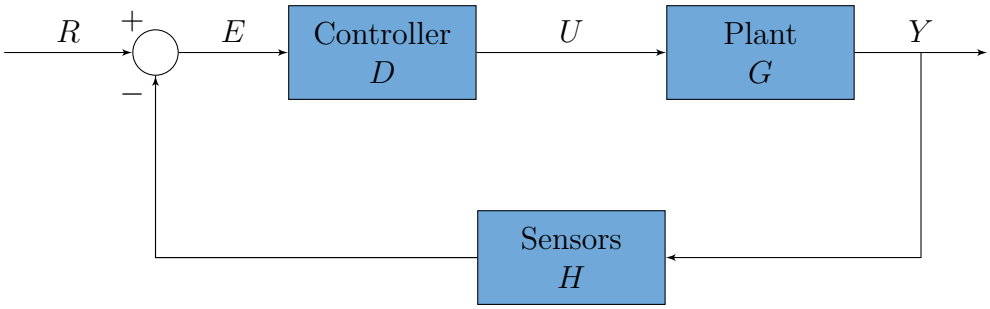


Figure 2.8: A feedback loop of the system that controls the balance of the segway.

The plant is based on a model of the system that describes the movement of the segway, where the output is $Y = \theta_p$. This model is found in chapter 5.

The controller block, D, generates the input to the plant, based on the difference between the desired angle, R, also known as the reference, and the actual angle measured by the sensor system, H. The output of the controller, U, is generated so that the error between output of the plant, Y, and the desired angle is minimized. The design of the controller unit is performed in chapter 6.

Now that the segway platform has been described, it is possible to set up the requirements for the system, which is done in the following chapter.

3 | Requirements

This chapter has the purpose of presenting considerations concerning setting up requirements for the the segway. Firstly, the considerations will be made upon a list of requirements set from the preanalysis as well as common sense, for those requirements which has to be set by choice rather than theory. These considerations will result in a list of specific requirements, from which the modelling and controller design of the segway will be designed upon.
The requirement considerations follows in the next section.

3.1 Requirements Considerations

The most important functionality of a segway is for it to keep its balance. For this to be achieved, the segway must be able to obtain a stable position after being given an impulse, such as a push. It is preferable to set requirements for the system's step response, as this is a typical measure for a system's dynamic behaviour that is rather easy to measure and set requirements for. In the following, the system's step response's parameters will be described. Afterwards the relation of the system's stability and the phase margin and gain margin will be presented.

3.1.1 Functional Requirements

For the system, there are some requirements for the overall functionalities of the system. These requirements are derived from intuition, based on what a segway should be able to do. This includes that the segway has to stabilize itself and balance in an upright position, as well as being able to drive and turn, without falling over. Also, the segway should be able to withstand a light push, without falling over. A "light push" is not defined further, but the push needs to be so big that it makes the segway move in order to stabilize. It is also decided that it should also be possible to remote control the segway wirelessly, by means of making the segway drive and turn. The user should also be able to wirelessly receive information from the segway, such as the pendulum angle, θ_p.

3.1.2 Time Domain Requirements

The step response displays the time behaviour of the system when given an impulse. Figure 3.1 shows an example of a step response of a dynamic system.

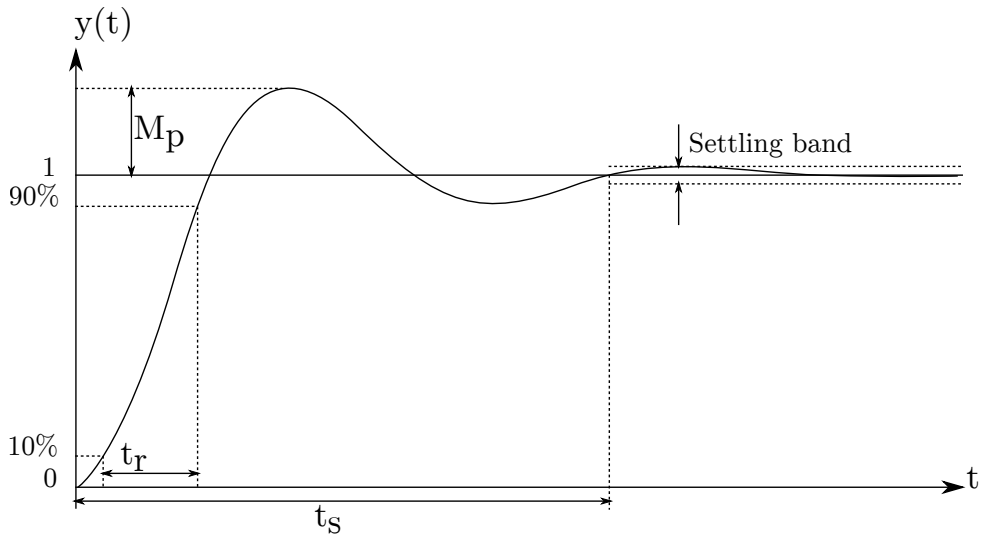


Figure 3.1: General step response in time domain.

In Figure 3.1, the variable t_r is the rise time, defined as the time it takes for the step response to go from 10% to 90% of the step. The settling time, t_s, describes the time it takes for the system to settle within a certain margin of the final value, typically 1%, 2% or 5%. The steady state error is the difference between the desired steady state value, and the actual steady state value. The ratio between the step response's ideal level and the maximum level which the system reaches is the overshoot, M_p.
To make sure the control system is stable, the system should not have any poles in the right-half-plane of the s-plane [**?**, p. 146], or any poles outside the unit circle, if considering the z-plane [**?**, p. 619].

It is desirable not to have a steady-state error, as the balancing of the segway shall be as accurate as possible, to ensure that the angle the segway is to maintain is also met, as an steady-state error in the inverted pendulum angle will cause the segway to move forward or backwards. It is likewise desirable to have a fast rise time, as it would decrease the risk of the segway of falling over if the controller regulates the segway to an upright balanced position quickly. However, this will result in an overshoot, as the system's quick response to the impulse, for example a push, will make the system overcompensate to minimize the error quickly. This will increase the risk of the segway of falling over, if the overshoot is so high that the segway falls to the other side. The overshoot can also cause oscillations, if the controller constantly tries to obtain the desired angle, but has an overshoot so high, that the overshoot angle also has to be corrected by the controller. Disturbances to the controller making the system unstable, such as the gravity, can affect the way the segway moves forward at a steady velocity. It is important to reduce both disturbances and noise affecting the sensor values, to obtain a control system with high precision.
The segway's balance controller shall thus, as with any control system, be a compromise of an agile behaviour, where the settling time is not too long and where overshoot is limited.

The step response is a useful means of evaluating the dynamics of a system, i.e. the rise and settling time, the overshoot and the steady state error. The step is applied to the closed loop, as the step response then includes the effect of the feedback.

The general transfer function for the basic closed loop feedback system is shown in Equation 3.1 and the corresponding block diagram is shown in Figure 3.2.

$$T(s) = \frac{Y(s)}{R(s)} = \frac{\text{Direct term}}{1 + \text{Open loop}} \tag{3.1}$$

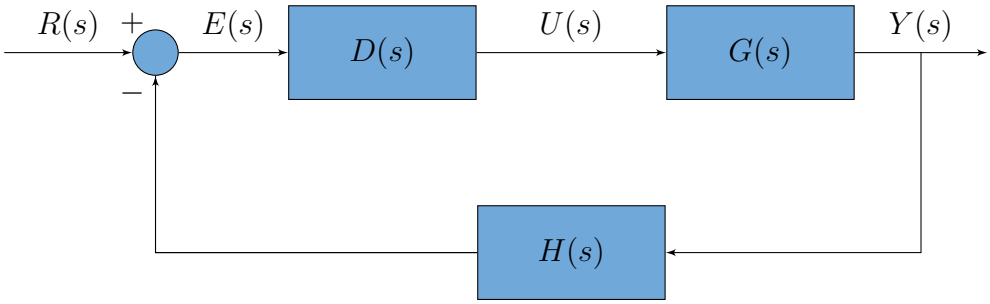


Figure 3.2: Block diagram of a general closed loop feedback system.

It is assumed that the system that is to be controlled is a second order stable closed loop system, on the form as listed in Equation 3.2.

$$T(s) = \frac{A}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{3.2}$$

Where:

A	is the DC gain	[1]
ζ	is the damping factor	[1]
ω_n	is the natural frequency	[rad/s]

Due to this, the relationship between the different quantities can be expressed from the following rules of thumb [**?**, p. 152]:

$$M_p = \exp\frac{-\pi\zeta}{\sqrt{1-\zeta^2}} \tag{3.3}$$

$$t_r \approx \frac{1.8}{\omega_n} \tag{3.4}$$

$$t_s = \frac{-\ln(x)}{\zeta\omega_n}, x = \text{settling band, e.g. 0.01 for 1\%} \tag{3.5}$$

$$\zeta \approx \frac{PM}{100} \tag{3.6}$$

Assuming that the system is 2nd order, it is thus possible to set up realistic requirements for the system based on these equations. First of, it is decided that no steady-state error is acceptable, as it will cause the pendulum to move forward at a steady velocity at all times, which is not desired.

It is chosen that the overshoot should be no more than 10%, i.e. $M_p \leq 0.10$. Using Equation 3.3, ζ is found to be equal to $\zeta = 0.8$. By deciding that the settling time t_s should be less than 3 seconds for a 1 % settling band, it is found that $\omega_n = 1.92$ using Equation 3.5 and the found value for ζ. This can be inserted in Equation 3.4, giving a rise time of $t_r = 0.94\,s$. This is considered acceptable, but to give some margin, the requirement is increased to 1 s. Thus, the requirements for the step response of the system has been determined. The requirements are summed up below.

Steady state error: 0 %
Rise time: $<$ 1 s
Settling time: $<$ 3 s
Overshoot: $\leq$ 10 %

With requirements set up for the time domain response of the signal, it is now desired to set up requirements for the frequency domain. This is done by means of requirements to the phase and gain margin.

3.1.3 Frequency Domain Requirements

Phase margin (PM) and gain margin (GM) are two correlating values that describes how far the system is from instability, i.e. how much uncertainty can be allowed before it may cause the system to become unstable [**?**]. For this reason, the requirements for the segway should take these margins into account.

The closed loop feedback system will be unstable if the signal that is fed back from the output to the controller, see Figure 3.2, reinforces the error signal, E rather than diminishing it. This is the case if the open loop term is negative, i.e. the signal is phase shifted more than $-180°$ at unity gain.

Thus, through analysis of the open loop, it can be determined if the closed loop will be stable. The analysis is performed by determining the PM and GM. This can be done through bode plots of the open loop, as shown in Figure 3.3 and described in the following.

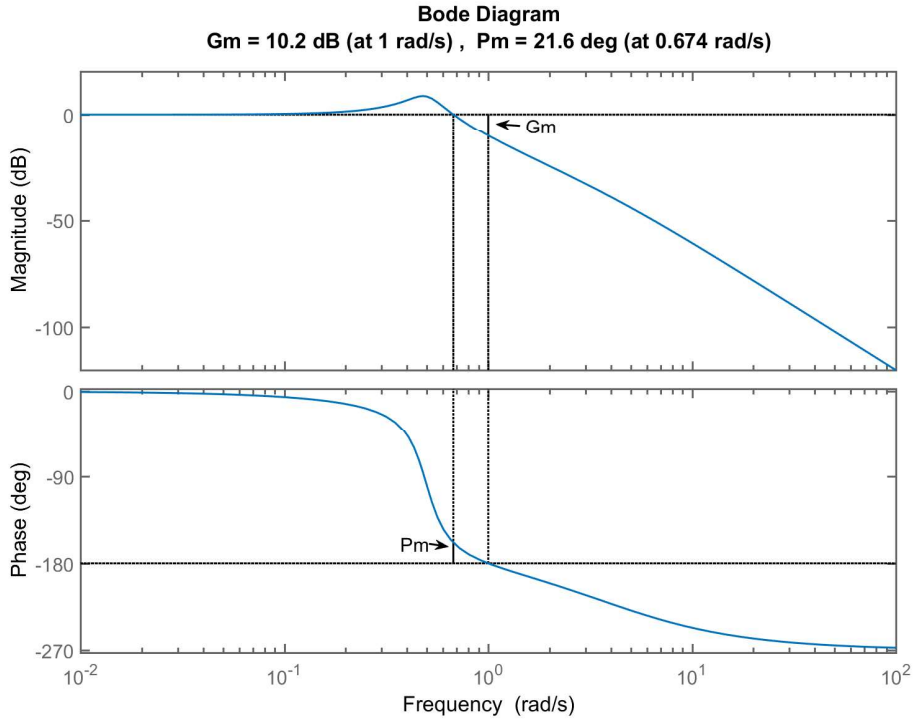


Figure 3.3: Bodeplots of an open loop illustrating PM and GM.

The phase margin is found by reading the angle velocity from the magnitude bode plot at gain = 0 dB, and finding the phase angle at this anglular velocity on the phase angle plot. The phase angle subtracted from $-180°$ yields the PM [**?**].

The gain margin is found by reading the angular velocity from the phase angle plot at the phase of $-180°$, and finding the gain at this angular velocity on the magnitude plot. The difference between this gain and unity gain (0 dB) is the gain margin.

From Figure 3.3, it can be concluded that the phase margin is defined as how much additional phase shift it would take before the system becomes unstable. On the other hand, the gain margin describes how much the gain may change before the system becomes unstable.

In the beginning of this section, it was stated that the purpose with PM and GM is to keep uncertainties from making the system unstable. In this context, uncertainties being everything that causes the closed loop system to not behave ideally, such as circuit component sensitivity, delay caused by sampling etc. The scale of these inconsistencies is not yet know, so the exact influence these will have on the system cannot be determined. A rule of thumb regarding the PM and GM, states that PM should be greater than or equal to 45° whereas GM should be greater than or equal to 6 dB [**?**]. However, to fulfill the time domain requirements, it is necessary to have $\zeta \geq 0.8$, which, based in Equation 3.6, results in a phase margin requirement of at least 80°.
From the above considerations, a list of specific requirements can now be set.

3.2 List of Requirements

Based upon the previous considerations, requirements for the segway can be determined. The requirements are as follows:

Functional Requirements

1. The segway must be able to stabilise itself in an upright equilibrium state.
2. The segway must be able to drive forward and backwards without causing the segway to fall over.
3. The segway must be able to turn without causing the segway to fall over.
4. The segway may not fall over when pushed lightly.
5. The user must be able to make the segway drive by a wireless controller.
6. The user must be able to make the segway turn by a wireless controller.
7. The user must be able to receive data from the segway on a wireless controller.

Performance Requirements

The following requirements describes the desired performance of the segway. The first list of requirements is for the time domain, determining the dynamic behaviour of the segway.

- Steady state error: 0 %
- Rise time: ≤ 1 s
- Settling time: ≤ 3 s
- Overshoot: ≤ 10 %

The following list describes the desired stability margins, i.e. the frequency domain requirements:

- Phase margin: $\geq 80°$
- Gain margin: ≥ 6 dB

Now that the requirements are set, the development of the system can begin. This includes designing the controller which is to stabilize the system, together with a filter design and the design of the remote controller and the communication protocol. In the following section, a model of the system is derived, as this is needed to make a controller for the system.

4 | Sensors

In this chapter, the setup of the sensors used by the segway is described. In a control system the sensors measure the output of the system, Y, and feed it back into the system as seen in Figure 4.1.

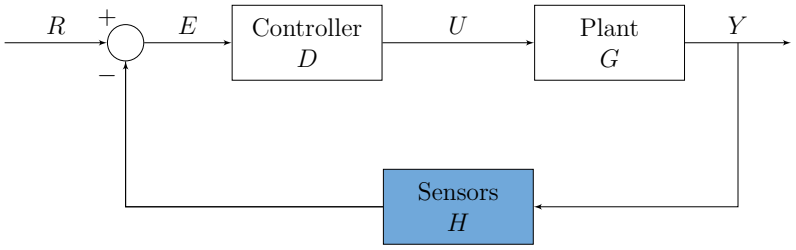


Figure 4.1: Feedback loop of a system, with the sensor, H, highlighted.

A general feedback loop, see Figure 4.1, consists of three blocks. A controller, D, the system that is to be controlled, known as the plant, G, and sensors, H. The loop holds a reference signal, R, an error signal, E, the control signal, U, between the controller and the plant, and an output, Y. It is the sensor block, that is to be determined in this chapter, as highlighted in Figure 4.1.

In Section 2.3.1, it was argued why the transfer function of the sensor block was set to be equal to 1, i.e. $H(s) = 1$. Even though this is the case, the sensors used in the system still need to be described, so it is known which parameters of the system are measurable. This is also important to investigate since the way the data is obtained might influence the implementation of the controller.

The segway has four sensors that can be used to measure the speed, angular velocity and angle of the segway. To measure the speed, two encoders are available to measure the rotational velocity for both motors. The angle and angular velocity can be found from the gyroscope and the accelerometer. The structure of the following sections is to first describe the interface between the sensor hardware and the microcontroller (MCU) ATxmega128A3U and afterwards explain the implementation.

4.1 Accelerometer and Gyroscope

From section 2.2 it is known that the accelerometer and gyroscope are part of the MPU6050 transducer. This sensor has six degrees of motion, since both the accelerometer and gyroscope are 3-axis sensors. The accelerometer and gyroscope are used to derive the angle and angular velocity of the segway. To acquire the measured data from the sensors, the sensors have an inter-integrated circuit (I^2C) bus, which can be interfaced to from the MCU.

4.1.1 I²C Data

The physical interface consists of two wires Serial Data (SDA) and Serial Clock (SCL). In normal mode the bus supports clock speeds up to 400 kHz, but additions can be made so the speed can go up to 5 MHz [?]. In idle the voltage on the data line is pulled high. On a data level, a transmission consists of a start sequence, 7 bit address, R/W bit, acknowledge, followed by a start sequence, 8 data bit, acknowledge and a stop sequence. This can also be seen in Figure 4.2. Note that it is possible to transmit multiple data bytes without having to specify the address between each data transfer.

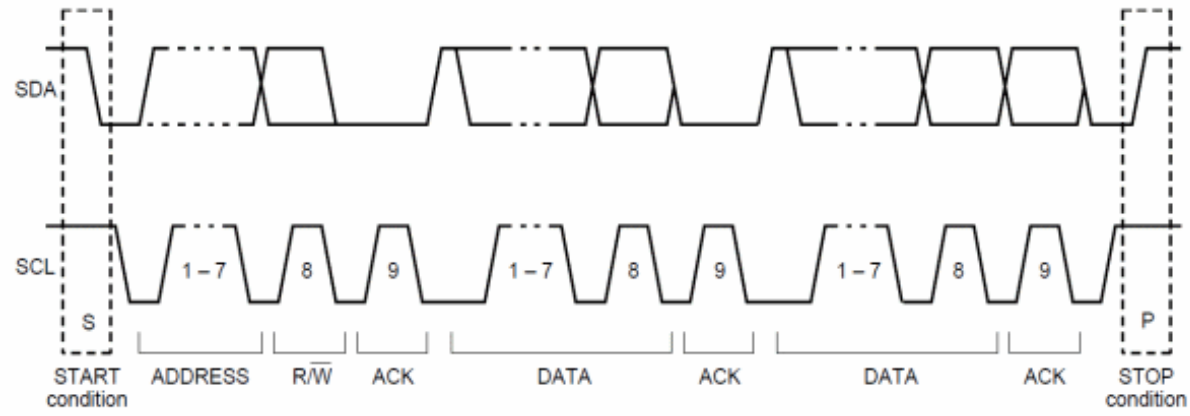


Figure 4.2: A visual representaion of a I²C frame [?].

4.1.2 Data Acquisition from Gyroscope and Accelerometer

The I²C communication protocol is implemented in the MCU as a software module, since the pins, connected to pin 0 and 1 on port A, do not support hardware I²C. The implemented I²C driver, has been provided by group 15gr633, from AAU, who previously worked with the segway. From the header file, it can be seen that the library has 11 functions Listing 4.1. From these functions the three functions i2cInit, i2cWrite and i2cRead are the main functions, the rest are subroutines that are called from the main functions.

```
1  void i2cInit();
2  uint8_t i2cWrite(uint8_t slaveAddr, uint8_t regAddr, uint8_t *data, uint8_t length);
3  uint8_t i2cRead(uint8_t slaveAddr, uint8_t regAddr, int8_t *data, uint8_t length);
4
5  void i2cRepeatedStart();
6  void i2cStart();
7  void i2cStop();
8  void i2cTransmit(uint8_t data);
9  uint8_t i2cReceive();
10 uint8_t i2cGetAck();
11 uint8_t i2cSendAck();
12 uint8_t i2cSendNack();
```

Listing 4.1: Pre initialization of I²C functions.

From the datasheet it is known that the MPU6050 sensors I^2C address is 0x68 [**?**, p. 46]. When the sensor is powered on, it is in sleep mode. To wake up the sensor, 0x00 is written to the PWR_MGMT_1(Power Management 1) 8-bit register. Each sensor has six 8-bit registers for measured data. The three axes' values are sampled by three 16-bit ADC and then stored as MSB and LSB, giving six data registers for each sensor. From the datasheet it is known that the accelerometer data is stored in the registers from 0x3B to 0x40, while the gyroscope data is stored in the registers from 0x43 to 0x48.

Note that both the gyroscope and the accelerometer work in polling mode, meaning that the current measurements are not sent automatically to the microcontroller, but will instead have to be requested over I^2C. The sensor modules takes care of obtaining the measurements and storing them in an internal register automatically - it is only the extraction of this data that is done in polling mode.

4.1.3 Data Processing the Gyroscope and Accelerometer

When the data is collected from the registers, they are put together to three 16 bit words. The accelerometer and gyroscope use the same coordinate system, but the orientation of this coordinate system (x, y, z) must be compared to the segway, which can be seen in Figure 4.3.

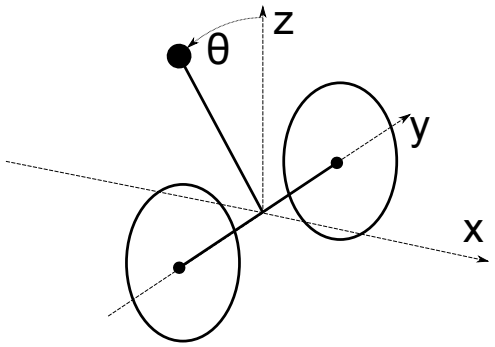


Figure 4.3: The sensors' orientation compared to the segway.

To calculate the tilting angle of the pendulum, θ_p, different methods can be used. For instance, the accelerometer can calculate the angle based on the direction of gravity, or by integrating the gyroscope data, which measures angular velocity. These methods have a couple of problems though. For instance, the accelerometer is very sensitive to changes in movement due to the acceleration applied. The gyroscope however, has a tendency to drift due to the integration [**?**]. To obtain the angle it can therefore be advantageous to combine these measurements. This is done in a complementary filter [**?**], where measurements from both of the sensors are weighted and summed.

From Figure 4.3 it can be deducted that the pendulum angle can be calculated based on the accelerometer as:

$$\theta_{pAx} = -\tan^{-1}\left(\frac{x_{Ax}}{z_{Ax}}\right)$$

Where:

θ_{pAx}	is the angle of pendulum	[rad]
x_{Ax}	is the accelerometer measurement in the x-axis	[1]
z_{Ax}	is the accelerometer measurement in the z-axis	[1]

To get the angular velocity from the gyroscope, it is first recognised, based in Figure 4.3, that it is the data in the y-axis, that is of interest. The angular velocity can then be calculated as follows:

$$\omega_p = \frac{y_{Gyro} \cdot max}{n} \tag{4.1}$$

Where:

ω_p	is the angular velocity of the pendulum	[rad/s]
y_{Gyro}	is the data from the gyroscope in the y-axis	[1]
max	is the gyroscope maximum measurement	[rad/s]
n	is the number of digital representation for the gyroscope data	[1]

Using the complementary filter, the angle can be found as:

$$\theta_p = k \cdot \theta_{pAx} + (1 - k) \cdot \int \omega_p \, \mathrm{dx} \tag{4.2}$$

From Equation 4.2, it is seen how the angle is estimated from both accelerometer and integrated gyroscope data, and then the two values are weighted. The weighting factor k is found experimentally to 0.05.

4.2 Encoder and Quadrature Decoder

To determine the velocity of the wheels at a specified time, the exact change of position over an infinitesimal timespan shall be known, as the velocity $v(t)$ can be expressed by the time derivative of the position $s(t)$:

$$v(t) = \frac{\mathrm{d}s(t)}{\mathrm{d}t} \tag{4.3}$$

Where:

$v(t)$	is a function for velocity	[m/s]
$s(t)$	is a function for position	[m]

To determine a function for the position it is desired to decode the output signal from the encoders since they provide measurements about position changes over time.

The encoders used for the segway are quadrature encoders. The output from a quadrature encoder is two square waves phase shifted by 90° from two channels as illustrated in Figure 4.4.

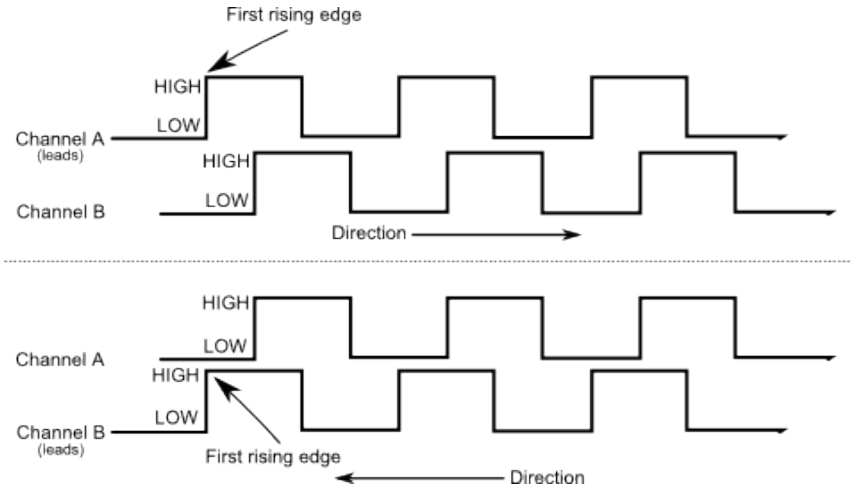


Figure 4.4: Quadrature signal [**?**].

The signals from the encoders provide information about the direction and velocity of the wheel. Since the square waves from channel A and B are 90° out of phase, the direction of the wheels can be found by determining the first rising edge of the signals from channel A and B. As the direction of the wheels change, the first rising edge will occur at the other channel.

The velocity can be determined by the number of of the square waves per time unit. One encoder rotation correspond to 512 high/low cycles or 2048 counts. A count is an event change in either channel A or B. E.g. if channel A goes low it will result in a count. The relation between the position and the number of counts is given by:

$$s(t) = \frac{d \cdot \pi \cdot N_{ms} \cdot N_{sw}}{4 \cdot CPR} \cdot c(t) \qquad \{c(t) \in \mathbb{Z}\} \tag{4.4}$$

Where:

d	is the diameter of the wheel	[m]
N_{ms}	is the encoder/shaft ratio	[1]
N_{sw}	is the shaft/wheel ratio	[1]
$c(t)$	is number of counts. $c(t)$ can only be an integer.	[1]
CPR	number of cycles in one encoder rotation	[1]

The function $c(t)$ changes by the velocity of the wheels. Since $c(t)$ is an integer, the resolution of the position change is limited to the a-coefficient in the equation of $s(t)$, which is a linear expression on the form $s(t) = a \cdot c(t)$. The a-coefficient can be calculated by inserting the values given by the datasheets and measurements, see section A and section B.

$$s(t) = \frac{d\pi \cdot N_{ms} \cdot N_{sw}}{4 \cdot CPR} \cdot c(t) \Rightarrow s(t) = \frac{0.117\,\mathrm{m} \cdot \pi \cdot \frac{1}{19} \cdot \frac{25}{90}}{4 \cdot 512} \cdot c(t) \approx 2.62 \cdot 10^{-6}\,\mathrm{m} \cdot c(t) \tag{4.5}$$

Equation 4.5 yields that 1 count is equal to $2.62 \cdot 10^{-6}$m. The expression of $s(t)$ from Equation 4.5 is inserted into Equation 4.3, yielding:

$$v(t) = 2.62 \cdot 10^{-6}\,\mathrm{m} \cdot \frac{\mathrm{d}c(t)}{\mathrm{d}t} \qquad \{c(t) \in \mathbb{Z}\} \tag{4.6}$$

Since the microcontroller responsible for measuring the encoder only works in discrete time, it is impossible to compute the derivative in Equation 4.6. Instead, a close approximation of Equation 4.6 can be described as:

$$v(t) \approx 2.62 \cdot 10^{-6}\,\mathrm{m} \cdot \frac{c(t) - c(t_1)}{t - t_1} = 2.62 \cdot 10^{-6}\,\mathrm{m} \cdot \frac{\Delta c(t)}{\Delta t} \tag{4.7}$$

Whereas Equation 4.6 gives the exact velocity of the wheels at a given time t, Equation 4.7 gives the approximated velocity in the time span Δt. This means the precision of the velocity is determined by Δt. A smaller time span Δt gives a better precision. If the time span Δt is fixed, meaning Δt is always set to 1 ms, the time span can known as the sampling period T_s. The inverse of T_s is the sampling frequency f_s.

$$v(t) \approx 2.62 \cdot 10^{-6}\,\mathrm{m} \cdot f_s \cdot \Delta c(t) \tag{4.8}$$

To get an accurate velocity, the sampling frequency is set to 1 kHz, but due to rounding in the frequency scaling in the MCU, this frequency cannot be obtained exactly. This small error is

however disregarded.

$$v(t) \approx 2.62 \cdot 10^{-6}\,\mathrm{m} \cdot 1\,\mathrm{kHz} \cdot \Delta c(t) = 2.62 \cdot 10^{-3}\,\frac{\mathrm{m}}{\mathrm{s}} \cdot \Delta c(t) \tag{4.9}$$

From this, the wheel angular velocity can also be found:

$$\omega_w(t) = \frac{2\pi}{r_w} \cdot v(t) = 963 \cdot 10^{-6} \cdot \Delta c(t) \frac{\mathrm{rad}}{\mathrm{s}} \tag{4.10}$$

The next step is to implement Equation 4.9 on the microcontroller.

4.2.1 Quadrature Decoder Setup

To process the data from the encoders, some decoders are needed. As stated previously, every event equals to a logical change from channel A or B from the encoders. Since the segway features two encoders, one encoder is connected to port C and another to port E. To allow event detection on the XMEGA, the registers associated with the quadrature decoding are set. The functions used for the setup of the quadrature decoders are seen in Listing 4.2.

```
1 void qdec_setup();          // Setup registers for the quadrature decoder
2 void en_interrupt();        // Enable interrupt
3 float motor_speed_left();   // Calculate and return the velocity of the left wheel
4 float motor_speed_right();
```

Listing 4.2: Pre-initialization of quadrature decoder functions.

All registers associated with the quadrature decoders are set in the function qdec_setup(), where the hardware for the decoder is set up. The XMEGA has external hardware that supports quadrature decoding. This means that the microprocessor does not directly need to handle the encoder itself, but only needs to read the data from the decoder through an interrupt service routine (ISR). After calling this function, a counter register will increment by one every time an event changes is occurring from channel A and B from both encoders.

4.2.2 Data Processing the Encoder Measurements

As previously stated, the sampling frequency for the decoders are set to $f_s = 1$ kHz. To ensure a sampling every 1 ms, a timer is set to count up 1 ms and interrupts in order to sample the counter from the decoders. The interrupt service routine is seen in Listing 4.3.

```
1 ISR(TCD1_OVF_vect){
2     encoder_ccnt = TCC1.CNT;        // Get the value from port C counter register
3     encoder_ecnt = TCE1.CNT;        // -||- port E
4     TCE1.CNT = 0;                   // Reset the counter register
5     TCC1.CNT = 0;
6 }
```

Listing 4.3: Interrupt Service Routine (ISR) for sampling the counters.

If the velocity of the wheels is required, the functions motor_speed_left() and motor_speed_right() are called. The functions then return the speed of the left and right wheel. This is done by converting the encoder counter to velocity using Equation 4.9.

4.3 PWM and H-bridge

To control the DC-motors connected to the wheels, two H-bridges are used. By using H-bridges it is possible to control both the direction and the speed of the DC motors. For each H-bridge, the pins D2, IN1 and IN2 on the H-bridge are used for this purpose. By feeding D2 with a PWM signal, the velocity is changed by changing the duty cycle. A duty cycle of 100% results in the motors going on full speed, whereas a duty cycle of 0% results in no rotation. To determine the direction, either IN1 or IN2 should be pulled high.

The H-bridges' IN1 and IN2 pins are connected to the microcontroller's pin PC2 and PC3, and PE2 and PE3. The D2 pin is connected to PC1 and PE1 - these pins are set to be the PWM generating outputs in the MCU. The functions associated with PWM generation can be seen in Listing 4.4.

```
1 void PWM_setup();
2 void PWM_left(int duty);
3 void PWM_right(int duty);
```

Listing 4.4: Pre initialization of PWM generation functions.

When generating a PWM signal, the resolution of the PWM signals are set to 256 counts, which is considered to be sufficient.
As the sensor block in the feedback loop, see Figure 4.1, is now known, the system model of the segway is to be derived in the following chapter.

5 | Modelling

To make the segway able to balance in an upright position and be able to drive as well, it is necessary to have a model of the system, to understand how the system behaves from a physical and mathematical perspective. In this chapter, this model will be derived. A model is a mathematical description of the system's behaviour, in this case both the electrical and mechanical parts of the system. This is needed to design a controller for the system.

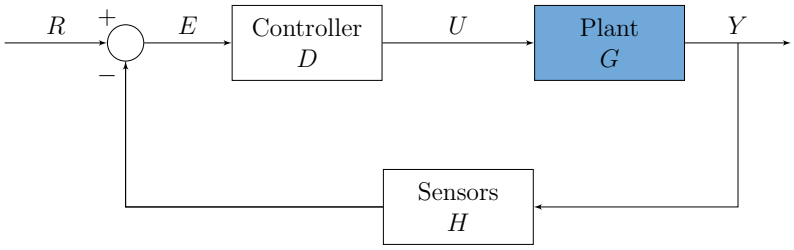


Figure 5.1: Feedback loop of a system, with the plant, G, highlighted.

Looking at a general feedback loop, see Figure 5.1, it consists of three blocks. A controller, D, a plant that to be controlled, G, and sensors, H. The control loop also features a reference signal, R, an error signal, E, the control signal, U, between the controller and the plant, and an output, Y. It is the model of the plant, that is to be determined in this chapter, as highlighted in Figure 5.1.

5.1 Modelling overview

The segway can be described as an inverted pendulum, the cart can be moved by a motorized two-wheeled system in order to stabilize the segway in an upright position.

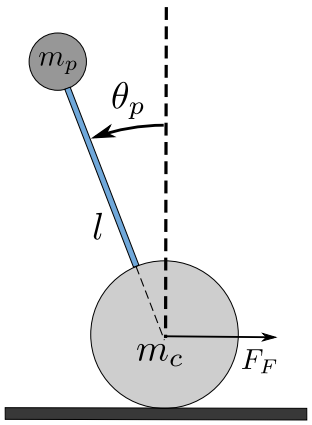


Figure 5.2: An overall diagram of the segway.

A figure of the segway with the cart mass, m_c, and inverted pendulum mass, m_p, can be seen in Figure 5.2. Here, the inverted pendulum's tilting angle is represented by θ_p, it can be further noted that the wheel's angle is denoted θ_w.

The model of the segway is split in two smaller models, which is a model for the motor and wheel, and a model for the cart and inverted pendulum. These two models are later combined into a model for the segway. The splitting of the model can be seen in Figure 5.3, together with the interfaces between the models.

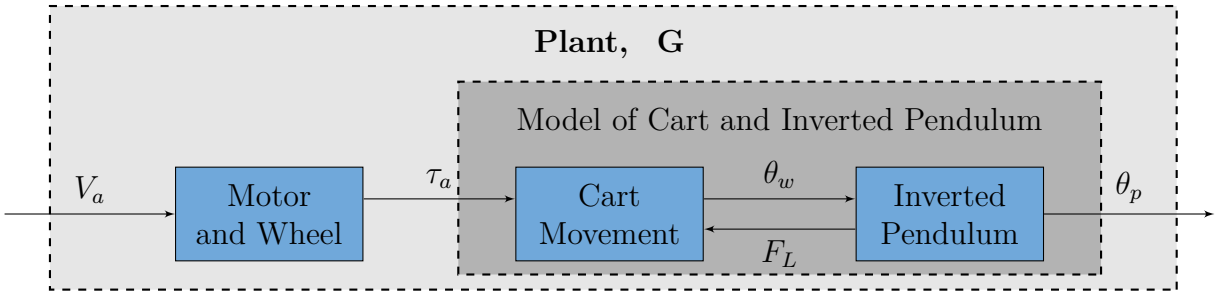


Figure 5.3: The plant in the feedback loop. Within it is a motor and wheel model and a model for the inverted pendulum.

The input to the system is the voltage applied to the motors, V_a. This voltage is controlled by simply changing the PWM duty cycle that controls the motors. In practice, this is equal to changing the motor input voltage. The interface signal between the motor and wheel model and the model describing the cart's movement is the torque applied by the motors to the cart, $\tau_a(t)$. Based on this torque, an expression for the force $F_F(t)$ can be made, which is the force that makes the segway move either forward or backwards. The interface between the cart and the inverted pendulum is chosen to be the the angle of the wheels, θ_w, as they determine the position of the segway. The model of the inverted pendulum has this angle, θ_w as input and the angle of the inverted pendulum, θ_p, as output. Also, a load force $F_L(t)$ is inserted from the inverted pendulum to the cart model, as the tilting of the pendulum makes the cart move.

In this chapter, an expression for the motor and wheel model will be derived. Afterwards, a model of the cart and inverted pendulum is derived, see Figure 5.3. These models will be simulated and verified, before being combined. Lastly the system model is simulated and verified. To be able to design a controller for the system, this model is then linearised and Laplace transformed to allow a transfer function to be derived, which is the final step performed in this chapter. Firstly the motors and wheels model is to be derived in the following section.

5.2 Model of Motor and Wheel

The model of the motors and wheels can be split into two parts; an electrical part and a mechanical part, since the two parts can be analyzed separately, though they are connected. This is due to the feedback that is present between the rotational velocity and the voltage, when modelling a motor [**?**, 15].
The electrical model gives an output describing the motor current, $I_a(t)$. This is linked to the motor torque through the motor constant, k_t [**?**, p. 14].
The mechanical part of the motor model describes both the motors, gears and wheels, since these are all connected. The mechanical part of the model has the angular velocity of the wheel,

ω_w as output. A feedback is present in the system from ω_w to the applied voltage v_a, due to the back-EMF in the motor.
In the following, the electrical motor model is derived. Then, a mechanical model of the motors and wheels is put up. These two models can then be combined.
Note that the modelling will be based on a single motor and wheel system, as the two motors and wheels on the segway are assumed identical. However, when deriving the expression for the applied force, F_F, both motors and wheels in the system are accounted for.

5.2.1 Electrical Motor Model

The electrical part of a motor can be modelled using a resistor, R_a, an inductor, L_a, and a voltage generator, V_e, which delivers a voltage proportional to the motor's angular velocity ω_m, also known as the back-EMF voltage [**?**, p. 15]. This can be seen in Figure 5.4, where the input voltage, V_a, and the motor current, I_a, are shown, with the index a denoting the armature of the motor. This model assumes that it is a permanent magnet DC motor that is used [**?**, p. 15], but as explained in section 2.2.1, the motors used are of this type, meaning the model structure is valid.

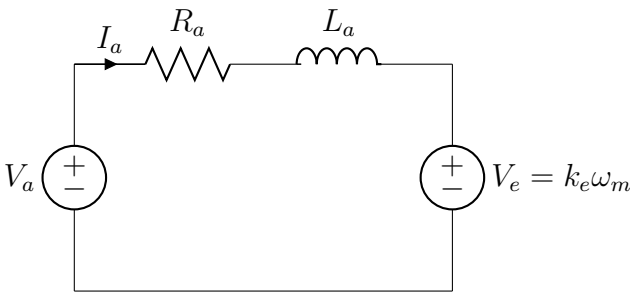


Figure 5.4: Electrical curcuit equivalent of a DC motor.

Kirchoff's voltage law is applied to the circuit yielding the following expression:

$$0 = V_a(t) - R_a \cdot I_a(t) - L_a\dot{I}_a(t) - V_e(t) \tag{5.1}$$

Where:

$V_a(t)$	is the motor input voltage	[V]
R_a	is the motor resistance	[Ω]
L_a	is motor inductance	[H]
$I_a(t)$	is motor current	[A]
$V_e(t)$	is the back-EMF	[V]

The back-EMF is directly proportional to the rotational speed of the motor:

$$V_e(t) = k_e \cdot \omega_m(t) \tag{5.2}$$

Where:

$\omega_m(t)$	is the rotational speed of the motor	[rad/s]
k_e	is the back-EMF constant	$\left[\mathrm{V}/\frac{\mathrm{rad}}{s}\right]$

Now, the expression for $V_e(t)$ can be inserted in Equation 5.1. The equation is isolated for $I_a(t)$, since the current is the interface between the electrical and mechanical parts of the model. It is assumed that $\dot{I}_a(t) = 0$, since the electrical time constant is usually much higher than mechanical time constant [**?**, p. 74]. This yields Equation 5.3.

$$I_a(t) = \frac{1}{R_a}\left(V_a(t) - k_e \cdot \omega_m(t)\right) \tag{5.3}$$

The torque of a motor is proportional to the current [**?**, p. 14]:

$$\tau_m(t) = k_t \cdot I_a(t) \tag{5.4}$$

Where:

$\tau_m(t)$	is the torque of the motor	[Nm]
k_t	is the motor constant	[Nm/A]

Inserting the expression for the current $I_a(t)$ from Equation 5.3 into Equation 5.4 gives the following:

$$\tau_m(t) = \frac{k_t}{R_a}\left(V_a(t) - k_e\omega_m(t)\right) \tag{5.5}$$

A model for the gears and wheel needs to be determined to find the torque τ_a that is applied to the segway, since the inertias and dampers in the gear-wheel setup influences this. This is done in the following section.

5.2.2 Mechanical Motor and Wheel Model

A model for the mechanical behaviour of the motors and wheels is to be found, so it can be combined with the electrical model for the motors, thus forming the combined model for the motors and wheels.
It is necessary to take the entire mechanical system, eg. both wheels and gearing ratios, into account, as these affect the motor's rotation.

Gear relations

First, the configuration of the gears are determined before the modelling can begin. The motor and wheel system consists of a DC motor, which has a gear mounted. The output of this gear

is applied to the motor shaft, onto which a secondary gear is mounted. The inner wheel gear is mounted directly on the shaft, and thus has the same angular velocity as the motor. Therefore their inertia can be seen as one, being the shaft inertia. The configuration of the gears can be seen in Figure 5.5.

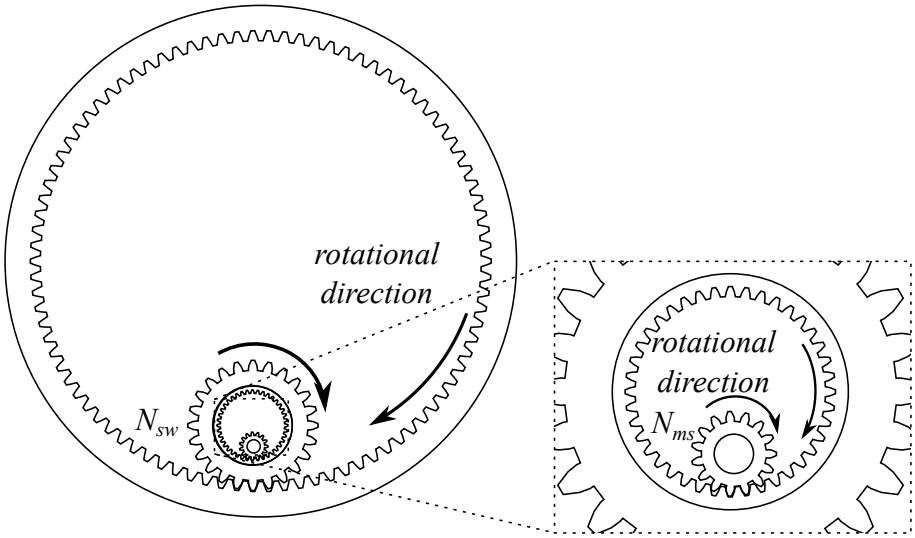


Figure 5.5: The different parts of the wheel's rotational system and its relative direction of rotation along with the gear ratios, N_{ms} and N_{sw}.

In Figure 5.5 the gear ratio between the motor and shaft is shown as N_{ms}, and the gear ratio for the shaft and wheel as N_{sw}.

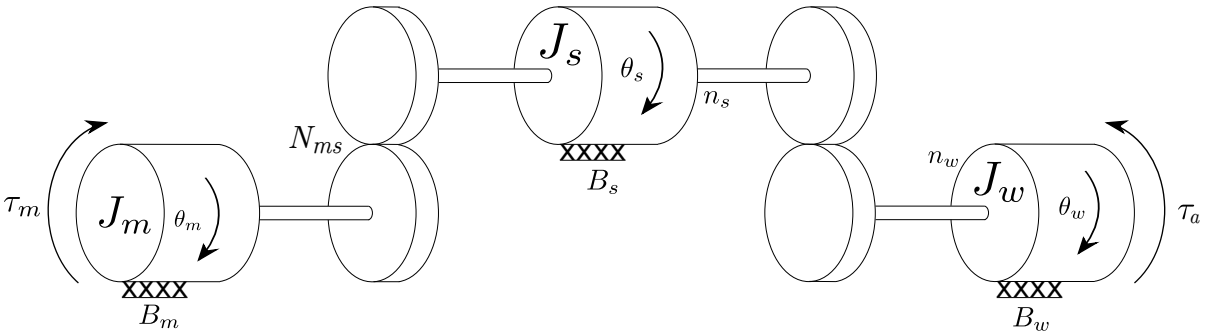


Figure 5.6: Mechanical model of the motor and wheel setup.

The gear ratio is used to determine the relation between angle and its derivatives for two or more rotational objects. The basis for determining is Figure 5.7, where it is assumed that there is no sliding between two gears. A torque τ_1 is applied to the inner gear, giving it a certain angular position, θ_1. The second gear also has a torque, τ_2 and an angular position, τ_2. Since the two gears have to move the same amount when rotating, as no slipping is assumed, Equation 5.6 can be put up, from which the gear ratio can be determined.

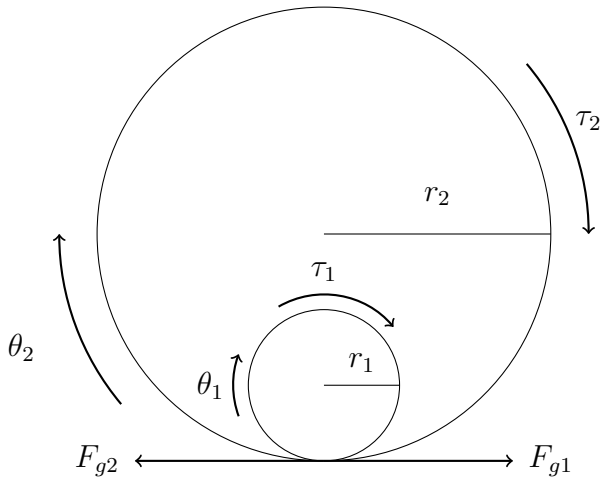


Figure 5.7: Correlation of gears.

$$\theta_1 r_1 = \theta_2 r_2 \Rightarrow N = \frac{r_1}{r_2} = \frac{\theta_2}{\theta_1} = \frac{\omega_2}{\omega_1} \tag{5.6}$$

Where:

N	The gear ratio	[1]
θ_1	Angle of the inner gear	[rad]
θ_2	Angle of the outer gear	[rad]
r_1	The radius of the inner gear	[m]
r_2	The radius of the outer gear	[m]
ω_1	Angle velocity of the inner gear	[rad/s]
ω_2	Angle velocity of the outer gear	[rad/s]

This, however, does not yield the relation between the torque of the gears and the gear ratio. This relation can be obtained through steady state analysis of each gear. This yields Equation 5.7 and Equation 5.8 for the inner and outer gear respectively, since the force of the two gears are of the same magnitude, i.e. $F_{g1} = F_{g2} = F_g$. This is because the force one gear excerts on the other is balanced by an equal, but opposite force, in correlation with Newton's 3rd law of motion. Based on this, it is possible to put up the following equations:

$$\tau_1 - r_1 \cdot F_g = 0 \tag{5.7}$$
$$\tau_2 - r_2 \cdot F_g = 0 \tag{5.8}$$

Where:

F_g	Translatoric force that works against the gear.	[N]

Thus, the gear ratio affects the relation between the torque in the two gears as shown in Equation 5.9.

$$\frac{\tau_1}{\tau_2} = \frac{r_1}{r_2} = N \tag{5.9}$$

To describe the relation of the torque and the angle for two or more gears, the gear ratio, N, must be determined. This is done through the number of cogs in each cogwheel, as this can be thought of being equivalent to the wheel radii, which has previously been used to determine the gear ratio. From subsubsection 2.2.1, it's known that $n_s = 25$ and $n_w = 90$. From these, the gear ratio can be found.

$$N_{sw} = \frac{n_s}{n_w} = \frac{\theta_w(t)}{\theta_s(t)} = \frac{25}{90} \approx 0.2778 \tag{5.10}$$

Where:

N_{sw}	is the shaft-wheel gear ratio	[1]
n_s	is number of cogs on the motor gear	[1]
n_w	is number of cogs on the wheel	[1]
$\theta_s(t)$	is the motor angle	[rad]
$\theta_w(t)$	is the wheel angle	[rad]

From [**?**] the gear ratio from motor to shaft is known:

$$N_{ms} = \frac{1}{19} \approx 0.053 \tag{5.11}$$

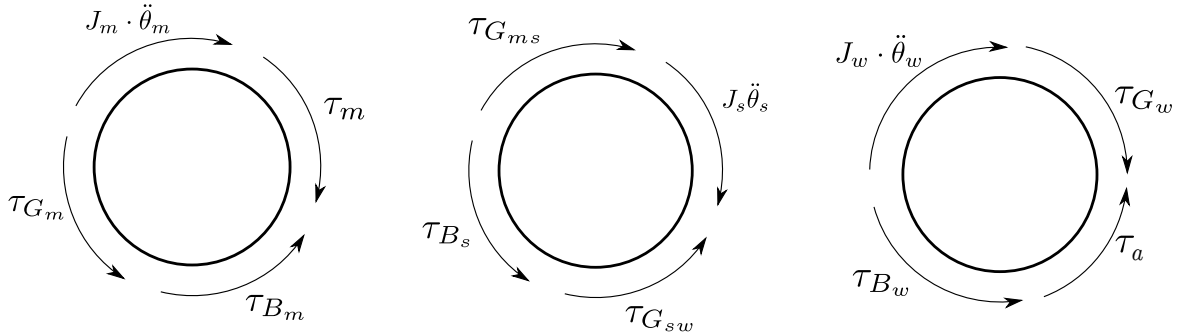
With the gear relations determined, it is now possible to derive the mechanical motor and wheel model.

Derivation of Mechanical Motor and Wheel Model

Now that the configuration of the rotational system is known, the motor, shaft and wheel system can be drawn using ideal elements, i.e. inertias and dampers, which can be seen in Figure 5.6. Note, that all gears are assumed to have an inertia and a friction as no gears are assumed ideal. Also note that the friction in the gears is modelled as a damper, and thus each of the friction torques can be described in the same way as the friction torque for the shaft shown here: $\tau_{B_s}(t) = B_s \cdot \omega_s(t)$. The coulomb friction which is also present is not taken into account, due to the fact that an offset in the software can take care of this non-linearity, which is not easily modelled.

In Figure 5.6 it is seen how the motor connects to the shaft, which again connects to the wheel. All connections are made through gears. The constants n_s and n_w, are the number of cogs on the shaft and wheel cog-wheels. These are used to determine the gear ratios.

The input torque to the motor $\tau_m(t)$, see Figure 5.6, sets the direction of all the resultant angular positions, and therefore also angular velocities and angular accelerations in accordance with Figure 5.5.
The free body diagrams of the motor, shaft and wheel are illustrated in Figure 5.8.



(a) FBD of rotational system for the motor.

(b) FBD of rotational system for the shaft.

(c) FBD of rotational system for the wheel.

Figure 5.8: Free body diagrams of the three rotational systems.

Note that the following relationships between the gear torques exist, based on the gear ratios:

$$\tau_{G_{ms}}(t) = \tau_{G_m}(t) \cdot \frac{1}{N_{ms}} \tag{5.12}$$

$$\tau_{G_w}(t) = \tau_{G_{sw}}(t) \cdot \frac{1}{N_{sw}} \tag{5.13}$$

Where:

$\tau_{G_{ms}}(t)$	is the torque from motor to shaft	[Nm]
$\tau_{G_m}(t)$	is the torque to shaft from motor	[Nm]
$\tau_{G_w}(t)$	is the torque from shaft to wheel	[Nm]
$\tau_{G_{sw}}(t)$	is the torque to wheel from shaft	[Nm]

From Figure 5.8, the following differential equations can be put up:

$$J_m\ddot{\theta}_m(t) = \tau_m(t) - \tau_{G_m}(t) - \tau_{B_m}(t) \tag{5.14}$$

$$J_s\ddot{\theta}_s(t) = \tau_{G_{ms}}(t) - \tau_{G_{sw}}(t) - \tau_{B_s}(t) \tag{5.15}$$

$$J_w\ddot{\theta}_w(t) = \tau_{G_w}(t) - \tau_{B_w}(t) - \tau_a(t) \tag{5.16}$$

Where:

J_m	is the motor moment of inertia	$[\mathrm{kg} \cdot \mathrm{m}^2]$
J_s	is the shaft moment of inertia	$[\mathrm{kg} \cdot \mathrm{m}^2]$
J_w	is the wheel moment of inertia	$[\mathrm{kg} \cdot \mathrm{m}^2]$
$\ddot{\theta}_m(t)$	is the angular acceleration of the motor	$[\mathrm{rad/s}^2]$
$\ddot{\theta}_s(t)$	is the angular acceleration of the shaft	$[\mathrm{rad/s}^2]$
$\ddot{\theta}_w(t)$	is the angular acceleration of the wheel	$[\mathrm{rad/s}^2]$
$\tau_m(t)$	is the delivered torque from the motor	$[\mathrm{rad/s}^2]$
$\tau_{B_m}(t)$	is the motor friction torque	[Nm]
$\tau_{B_s}(t)$	is the shaft friction torque	[Nm]
$\tau_{B_w}(t)$	is the wheel friction torque	[Nm]
$\tau_a(t)$	is the torque applied to the cart	[Nm]

In the following, a model for the motors and wheels system is obtained. This is done using the differential equations derived from the free body diagrams, see Equation 5.14, Equation 5.15 and Equation 5.16.
The model is derived by working from the outside and in, starting with the equations for the wheel and inserting them to get an expression for the motor that includes the wheel and shaft. The differential equation for the wheel rotational system, Equation 5.16, states:

$$J_w\ddot{\theta}_w(t) = \tau_{G_w}(t) - \tau_{B_w}(t) - \tau_a(t)$$

The linear approximation of the friction torque, which is modelled as a damper, can be described as $\tau_{B_s}(t) = B_s \cdot \dot{\theta}_s(t)$. Inserting this together with the expression for the gear torque from Equation 5.13 gives:

$$J_w\ddot{\theta}_s(t)N_{sw} = \tau_{G_{sw}}(t) \cdot \frac{1}{N_{sw}} - B_wN_{sw} \cdot \dot{\theta}_s(t) - \tau_a(t) \tag{5.17}$$

Isolating for $\tau_{G_{sw}}$ gives:

$$\tau_{G_{sw}}(t) = J_w\ddot{\theta}_s(t)N_{sw}^2 + B_wN_{sw}^2\dot{\theta}_s(t) + \tau_a(t) \cdot N_{sw} \tag{5.18}$$

Now that an expression for $\tau_{G_{sw}}(t)$ is obtained, it can be combined with the differential equation for the second rotational system, Equation 5.16, which states:

$$J_s\ddot{\theta}_s(t) = \tau_{G_{ms}}(t) - \tau_{G_{sw}}(t) - \tau_{B_s}(t)$$

The friction can, as previously mention be described as $\tau_{B_s}(t) = B_s \cdot \dot{\theta}_s(t)$. Inserting this and isolating for $\tau_{G_{ms}}(t)$ yields:

$$\tau_{G_{ms}}(t) = \tau_{G_{sw}}(t) + B_s \cdot \dot{\theta}_s(t) + J_s\ddot{\theta}_s(t) \tag{5.19}$$

Equation 5.18 is inserted in Equation 5.19 together with the expression for the motor-shaft gear from Equation 5.12. An expression for $\tau_{G_m}(t)$ can thus be found:

$$\tau_{G_m}(t) = J_w\ddot{\theta}_m(t)N_{ms}^2N_{sw}^2 + B_wN_{ms}^2N_{sw}^2\dot{\theta}_m(t) + \tau_a(t)\cdot N_{ms}N_{sw} + B_sN_{ms}^2\cdot\dot{\theta}_m(t) + J_sN_{ms}^2\ddot{\theta}_m(t) \tag{5.20}$$

This expression can be combined with the differential equation for the motor rotational system, as seen in Equation 5.14 and stated here:

$$J_m\ddot{\theta}_m(t) = \tau_m(t) - \tau_{G_m}(t) - \tau_{B_m}(t)$$

Inserting the expression for the friction torque, $\tau_{B_m}(t) = B_m\cdot\dot{\theta}_m(t)$ and $\tau_{G_m}(t)$ as found in Equation 5.20 yields Equation 5.21 where the motor torque $\tau_m(s)$ has also been isolated:

$$\begin{aligned}\tau_m(t) =& J_m\ddot{\theta}_m(t) + B_m\cdot\dot{\theta}_m(t) + (J_w\ddot{\theta}_m(t)N_{ms}^2N_{sw}^2 + \\ & B_wN_{ms}^2N_{sw}^2\dot{\theta}_m(t) + \tau_a(t)\cdot N_{ms}N_{sw} + B_sN_{ms}^2\cdot\dot{\theta}_m(t) + J_sN_{ms}^2\ddot{\theta}_m(t))\end{aligned} \tag{5.21}$$

This simplifies to:

$$\tau_m(t) = \ddot{\theta}_m(t)\left(J_m + N_{ms}^2(J_s + J_wN_{sw}^2)\right) + \dot{\theta}_m(t)\left(B_m + N_{ms}^2(B_s + N_{sw}^2B_w)\right) + \tau_a(t)\cdot N_{ms}N_{sw} \tag{5.22}$$

To simplify the expressions, the following definitions are made for the total equivalent inertia, J_T and the total equivalent friction, B_T:

$$J_T \equiv J_m + N_{ms}^2(J_s + N_{sw}^2J_w) \tag{5.23}$$

$$B_T \equiv B_m + N_{ms}^2(B_s + N_{sw}^2B_w) \tag{5.24}$$

This means that the expression can be written as:

$$\tau_m(t) = \ddot{\theta}_m(t)\cdot J_T + \dot{\theta}_m(t)\cdot B_T + \tau_a(t)\cdot N_{ms}N_{sw} \tag{5.25}$$

Using this and inserting the motor torque, Equation 5.5, and isolating for $\tau_a(t)$ yields:

$$\tau_a(t) = \frac{1}{N_{ms}N_{sw}}\left(\frac{k_t}{R_A}\left(V_a(t) - k_e\dot{\theta}_m(t)\right) - \ddot{\theta}_m(t)J_T - \dot{\theta}_m(t)B_T\right) \tag{5.26}$$

Equation 5.26 is the final model expression for the motors and wheels part of the system. Now the model of the inverted pendulum is to be found. These model expressions are then to be combined. From this a controller can be designed and later implemented. The deriviation of the inverted pendulum model is done in the following section.

5.3 Model of Cart and Inverted Pendulum

In this section, a model of the inverted pendulum is derived. This model describes the movement of the cart and the inverted pendulum, based on governing equations put up regarding the behaviour of the system. To do this, it is necessary to investigate how the inverted pendulum moves, and how the interaction between the inverted pendulum and the cart can be described. Before this is done in detail, it is however first necessary to obtain an overview of the inverted pendulum.

5.3.1 Overview of the Inverted Pendulum

The model of the inverted pendulum describes how the two masses, i.e. the cart and the inverted pendulum, behaves when a force is exerted upon it by the surroundings. Note that the rod connecting the inverted pendulum mass to the cart is assumed massless, and is therefore not a part of the model. A schematic drawing of the inverted pendulum is shown in Figure 5.9.

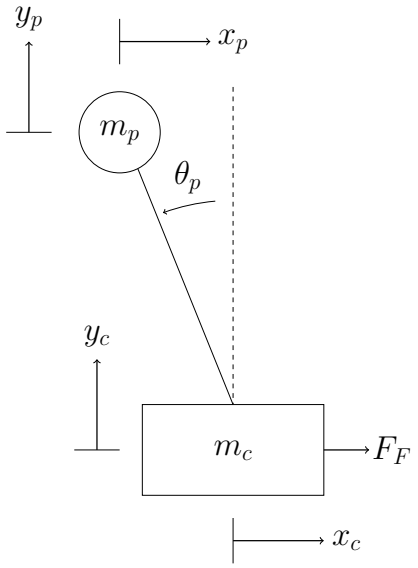


Figure 5.9: Mechanical model of the inverted pendulum.

In Figure 5.9 the mass at the end of the rod is m_p, the variable $\theta_p(t)$ describes the angle of the inverted pendulum from a vertical line at the center of the cart. Note that the angle is positive in the counter-clockwise direction. This angle can be described in relation to both the x- and y-direction. The force applied to move the segway is labelled $F_F(t)$.

For simplicity, the coordinate system is defined such that $y_c(t)$ is 0, thus:

$$x_p(t) = x_c(t) - l \cdot \sin(\theta_p(t)) \tag{5.27}$$

$$y_p(t) = l \cdot \cos(\theta_p(t)) \tag{5.28}$$

Where:

$x_p(t)$	is the x-position of the inv. pendulum	[m]
$x_c(t)$	is the x-position of the cart	[m]
l	is the length between the center of rotation of the cart and the center of mass of the inv. pendulum	[m]
$\theta_p(t)$	is the angle of the inv. pendulum	[rad]
$y_p(t)$	is the y-position of the inv. pendulum	[m]

From this, the accelerations can be found by taking the double time differential of both sides of the above equations:

$$\ddot{x}_p(t) = \ddot{x}_c(t) + l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t) \tag{5.29}$$

$$\ddot{y}_p(t) = -l \cdot \cos(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \sin(\theta_p(t)) \cdot \ddot{\theta}_p(t) \tag{5.30}$$

Where:

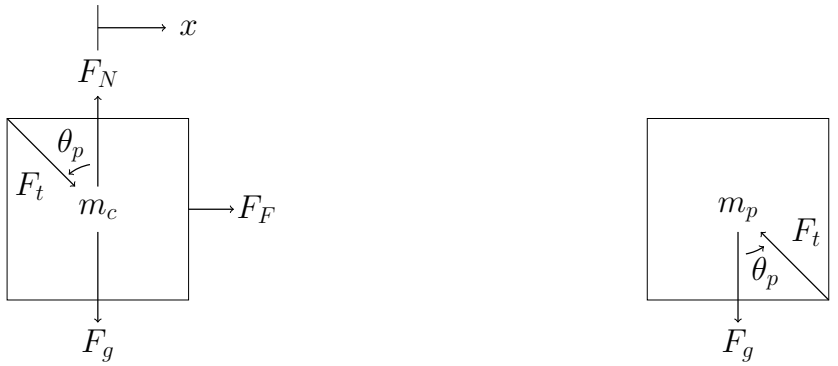
$\ddot{x}_p(t)$	is inv. pendulum's acceleration in the x-direction	$[\text{rad/s}^2]$
$\ddot{\theta}_p(t)$	is inv. pendulum's angular acceleration	$[\text{rad/s}^2]$
$\dot{\theta}_p(t)$	is inv. pendulum's angular velocity	[rad/s]
$\ddot{y}_p(t)$	is inv. pendulum's acceleration in the y-direction	$[\text{rad/s}^2]$

To derive a model expression for the inverted pendulum, which shall be combined with a model expression for the motors and wheels, an equation describing the change in $\theta_p(t)$ caused by any $F_F(t)$ must first be derived. This is done in the following section.

5.3.2 Equations of Motion

In this section, translatoric free body diagrams for the cart and the inverted pendulum are presented and used to derive equations of motion. From these, a model of the cart and the inverted pendulum can be obtained.

From Figure 5.9, free body diagrams can be determined. In this case, the cart's and the inverted pendulum's free body diagrams are representing the forces acting upon them while the segway is at an angle, $\theta_p(t)$.
The free body diagram of the mass at the end of the rod, is shown in Figure 5.10b.



(a) FBD of the cart. **(b)** FBD of the pendulum mass.

Figure 5.10: Free body diagrams of the mass of the inverted pendulum and the mass of the cart.

The forces acting on the the mass m_p is the gravitational force F_g, and the tension in the rod between the cart and the inverted pendulum, $F_t(\theta_p(t))$. This force is split up into two composants - a composant in the x-direction, $F_{tx}(\theta_p(t))$ and one in the y-direction, $F_{ty}(\theta_p(t))$. Figure 5.10a shows the free body diagram of the segway's cart. The force F_g is the gravitational force and F_N is the force pushing upwards from the ground. Note that the magnitude of the tension force $F_t(\theta_p(t))$ is equal to the force acting upon the inverted pendulum, but in opposite direction. From the free body diagrams, it is now possible to express the dynamics of the inverted pendulum with the following three equations of motion.

For the inverted pendulum:

$$m_p \cdot \ddot{x}_p(t) = -F_{tx}(\theta_p(t)) \tag{5.31}$$

$$m_p \cdot \ddot{y}_p(t) = F_{ty}(\theta_p(t)) - F_g \tag{5.32}$$

Where:

m_p	is the mass of the inv. pendulum	[kg]
$\ddot{x}_p(t)$	is inv. pendulum's acceleration in the x-direction	[m/s^2]
$F_{ty}(\theta_p(t))$	is the rod's tension force in the y-direction	[N]
$F_{tx}(\theta_p(t))$	is the rod's tension force in the x-direction	[N]
$\theta_p(t)$	is inv. pendulum's angle from cart's center	[rad]
$\ddot{y}_p(t)$	is inv. pendulum's acceleration in the y-direction	[m/s^2]
F_g	is the gravitational force	[N]

Note that Equation 5.31 is for the horizontal direction and Equation 5.32 is for the vertical direction. For the cart, it is only relevant to sum the forces in the horizontal direction, as it is assumed that the surface the cart moves on is immovable.

$$m_c \cdot \ddot{x}_c(t) = F_{tx}(\theta_p(t)) + F_F(t) \tag{5.33}$$

Where:

m_c	is the mass of the cart	[kg]
$\ddot{x}_c(t)$	is the cart's acceleration in the x-direction	[m/s^2]
$F_F(t)$	is the applied force from the motors	[N]

From equation Equation 5.33 it can be deducted that the load force F_L is equivalent to F_{tx} since this is the contribution from the inverted pendulum. By combining Equation 5.30, 5.31 and 5.33 the governing equation for cart's movement can be obtained.

$$m_c \cdot \ddot{x}_c(t) = F_F(t) - m_p \left(\ddot{x}_c(t) + l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t) \right) \tag{5.34}$$

From this, it can be seen that the load force can be described as:

$$F_L = m_p \left(\ddot{x}_c(t) + l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t) \right) \tag{5.35}$$

Now, the translatoric movement of the inverted pendulum has been described, and the rotation thus needs to be described. The rotation of the inverted pendulum is described from the resultant torque acting on the inverted pendulum, denoted as $J_p\ddot{\theta_p}(t)$. The rotation of the inverted pendulum occurs around the center of mass of the inverted pendulum. From Figure 5.11, it can be seen that there are three forces determining the rotation of the inverted pendulum, namely the gravitational force, F_g, and the x- and y-composants of the tension force, namely F_{tx} and F_{ty}.

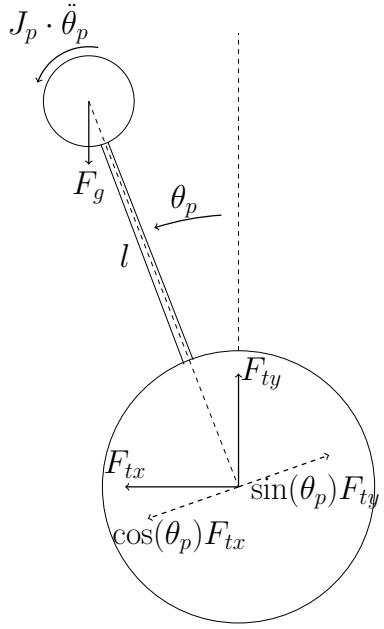


Figure 5.11: The forces contributing to the resultant torque of the inverted pendulum.

The mentioned forces exert a torque on the inverted pendulum based on their distance from the center of rotation. In the case of the segway, the arm is the length of the rod, where the forces acting upon it are F_{tx} and F_{ty}. It can be seen that F_g attacks at the point mass, resulting in the arm length being zero and is therefore not acting upon the inverted pendulum. It should be noted that it is only the composant of the forces which are perpendicular to the rod that contributes to the torque, which needs to be included in the torque expressions. Thus, summing over the torques, the resultant pendulum torque can be described as:

$$J_p\ddot{\theta_p}(t) = l \cdot \sin(\theta_p)F_{ty} - l \cdot \cos(\theta_p)F_{tx} \tag{5.36}$$

Where:

J_p	is the inertia of the inverted pendulum	[kg m^2]

A governing equation for the inverted pendulum can be obtained by inserting Equation 5.29 in Equation 5.31 and Equation 5.30 in Equation 5.32 and inserting these expressions in Equation 5.36:

$$\begin{aligned} J_p\ddot{\theta_p}(t) = m_p \cdot l \cdot \sin(\theta_p(t)) \left(g - l \cdot \cos(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \sin(\theta_p(t)) \cdot \ddot{\theta}_p(t)\right) \\ + m_p \cdot l \cdot \cos(\theta_p(t)) \left(\ddot{x}_c(t) - l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t)\right) \end{aligned} \tag{5.37}$$

Expanding this and using simple geometric relations, the expression can be simplified to:

$$(J_p + m_p \cdot l^2) \cdot \ddot{\theta}_p(t) = m_p \cdot l \cdot (\sin(\theta_p(t)) \cdot g + \cos(\theta_p(t)) \cdot \ddot{x}_c(t)) \tag{5.38}$$

Equation 5.34 and Equation 5.38 are the final model expressions for the inverted pendulum. The next step before transfer functions are derived, is to linearize these expressions. This is done in the following section.

5.4 Derivation of Segway Transfer Functions

In this section, the model expressions derived in previous sections are verified, linearized and combined into transfer functions. From these transfer functions, the controllers will be designed to allow the segway to balance in an upright position. In this section, the models are viewed and linked as illustrated in Figure 5.12.

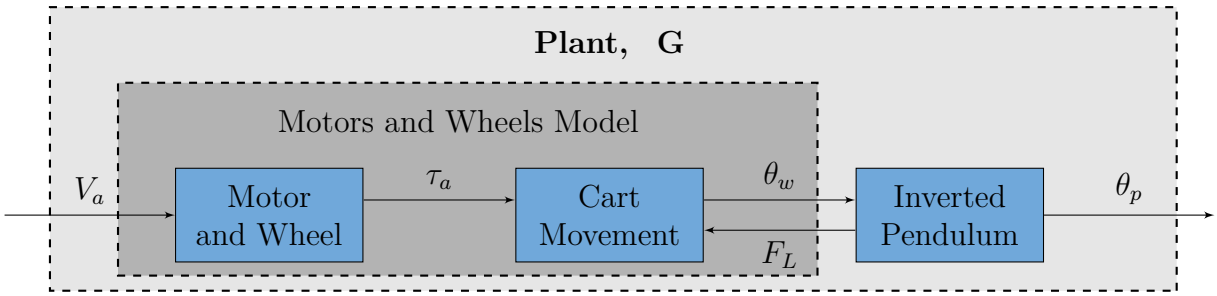


Figure 5.12: Block diagram of the plant, also known as system model.

Note that the models are linked differently than previously shown in Figure 5.3, since the cart movement is made part of the motor and wheel model, instead of being part of the inverted pendulum model. This is because it is not possible to measure the torque τ_a, while the wheel angle, θ_w, can be measured by the encoders. Thus, it will be possible to derive a transfer function for the motors and wheels model. The same applies for the inverted pendulum model, where it will be possible to derive and verify a transfer function based on an input being the wheel angle, θ_w, instead of the torque τ_a.

5.4.1 Verification of Models

The three model expressions that have been derived in the previous sections are Equation 5.26, 5.34 and 5.38. For repetition, these are shown below.

$$\tau_a(t) = \frac{1}{N_{ms}N_{sw}}\Big(\frac{k_t}{R_a}(V_a(t) - k_e\dot{\theta}_m(t)) - \ddot{\theta}_m(t)J_T - \dot{\theta}_m(t)B_T\Big) \tag{5.39}$$

$$m_c \cdot \ddot{x}_c(t) = F_F(V_a(t)) - m_p\Big(\ddot{x}_c(t) + l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t)\Big) \tag{5.40}$$

$$(J_p + m_p \cdot l^2) \cdot \ddot{\theta}_p(t) = m_p \cdot l \cdot (\sin(\theta_p(t)) \cdot g + \cdot \cos(\theta_p(t)) \cdot \ddot{x}_c(t)) \tag{5.41}$$

If no slip is assumed in the gearing and between the wheels and the ground, then $\dot{\theta}_m(t)$ and $\ddot{\theta}_m(t)$ can be replaced with $N_{ms}N_{sw} \cdot \dot{\theta}_w(t)$ and $N_{ms}N_{sw} \cdot \ddot{\theta}_w(t)$ respectively, and $\ddot{x}_c(t)$ can be replaced with $r_w \cdot \ddot{\theta}_w(t)$. This yields the three governing equations for the system model:

$$\tau_a(t) = \frac{\Big(\frac{K_t}{R_a}\Big(V_a(t) - \frac{K_e}{N_{ms}N_{sw}} \cdot \dot{\theta}_w(t)\Big) - \frac{J_T}{N_{ms}N_{sw}} \cdot \ddot{\theta}_w(t) - \frac{B_T}{N_{ms}N_{sw}} \cdot \dot{\theta}_w(t)\Big)}{N_{ms}N_{sw}} \tag{5.42}$$

$$m_c \cdot r_w \cdot \ddot{\theta}_w(t) = F_F(V_a(t)) - m_p\Big(\ddot{\theta}_w(t) \cdot r_w + l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t)\Big) \tag{5.43}$$

$$(J_p + m_p \cdot l^2) \cdot \ddot{\theta}_p(t) = m_p \cdot l \cdot \Big(\sin(\theta_p(t)) \cdot g + \cdot \cos(\theta_p(t)) \cdot r_w \cdot \ddot{\theta}_w(t)\Big) \tag{5.44}$$

To form the governing equation for the motors and wheels model, Equation 5.42 and Equation 5.43 are to be combined, see Figure 5.12. To meet the interfaces specified, see Figure 5.12, the relation between the motor and wheel model's output $\tau_a(t)$ and the force $F_F(t)$ which the cart movement model has as input, see Equation 5.42, must be determined.

There are two motors in the system, each exerting a torque, $\tau_a(t)$, which has to be included. A torque can be expressed as a force acting at a certain distance from the rotation point, also known as an arm. In this case, the arm is the wheel radius, so by using this, the force acting on the cart can be found. This relation can be expressed as:

$$F_F(t) = 2 \cdot \tau_a(t) \cdot \frac{1}{r_w} \tag{5.45}$$

Inserting the motor and wheel model, Equation 5.42, into Equation 5.45 yields:

$$F_F(t) = 2 \cdot \frac{\Big(\frac{K_t}{R_a}\Big(V_a(t) - \frac{K_e}{N_{ms}N_{sw}} \cdot \dot{\theta}_w(t)\Big) - \frac{J_T}{N_{ms}N_{sw}} \cdot \ddot{\theta}_w(t) - \frac{B_T}{N_{ms}N_{sw}} \cdot \dot{\theta}_w(t)\Big)}{r_w \cdot N_{ms}N_{sw}} \tag{5.46}$$

Combining Equation 5.46 and Equation 5.43 and using the fact that $\dot{\theta}_m(t) = \omega_m(t)$ yields:

$$\begin{aligned} m_c \cdot r_w \cdot \dot{\omega}_w(t) = 2 \cdot & \frac{\Big(\frac{K_t}{R_a}\Big(V_a(t) - \frac{K_e}{N_{ms}N_{sw}} \cdot \omega(t)\Big) - \frac{J_T}{N_{ms}N_{sw}} \cdot \dot{\omega}_w(t) - \frac{B_T}{N_{ms}N_{sw}} \cdot \omega_w(t)\Big)}{r_w \cdot N_{ms}N_{sw}} \\ & - m_p\Big(r_w \cdot \dot{\omega}_w(t) + l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t) - l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t)\Big) \end{aligned} \tag{5.47}$$

This differential equation describes the movement of the motors and wheels, and is simulated using Simulink, by inserting the values in the expression as found in Appendix A and Appendix B and applying a voltage step to the model. Since it is desired to test the relationship between the voltage applied to the motors and the movement of the cart, the final term in Equation 5.47 is set to zero. This is because this term is equal to F_L, as can be seen in Equation 5.35, i.e. this term describes the inverted pendulum's influence on the movement of the cart, which is disregarded in this test. The result of the test can be seen in Figure 5.13.

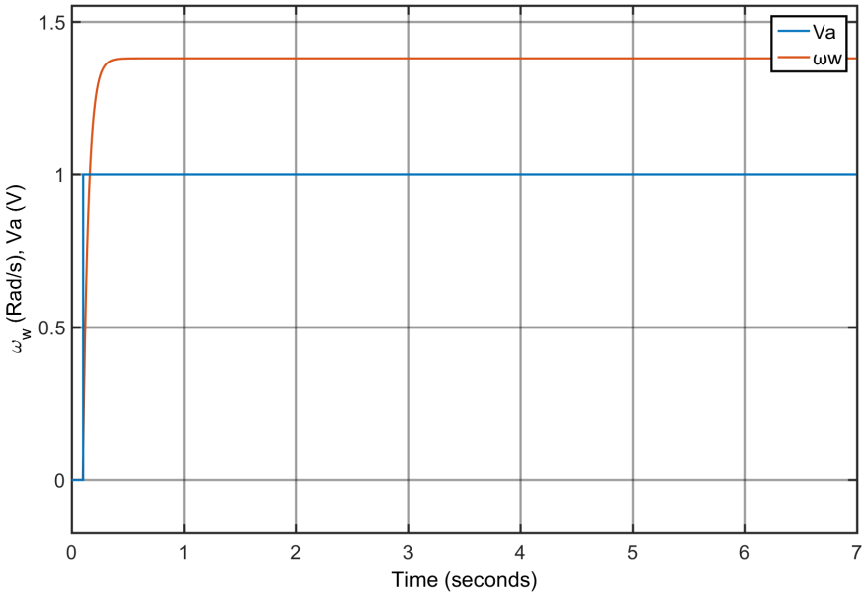


Figure 5.13: Simulation of motors and wheels model with an applied step as input.

Looking at the expression for the motors and wheels model in Equation 5.47, it is seen that the system is a first order when removing the final term in the expression. This is because the highest order of derivatives of $\omega_m(t)$ is one, giving a first order transfer function. This is partly due to the approximation done in Equation 5.3 about disregarding the inductance in the motor. From Figure 5.13, it can be seen that the output of the motor mimics a first order system's output, as the response is approaching the reference, without any oscillation or overshoot, which is not possible for a first order system. Therefore, the output of the simulation matches what is expected, and the model is therefore assumed to be valid.

The inverted pendulum model Equation 5.44 is also simulated in Simulink. The input to this model is chosen to be $\ddot{x}_c$ since this is the input parameter in Equation 5.44, by means of the expression $\ddot{x}_c = r_w\ddot{\theta}_w$. Also note that the inverted pendulum is initialised at an angle of -0.05 rad. The result can be seen in Figure 5.14.

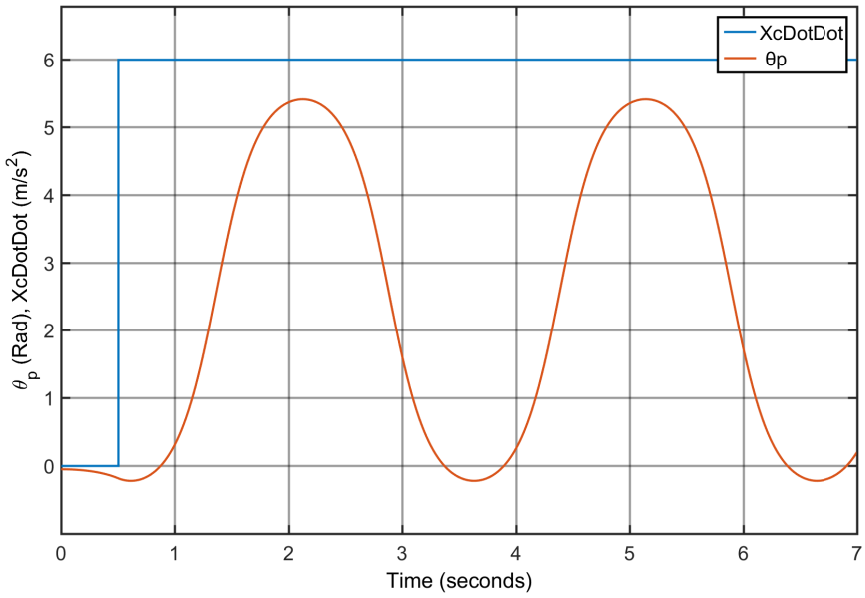


Figure 5.14: Simulation of the inverted pendulum model with an applied step as input.

From Figure 5.14 it can be seen that the inverted pendulum starts tilting to one side, and when the step is applied, the angle changes direction and lastly the inverted pendulum oscillates around a certain angle. This angle is π if the pendulum is swinging freely, corresponding to it hanging downwards, however an offset from π is seen, due to the step applied. Thus, the inverted pendulum model is assumed to be valid.

To validate the system model, the motors and wheels model and the inverted pendulum model are combine and simulated in MATLAB. As an initial conditions for the simulation, the inverted pendulum is set at an angle of 1 radian, as this will cause the inverted pendulum to fall. Furthermore, a small friction is inserted in the inverted pendulum model to make the result comply more with expectations. This is done because the angle's magnitude otherwise grows, i.e. the inverted pendulum will oscillate more violently over time. This can be due to a sign error on a small number in the model. Time constraints prohibits the localisation and correction of this minor issue. The results of the simulation can be seen in Figure 5.15.

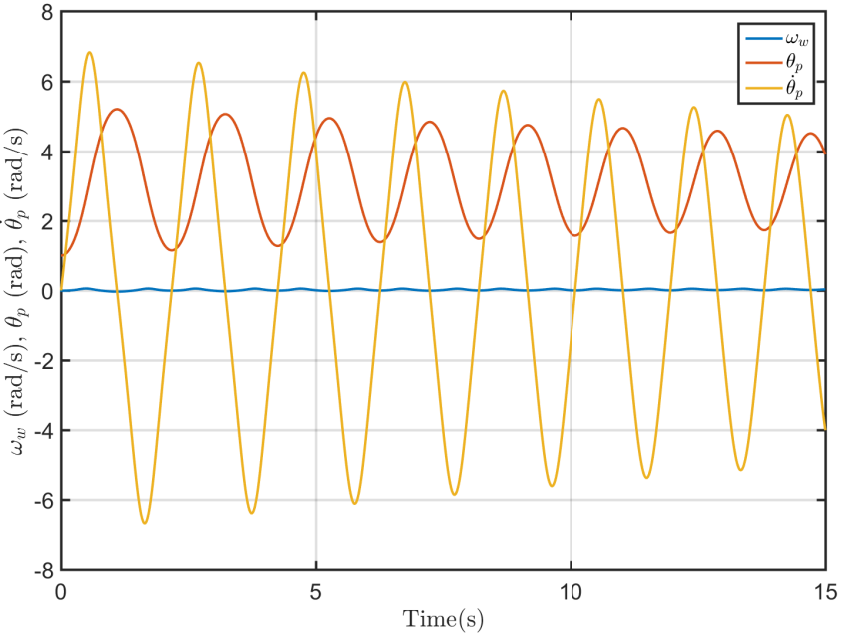


Figure 5.15: Simulation of the system model by setting the inverted pendulum at 1 radian and no input. Note: a small friction is inserted in the model to comply more with expectations.

From Figure 5.15, it can be seen that the pendulum slowly loses magnitude due to the inserted friction. Also, it oscillates around π which is to be expected. The model is hereby validated and can now be linearised.

5.4.2 Linearisation of Derived Equations of Motion

Before a transfer function of the system can be derived, it is necessary to linearise the two model equations, Equation 5.44 and 5.47, as it is only linear equations that can be Laplace-transformed to form a transfer function.

It is seen that the only part of the motors and wheels model, Equation 5.47, that is non-linear comes from Equation 5.43, thus only this part needs to be linearised. The second model equation, Equation 5.44, also needs to be linearised as it contains non-linear terms.

The method for doing the linearisation is chosen to be Taylor expansion. When linearising an equation, an expression for the tangent to the operating point is derived. This is shown in Figure 5.16.

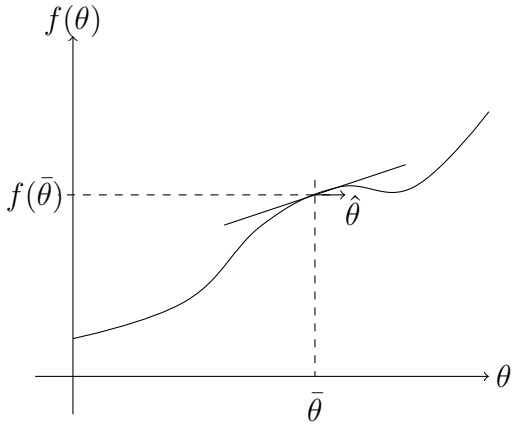


Figure 5.16: Illustration of linearisation principle.

Taylor expansion is based on the Taylor series, shown in Equation 5.48 [**?**]. The Taylor series approximation of a signal is enhanced by including a greater number of terms in the Taylor series, i.e. by having an infinite sum of terms, the approximation is exact. However, since the linearisation is a tangent, only the first two terms in the Taylor series is used in the Taylor expansion, as the resulting expression is otherwise not linear.

$$T(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + ... \tag{5.48}$$

Where:

T	is the taylor approximation
x	is the input
f	is the function to be linearised
a	is the linearisation point

The linearisation is performed around the segway's operating point, which is expressed as:

$$\theta_p(t) = \bar{\theta}_p(t) + \hat{\theta}_p(t) \tag{5.49}$$

Where:

$\theta_p(t)$	is the angle of the inverted pendulum	[rad]
$\bar{\theta}_p(t)$	is the operating point	[rad]
$\hat{\theta}_p(t)$	is the deviation from the operating point	[rad]

There are three variables included in the non-linear terms in the equations that are to be linearised, i.e. $\theta_p(t)$, $\dot{\theta}_p(t)$ and $\ddot{\theta}_p(t)$. Therefore, the Taylor series has to be done for each variable, which can then be added to form the linear expression with multiple linearised variables. The

Taylor expansion can be expressed as the following:

$$T = f(\bar{\theta}_p(t), \bar{\dot{\theta}}_p(t), \bar{\ddot{\theta}}_p(t)) + \frac{\partial f(\bar{\theta}_p(t))}{\partial \theta_p(t)} \cdot \hat{\theta}_p(t) + \frac{\partial f(\bar{\dot{\theta}}_p(t))}{\partial \dot{\theta}(t)_p} \cdot \hat{\dot{\theta}}_p(t) + \frac{\partial f(\bar{\ddot{\theta}}_p(t))}{\partial \ddot{\theta}(t)_p} \cdot \hat{\ddot{\theta}}_p(t) \tag{5.50}$$

The linearisation is done in the segway's upright position $\theta_p = 0$, i.e. this is the chosen operating point.

First, the cart model, Equation 5.43, is linearised. This is done in Appendix C, and the result can be seen in Equation 5.51.

$$(m_c + m_p) \cdot \ddot{x}_c(t) = F_F(t) + m_p \cdot l \cdot \hat{\ddot{\theta}}_p(t) \tag{5.51}$$

Analysing this equation, and comparing it to Figure 5.15, it is assumed that when the segway is being operated, i.e. a voltage is applied to the motors and the segway is therefore moving, the following is true:

$$F_F(t) >> m_p \cdot l \cdot \hat{\ddot{\theta}}_p(t)$$

This means that the force applied from the motors is much bigger than the force the inverted pendulum exerts on the cart. Based on this assumption, the inverted pendulum's influence on the cart's position is neglected, i.e. the last term in Equation 5.51 is removed. In a physical sense, this would mean that when the inverted pendulum is tilting, the cart will not move. Obviously, this is not the case, but the movement caused by the inverted pendulum is much smaller than the movement caused by the motors and wheels, and it is therefore seen as a fair assumption to make. This serves to make the model simpler, by removing the feedback from the pendulum to the cart.
This simplified linearised expression can then be combined with Equation 5.42 and Equation 5.45, to obtain a linear expression for the motors and wheels model, which can be seen in Equation 5.52.

$$(m_c + m_p) \cdot r_w \cdot \ddot{\theta}_w(t) = 2 \cdot \frac{\left(\frac{K_t}{R_a}\left(V_a(t) - \frac{K_e}{N_{ms}N_{sw}} \cdot \dot{\theta}_w(t)\right) - \frac{J_T}{N_{ms}N_{sw}} \cdot \ddot{\theta}_w(t) - \frac{B_T}{N_{ms}N_{sw}} \cdot \dot{\theta}_w(t)\right)}{r_w \cdot N_{ms}N_{sw}} \tag{5.52}$$

With the motors and wheels model linearised, the inverted pendulum model shall also be linearised. The inverted pendulum model, see Equation 5.44, is linearised in Appendix C, which yields:

$$(J_p + m_p \cdot l^2) \cdot \hat{\ddot{\theta}}_p(t) = m_p \cdot l\Big(g \cdot \hat{\theta}_p(t) + r_w \cdot \ddot{\theta}_w(t)\Big) \tag{5.53}$$

Both model expressions are now linear and can be Laplace transformed to yield transfer functions, which is necessary to be able to design a controller for the segway. The segway's transfer function is now to be determined.

5.5 Verification of Linear Models

Since the system model consists of two models, Equation 5.52 and 5.53, the verification of the system model will be done in two steps. First, the individual models will be verified followed by a verification of the system model, where the two models are combined.

5.5.1 Verification of Motors and Wheels Model

First, Equation 5.52 is Laplace transformed to obtain a transfer function for the motors and wheel as follows:

$$\frac{\Theta_w(s)}{V_a(s)} = \frac{2 \cdot k_t \cdot N_{ms} \cdot N_{sw}}{R_a(s^2(m_c \cdot r_w^2 \cdot (N_{ms} \cdot N_{sw})^2 + 2 \cdot J_T) + s(2 \cdot \frac{k_t \cdot k_e}{R_a} + 2 \cdot B_T))} \tag{5.54}$$

To be able to verify this linear model, parameter estimation is performed. It's purpose is to obtain a more precise model, as there may be deviations in the parameters, since there can uncertainties within the measurements. By using MATLAB it is possible to estimate parameters based on an initial guess. From this initial guess, MATLAB tunes the parameters to get a closer fit for the data. This fit is achieved by an optimization algorithm. By inserting the values found in Appendix B and Appendix A into Equation 5.54 yields:

$$\frac{\Theta_w(s)}{V_a(s)} = \frac{27.98}{s \cdot (s + 20.29)} \tag{5.55}$$

$$\frac{\Omega_w(s)}{V_a(s)} = \frac{27.98}{s + 20.29} \tag{5.56}$$

Note that the second equation is obtained by simply multiplying the equation with an s, which corresponds to a differentiation. This is done since it is preferable to have the output of the transfer function as angular velocity instead of the angle, as the encoders measure the angular velocity. Equation 5.56 is used as the initial guess for parameter estimation in MATLAB. To do parameter estimation, MATLAB needs pairs of a given input to the system and a measured output. The input should be chosen so that the dynamics of the system are seen. Here, the PWM input signal for the motors is chosen as the input signal, and is set to switch between the voltages 8.8 V and -8.8 V. The output is the angular velocity of the wheels. For MATLAB to perform parameter estimation, the model type is also needed. As previously mentioned, the motor is assumed to be a first order system, which is therefore used as the model type.

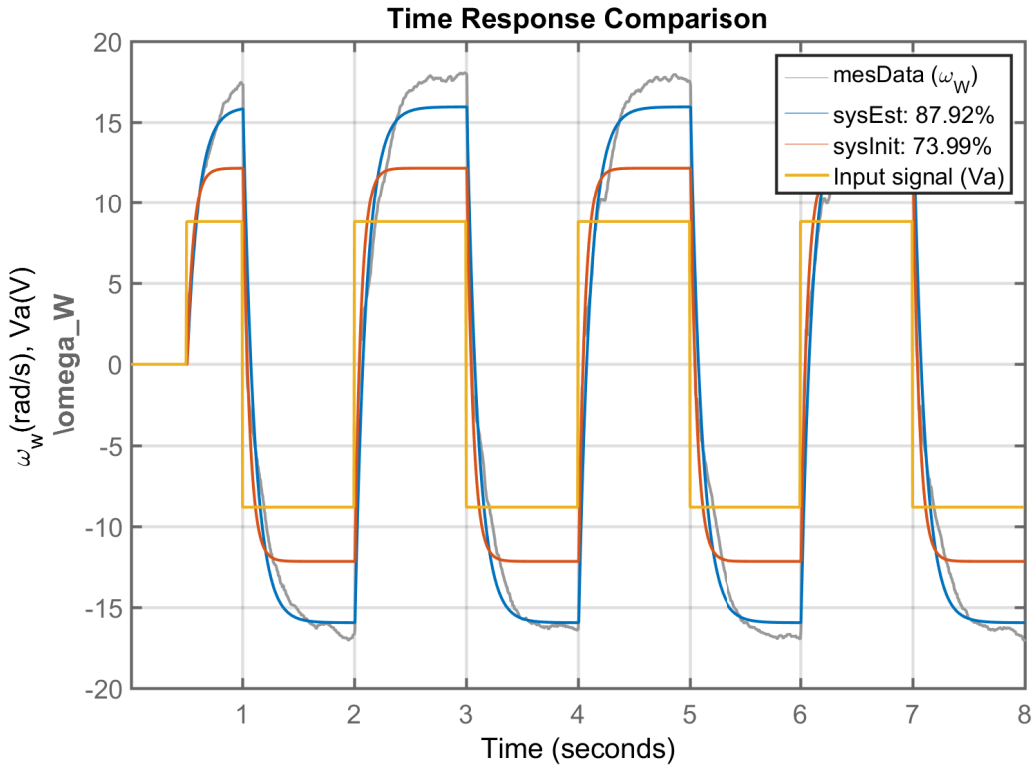


Figure 5.17: Time response of both initial guess and optimized model based on given input and measured output.

The response of the system over time can be seen in Figure 5.17. The input signal is shown in yellow, and the output of the initial guess model is shown in orange, together with the measured data, which is shown in grey. The blue axis is MATLAB's estimate, based on the model type and measured data. The fit of 87.92 % for the parameter estimation is considered acceptable, making the final model of the motors and wheels the following expression.

$$\frac{\Omega_w(s)}{V_a(s)} = \frac{17.72}{s + 9.799} \tag{5.57}$$

5.5.2 Verification of Inverted Pendulum Model

To verify the inverted pendulum model, a transfer function where $\Theta_w(s)$ is input and $\Theta_p(s)$ is output, needs to be derived. Thus Equation 5.53 has to be Laplace transformed as well as rearranged to obtain the desired transfer function:

$$\frac{\Theta_p(s)}{\Theta_w(s)} = \frac{m_p \cdot l \cdot r_w \cdot s^2}{s^2(J_p + m_p \cdot l^2) - m_p \cdot l \cdot g} \tag{5.58}$$

The model for the inverted pendulum, Equation 5.58, is to be verified by turning the segway

upside down and reversing the gravity in the model expression. This has the implications of the system becomes stable, which is necessary for parameter estimation to work. The segway is then moved on rails and the pendulum angle is measured with $\theta_p(t) = 0$ being downwards instead of upwards. This means that from a modelling perspective, the only thing changed is the direction of gravity. By inserting the values found in section B and section A into Equation 5.58 yields:

$$\frac{\Theta_p(s)}{\Theta_w(s)} = \frac{0.09217 \cdot s^2}{s^2 + 15.47} \tag{5.59}$$

$$\frac{\Theta_p(s)}{\Omega_w(s)} = \frac{0.09217 \cdot s}{s^2 + 15.47} \tag{5.60}$$

Note that as for the motors and wheels model's output, the input of this transfer function is the angular velocity of the wheel instead of the the angular position of the wheel. This is put through MATLAB's system identification tool as explained previously, from which the results can be seen in Figure 5.18. An input delay is used even though no delay is seen in the model, which is done to get a better fit.

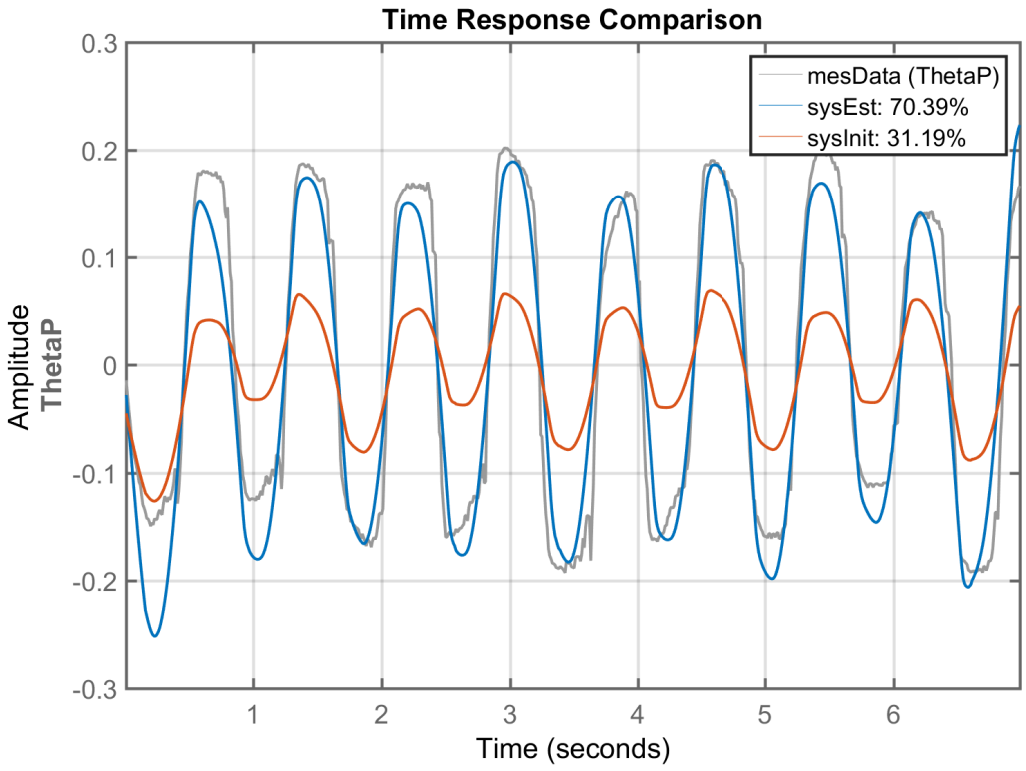


Figure 5.18: Time response of inverted pendulum model.

From Figure 5.18 it can be seen that a fairly big change happens to the output and fit of the system. This is seen as a problem, due to the fact that most of the parameters in the transfer function are lengths and masses, which are quite simple to measure without a lot of

uncertainties. A possible cause for this discrepancy is that the model may not include enough factors to represent reality adequately, but due to time constraints it is not possible to review the model further, and therefore the sysEst provided by MATLAB will be used for controller design. A concern is also that this sysEst fits this exact set of data, which only represent one type of input. After reversing gravity again, to describe the unstable system, the transfer function for the inverted pendulum model becomes:

$$\frac{\theta_p(s)}{\omega_w(s)} = \frac{0.2367 \cdot s}{s^2 - 21.92} \tag{5.61}$$

5.5.3 Verification of Segway Model

To verify the linear system model, the two newly derived models from the two previous sections are multiplied and compared to both the system model and the measured data. The data is measured in the same way as the inverted pendulum model verification. The comparison is done in Simulink where the system is simulated, and the results can be seen in Figure 5.19.

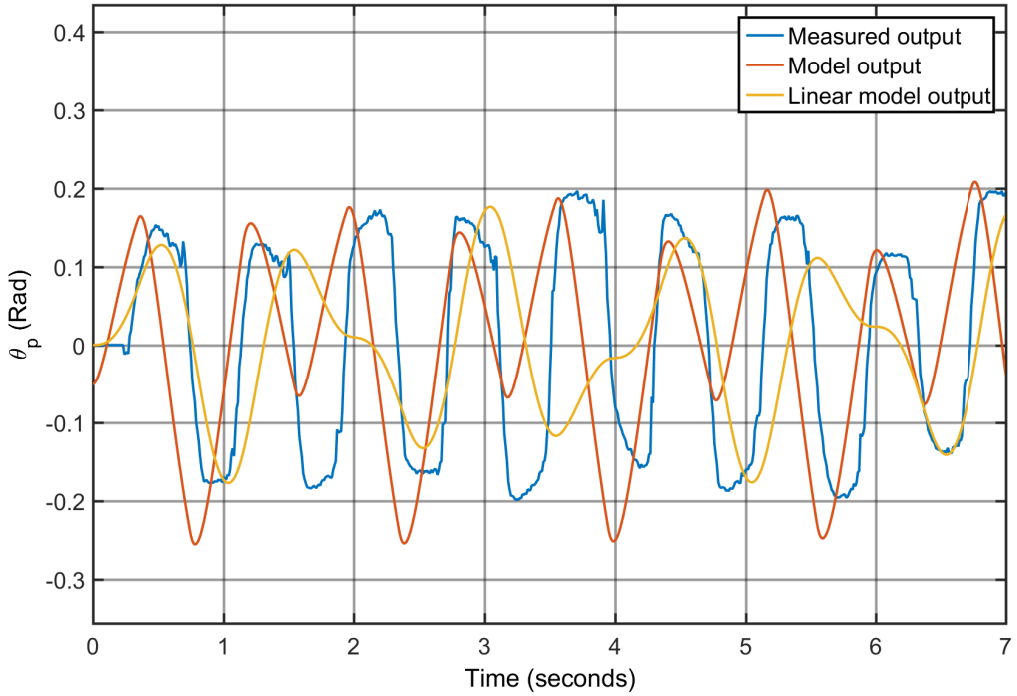


Figure 5.19: Time response of system model and linear model compared to measured data.

From Figure 5.19 it can be seen that the fit between the models and reality is poor. It is believed to be for the same reasons as mentioned in subsection 5.5.2, but could also be due to implementation errors in Simulink, which due to time constraints are not further investigated. A controller design will be based upon these models, despite this poor fit.

Now that the transfer function for the segway is found, it can be used for the controller design, which is done in the following chapter.

6 | Controller Design

The previous chapter had the purpose of deriving a model for the segway's behaviour to allow a controller to be designed. The control system can be an open or closed loop depending on which type of system that is to be controlled. The difference is whether or not the output of the controller is fed back and affects the input of the controller block. As the segway has to be able to balance in an upright position, it is necessary to feedback control it, see Figure 6.1, as the system model can never be perfect. In this chapter, a controller for the system is to be designed.

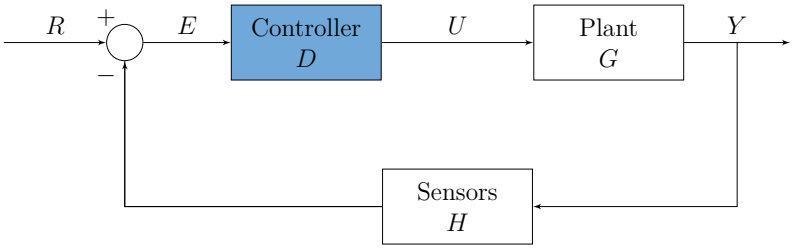


Figure 6.1: Feedback loop of the system, with the controller, D, highlighted.

Figure 6.1 shows the feedback loop of the system. It is a negative feedback loop, meaning the output signal, Y, of the plant, G, is fed back through the sensors block, H. This feedback signal is then subtracted from the reference signal, R, yielding an error signal, E, which is the input to the controller block, D. This will result in the signal, U, between the controller and plant block. As described in Section 2.3.1, the sensor block is assumed equal to 1, and will therefore not be seen in the transfer function for the controller.
In the following sections, requirements for the controller and choice of controller design is presented. The controller is the designed and verified by simulations and lastly implemented in the segway.

6.1 Design Considerations

The design of the controller has to comply with the requirements from section 3.2. This is done to ensure that the design of the controller meets the desired performance. The requirements are listed as:

- Steady state error: 0 %
- Rise time: < 1 s
- Settling time: < 3 s
- Overshoot: < 10 %
- Phase margin: $\geq 80°$
- Gain margin: ≥ 6 dB

The design of the controller will be based on these requirements and the model derived from the previous chapter.

6.1.1 Cascade Controller

When controlling a system where multiple sensors can be implemented, allowing feedback signals from different places in a process, cascade control is often a preferable way of controlling the system. This is because a cascade control improves robustness against disturbances and improves the dynamic performance of the system [**?**]. It has therefore been decided to design a cascade controller. Loops within a cascade controller are coupled such that there is an outer and inner loop. If the inner loop's rise time is 3-5 times faster than the outer loop's, then the inner loop can be seen as having a transfer function of 1, when designing the outer loop [**?**]. This simplifies the design process of the outer loop considerably.

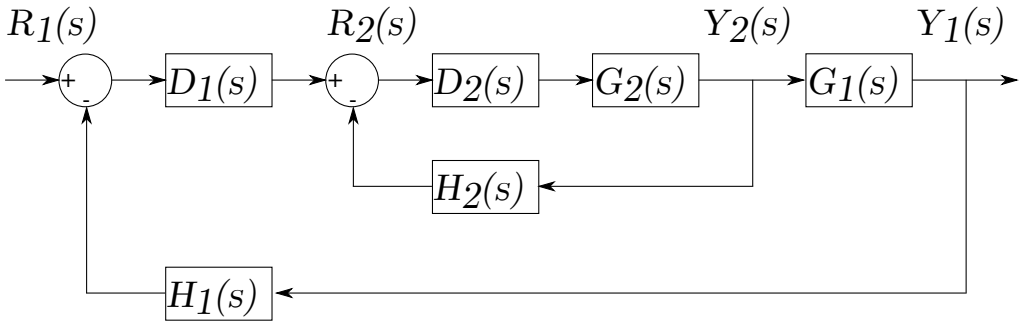


Figure 6.2: Cascade control with two loops.

Figure 6.2 displays a cascade control loop of two controllers. A cascade controller can hold more than two loops depending on how many sensor inputs are available in a system. When applying cascade control to the system of the segway, the inner loop consists of an inner loop controller, $D_2(s)$, the motors and wheels model, $G_2(s)$, along with the sensor block $H_2(s)$. The outer loop consists of an outer loop controller, $D_1(s)$, the inverted pendulum model, $G_1(s)$, along with $H_1(s)$, see Figure 6.2.

A cascade controller is designed in the following section.

6.2 Design of Cascade Controller

In this section, controllers for the two cascade loops are to be designed. First the inner loop is designed and afterwards the outer loop is designed. These will afterwards be verified in a later section.

6.2.1 Inner Loop

In this section the inner loop controller is designed. The inner loop has the purpose of controlling the motors and wheels model and ensure that the effect of noise and uncertainties in the system are diminished, resulting in better performance for the balancing segway.

The transfer function for the motors and wheels is:

$$G_2(s) = \frac{17.72}{s + 9.799} = \frac{1.81}{0.10s + 1} \tag{6.1}$$

Where:

$G_2(s)$	is the plant of the inner loop	[1]

The motors and wheels model is a first order system with a pole in -9.799. This means it is a type 0 system, as it does not have any poles in zero. This reveals some characteristics of the system. One of these is the presence of a steady-state error in its step response [**?**, p. 211]. This can only be fully eliminated if an integrator is implemented, as this yields a type 1 system. It is however chosen not to do this, but instead only implement a gain as this is assumed to reduce the steady-state error adequately and allows for a simpler controller design. A P-controller is therefore designed for the inner loop. A P-controller has the form as seen below:

$$D_2(s) = k_{p2} \tag{6.2}$$

Where:

$D_2(s)$	is the controller for the inner loop	[1]
k_{p2}	is the proportional gain	[1]

To design a P-controller it is necessary to look into how fast the system is sampled as this, together with saturation, sets an upper limitation for how much gain the system can handle. However, in the design of this controller, the saturation is disregarded as it is assumed that the influence is negligible. The motors' encoders sample with a frequency of 1 kHz. This is also the frequency of the inner loop. It is preferable, by rule of thumb, to have a system bandwidth that is more than 25 times smaller than the angular sampling frequency [**?**, p. 613]. It is however chosen to design with a factor of 10. Note that the sampling frequency is converted from hertz to radians per seconds, and thus the bandwidth is found as:

$$\omega_{BW} = 2\pi \cdot \frac{f_s}{10} = 2\pi \cdot \frac{1000}{10} = 628.32 \tag{6.3}$$

Where:

ω_{BW}	is the bandwidth of the inner loop	[rad/s]
f_s	is the sampling frequency of the inner loop	[Hz]

From Equation 6.3 the bandwidth of the plant in the inner loop is found to be 628.32 rad/s. By examining the bode plots of the plant without a controller, the plant's gain can be determined.

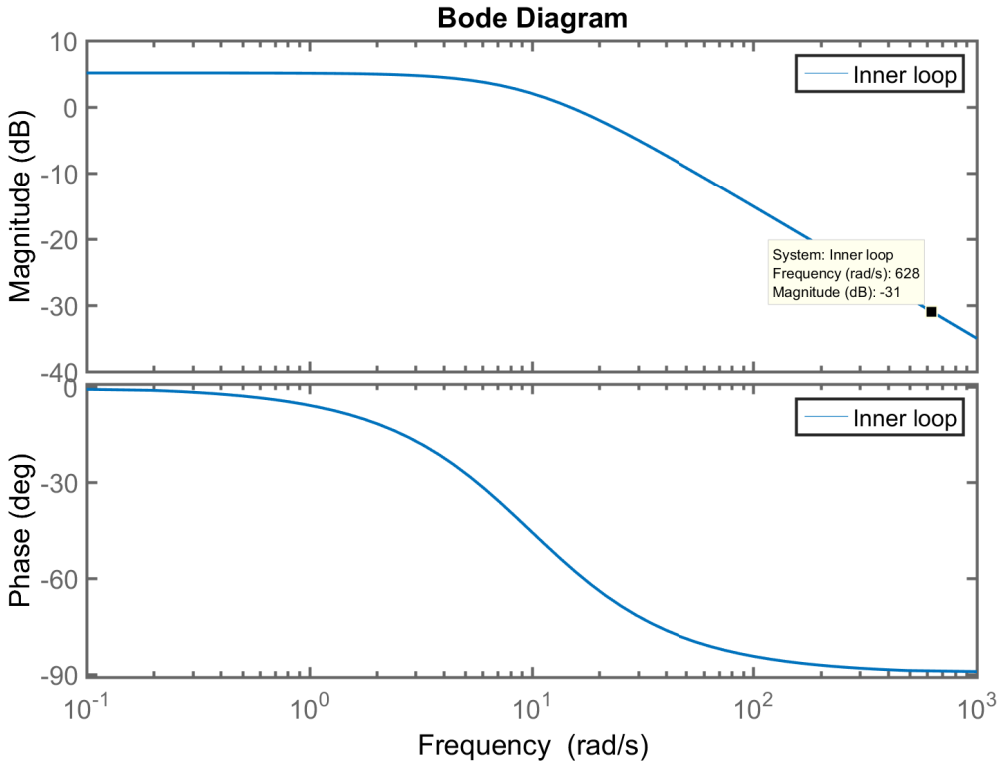


Figure 6.3: Open loop bode plots of motors and wheels model, where the controller has a gain of 1. The approximate bandwidth, 628 rad/s, is marked.

From Figure 6.3 it can be seen that the magnitude is approximately -31 dB. To obtain a magnitude of -3 dB, the graph shall be lifted 28 dB. The scalar gain can be found as follows:

$$k_{p2} = 10^{\frac{28}{20}} = 25.11 \tag{6.4}$$

The P-controller has now been designed and will in the following section be verified.

6.2.2 Verification of Inner Loop Controller

The gain for the inner loop controller shall be 25.11 according to Equation 6.4. The transfer function of the closed loop can be expressed as shown in Equation 6.5 where $H_2(s)$ is 1, due to no gain in the sensor block.

$$\frac{Y_2(s)}{R_2(s)} = \frac{D_2(s) \cdot G_2(s)}{1 + D_2(s) \cdot G_2(s) \cdot H_2(s)} = \frac{25.11 \frac{17.72}{s+9.799}}{1 + 25.11 \frac{17.72}{s+9.799}} \tag{6.5}$$

Where:

$Y_2(s)$	is the output of the inner loop	[1]
$R_2(s)$	is the reference to the inner loop	[1]
$H_2(s)$	is the sensor block in the inner loop	[1]

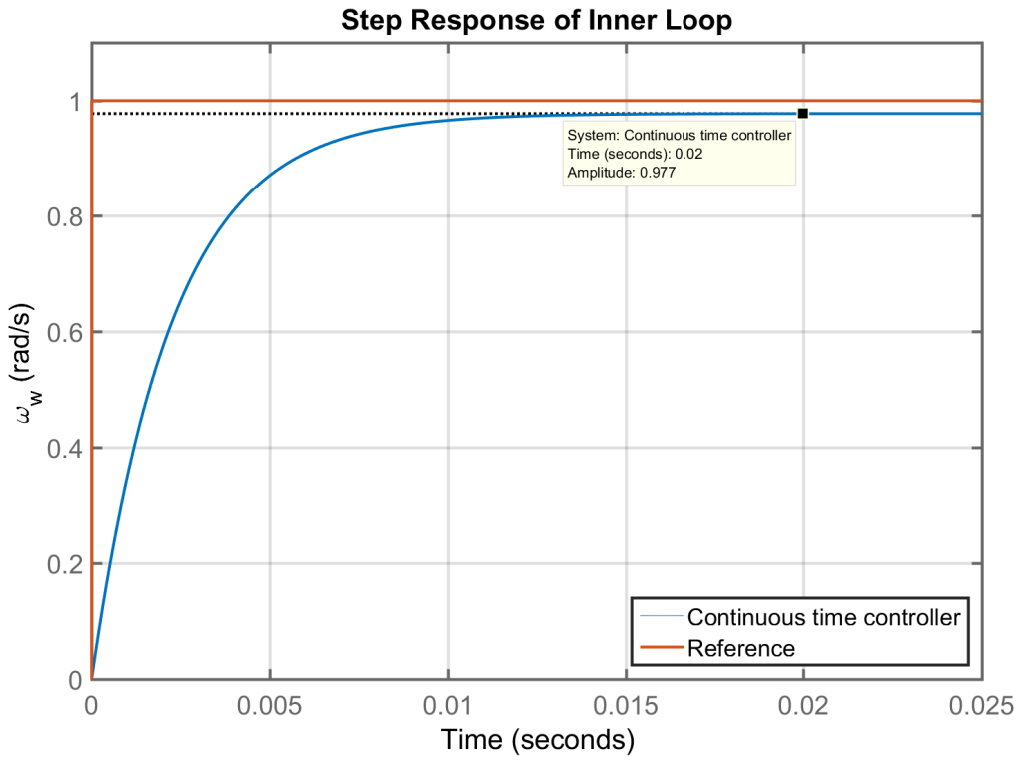


Figure 6.4: Simulation of the step response for inner loop with the designed continuous time P-controller.

From Figure 6.4 it can be seen, that the steady-state error is 2.2% which is deemed small enough that effect of this error on the full segway system is negligible. When looking at the characteristic equation, which is the main denominator in the closed loop transfer function, it can be seen that the gain changes the pole placement. The pole is moved far out in the left half plane (LHP). This increases the speed of the inner loop's dynamics, which can be seen in Figure 6.4, where the step response settles within 0.02 seconds.

The closed loop transfer function is now discretized using MATLAB. The method used is the bilinear transformation, which is further described in subsection 7.3.1. This yields:

$$\frac{Y_2(z)}{R_2(z)} = \frac{0.2151z^2 + 0.002119z - 0.213}{1.215z^2 - 1.978z + 0.7674} \tag{6.6}$$

This is compared to the continuous time controller which can be seen in Figure 6.5.

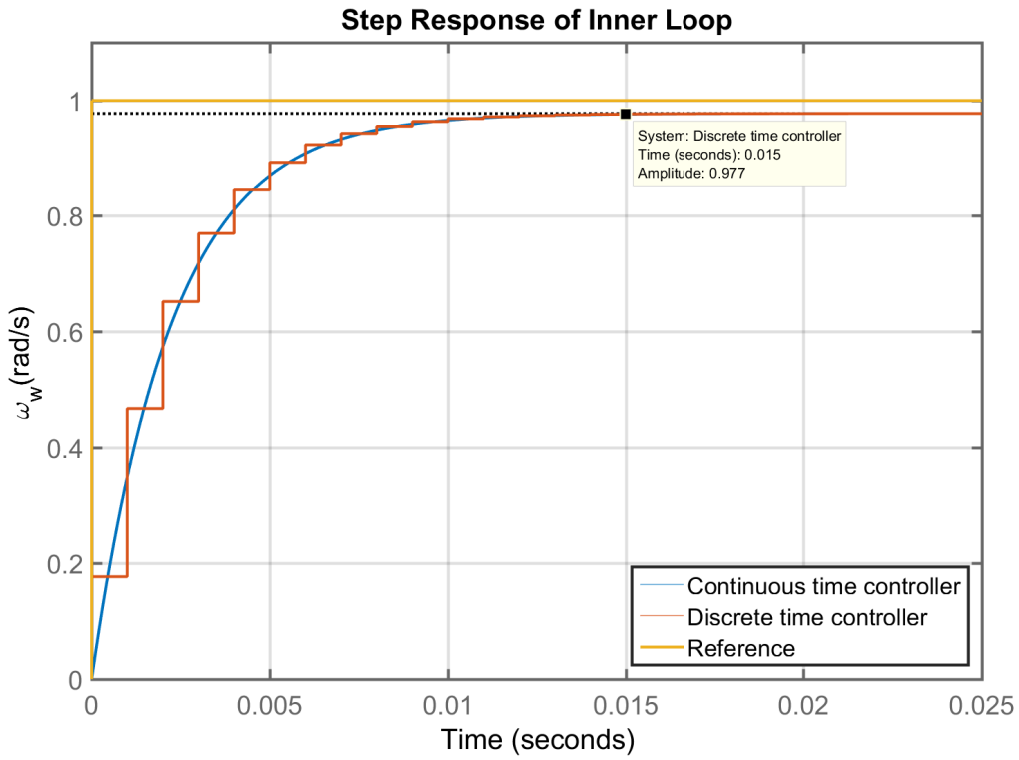


Figure 6.5: Simulation of the step response of continuous and discrete time inner loop controller.

From Figure 6.5 it can be seen that the discrete controller matches the continuous controller and the P-controller with a gain of 25.11 for the inner loop is hereby derived. It can further be seen, that at 15 ms the velocity has almost reached its steady-state value, which means that if 15 ms is chosen as sampling time for the outer loop, the inner loop can be seen as 1 in this regard.

The outer loop will be designed in the following section. These two loops forms the cascade controller for the full system.

6.2.3 Outer Loop

In this section a controller is designed for the inverted pendulum, who's transfer function can be seen in Equation 6.7. It can be seen that there is a pole in the right half plane (RHP), meaning that the system is unstable. The controller is therefore designed with the primary purpose of stabilizing the system. Additional requirements that are not met by this controller can be met by the use of compensators, such as lag or lead-lag compensators.

The transfer function of the inverted pendulum is the following:

$$\frac{\Theta_p(s)}{\Omega_w(s)} = \frac{0.2367 \cdot s}{s^2 - 21.92} = \frac{0.2367 \cdot s}{(s + 4.682)(s - 4.682)} \tag{6.7}$$

The first step is to move the pole that makes the system unstable to the left half plane (LHP), by adding a zero in the LHP to the left of the leftmost plant pole [**?**, p. 273]. The second step is to add a gain, which actually moves the poles. This can be implemented with a PD-controller in the form of:

$$D_1(s) = (s - k_z)k_{p1} \tag{6.8}$$

Where:

k_z	is the location of the zero	[1]
k_{p1}	is the proportional factor	[1]

With this controller type, there will be two zeros and two poles in the open loop. Increasing the k_{p1} value will cause each pole to move closer to a zero. Leading to that the RHP pole will end in the zero in origin and the LHP pole will end in the LHP zero [**?**, p. 268]. This makes it impossible to move the RHP pole into the LHP, as it will stop in the origin, due to the zero there. Therefore this zero has to be cancelled. This is done through pole-zero cancellation, by adding an integrator to the controller. This yields the following PID-controller:

$$D_1(s) = \frac{(s - k_z)k_{p1}}{s} \tag{6.9}$$

The zero has to be placed in the LHP to the left of the leftmost plant pole, as this causes the root locus shown in Figure 6.6. The zero is placed in -10 to give a margin for errors in the model

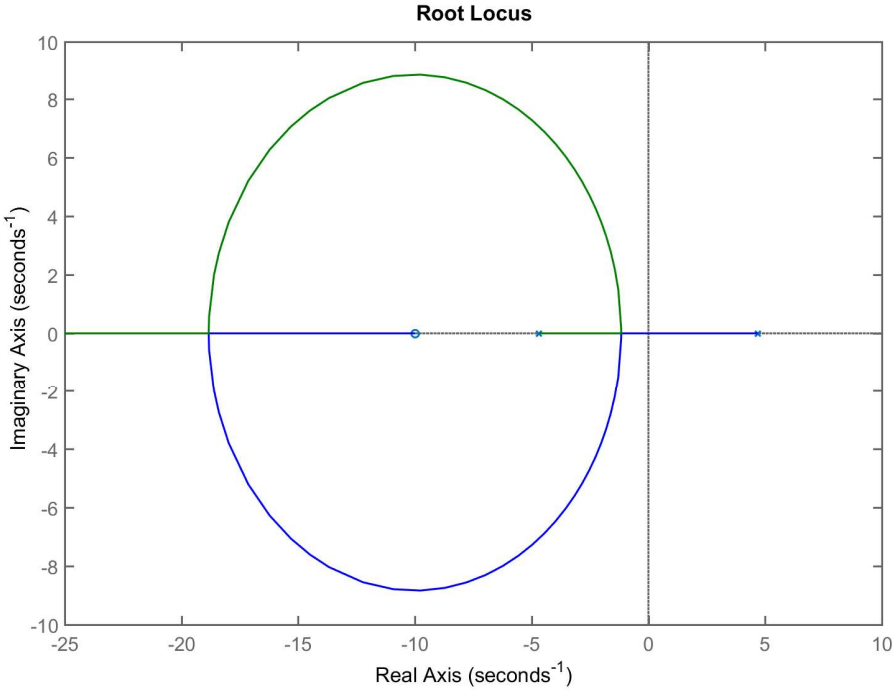


Figure 6.6: The root locus of the system, where a zero has been inserted in -10 and an integrator has cancelled the zero in origin.

The task now is to determine the k_{p1} factor. This is done through consideration of the requirements. The requirement for overshoot, MP, is that it has to be within 10%. This yields a damping, ζ, of 0.8, as the plant is a second order system. This value is calculated with Equation 6.10 [**?**, p. 153].

$$MP = e^{\frac{-\pi \cdot \zeta}{\sqrt{1-\zeta^2}}} \tag{6.10}$$

The bandwidth, ω_{BW}, is now calculated based on the sampling frequency, f_s, in the same manner as done for the inner loop:

$$\omega_{BW} = 2 \cdot \pi \frac{f_s}{10} = 2 \cdot \pi \frac{1}{T_s \cdot 10} = 2 \cdot \pi \frac{1}{15 \cdot 10^{-3} \cdot 10} = 41.888 \tag{6.11}$$

Where:

ω_{BW}	is the bandwidth of the system	$\left[\frac{rad}{s}\right]$
f_s	is the sampling frequency	$\left[\frac{1}{s}\right]$
T_s	is the sampling time	$[s]$

The natural frequency can be calculated as shown in Equation 6.12 [**?**]:

$$\omega_n \approx \frac{\omega_{BW}}{1.4} = \frac{41.888}{1.4} = 29.92 \tag{6.12}$$

Where:

ω_n	is the natural frequency	$\left[\frac{rad}{s}\right]$

Thus the highest allowable ω_n is $29.92\frac{rad}{s}$, which yields an approximate rise time, t_r, as calculated in Equation 6.13 [**?**, p. 152].

$$t_r \approx \frac{1.8}{\omega_n} = \frac{1.8}{29.92} = 0.060 \tag{6.13}$$

Where:

t_r	is the rise time of the step response	$[s]$

This rise time is considerably less than the 1 second from the requirements, thus the ω_n leads to this requirement being fulfilled. The fast rise time may however lead to saturation problems, as a fast rise time requires a high control gain, leading to a high controller output, making saturation more likely. This will be dealt with through tuning.

The open loop is now plotted as a root locus, and the damping, ζ and natural frequency, ω_n, are inserted as boundaries. The white area fulfills the requirements and thus the poles should be placed here. The k_{p1} value is chosen to be 160, yielding the pole locations shown in Figure 6.7. The poles are placed on the real axis close to the break-in-point, as a high gain yields a low steady-state error and overshoot.

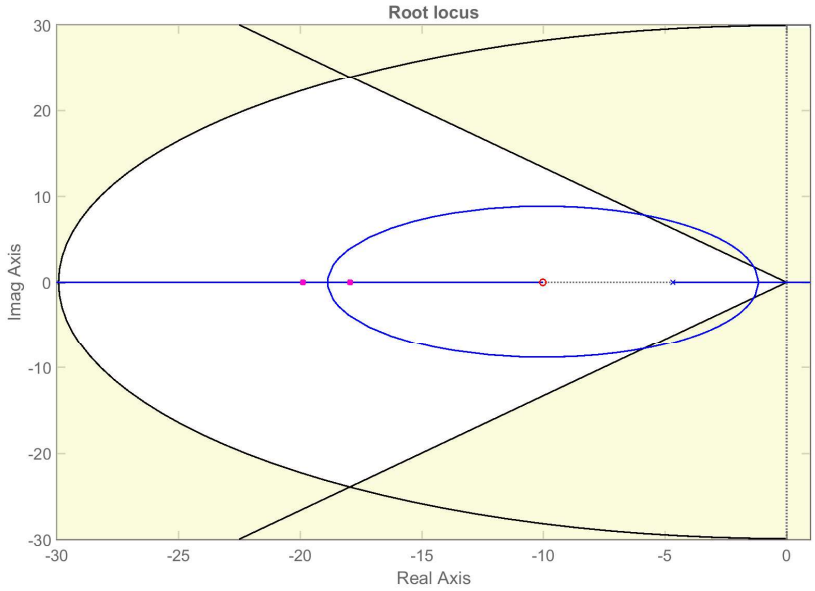


Figure 6.7: The root locus diagram of the open loop, with the requirement for damping and natural frequency included. The white area is the area where the requirements are met.

The controller can thus be written as:

$$D(s) = \frac{(s+10)160}{s} \tag{6.14}$$

The bode plots for the open loop of the outer loop is shown in Figure 6.8.

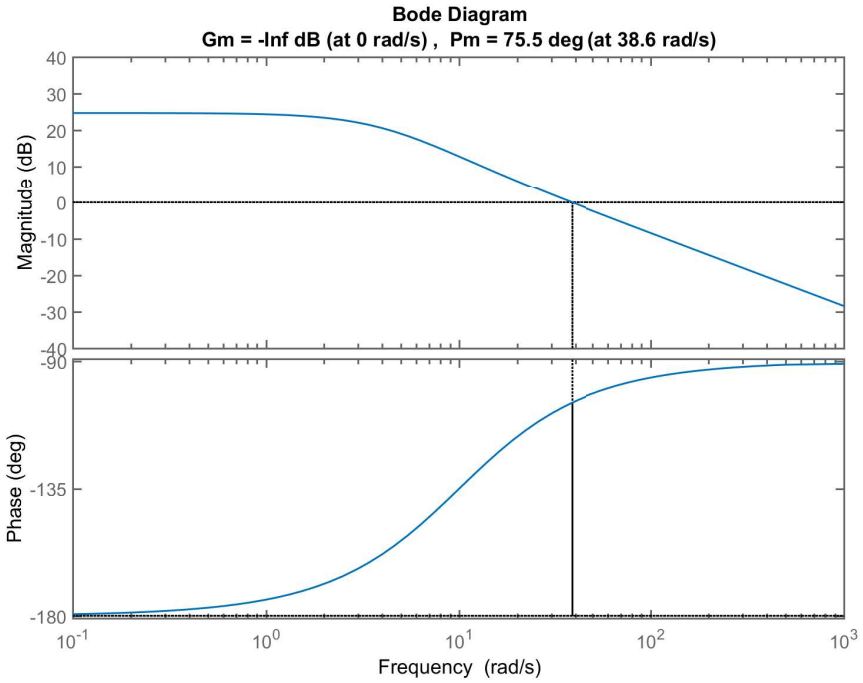


Figure 6.8: The open loop bode plots of the outer loop.

From this it is seen that the phase margin and gain margin requirements are not met. This is not an issue, as the system has been stabilized as a closed loop system, due to the root locus method. Thus the behaviour of the open loop system is not required to be stable. Since the phase and gain margins are stability criterias for the open loop, which is not required to be stable, the issue is disregarded. A simulation of the step response of the outer loop, with this controller is shown in Figure 6.9.

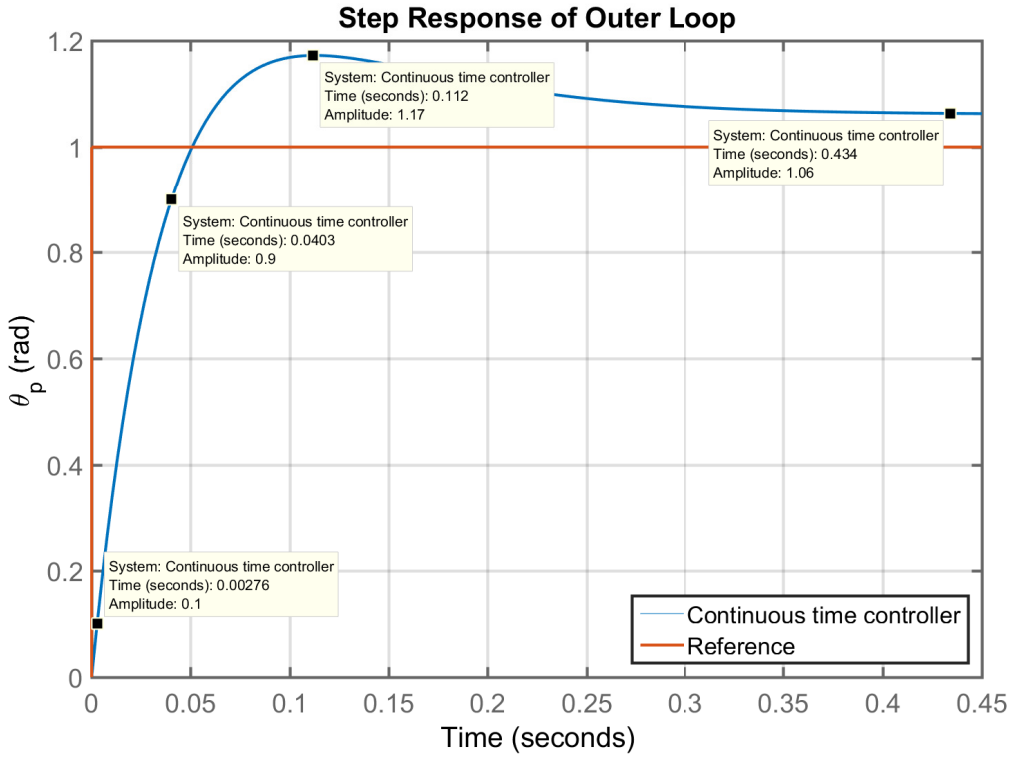


Figure 6.9: Simulation of the step response of the outer loop with the designed continuous time PID-controller.

From the step response, the following dynamics are determined as:

- Rise time: $t_r = 0.038s$
- Steady-state error: $e_{ss} = 6\ \%$
- Overshoot: MP $= 9.4\ \%$
- The system never settles at 1 in amplitude, it does however reach a steady state at approximately 0.45 s. Thus $t_s \approx 0.45s$.

Comparing these system dynamics with the requirements, it is seen that the rise time, settling time and overshoot are within the requirements. On the other hand, a steady-state error is present which is a violation of the requirements. The reason for the steady-state error is that the closed loop is of system type zero [**?**, p. 211]. This can however be corrected by an additional cascade which will not be designed or implemented due to time constraints. The system is stable and most of the requirements are met.

Now the controller can be discretized. This is done in MATLAB using bilinear transformation, which yields:

$$D_1(z) = \frac{172z - 148}{z - 1} \tag{6.15}$$

This controller is compared to the continuous time controller which can be seen in Figure 6.10.

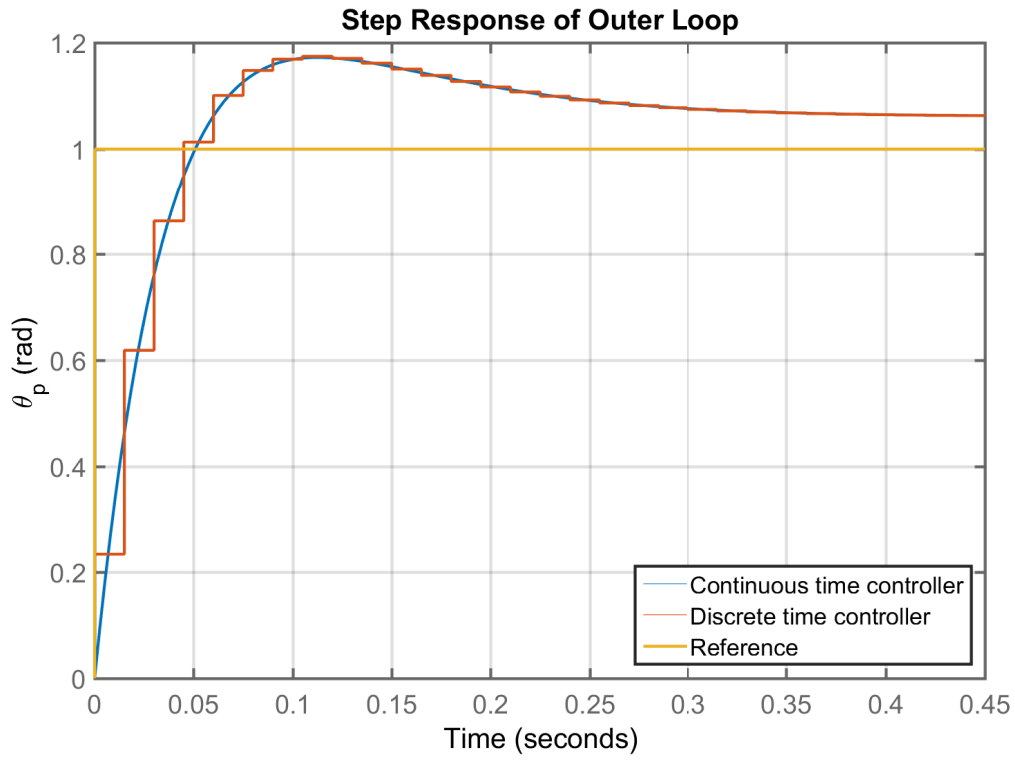


Figure 6.10: Simulation of the step response of continuous and discrete outer loop controller.

From Figure 6.10 it can be seen that the discrete controller matches the continuous controller. The controller design is therefore satisfying.

6.3 Validation of Cascade Controller

The final step before implementing the controller is to verify that it can stabilize the model as well as the linearised model. For this purpose the controllers have been implemented in MATLAB. Note that the inner loop controller has been implemented as a continuous time controller due to simplicity, and a step on the reference is applied. This step is set to 0.1 radians instead of 1 radian as the controller is designed to work close to the operating point and 1 radian is seen as too far away. Due to the characteristics of the linearity of the linear model it is still possible to compare the results from this simulation to the simulation of the linear control system. The result can be seen in Figure 6.11.

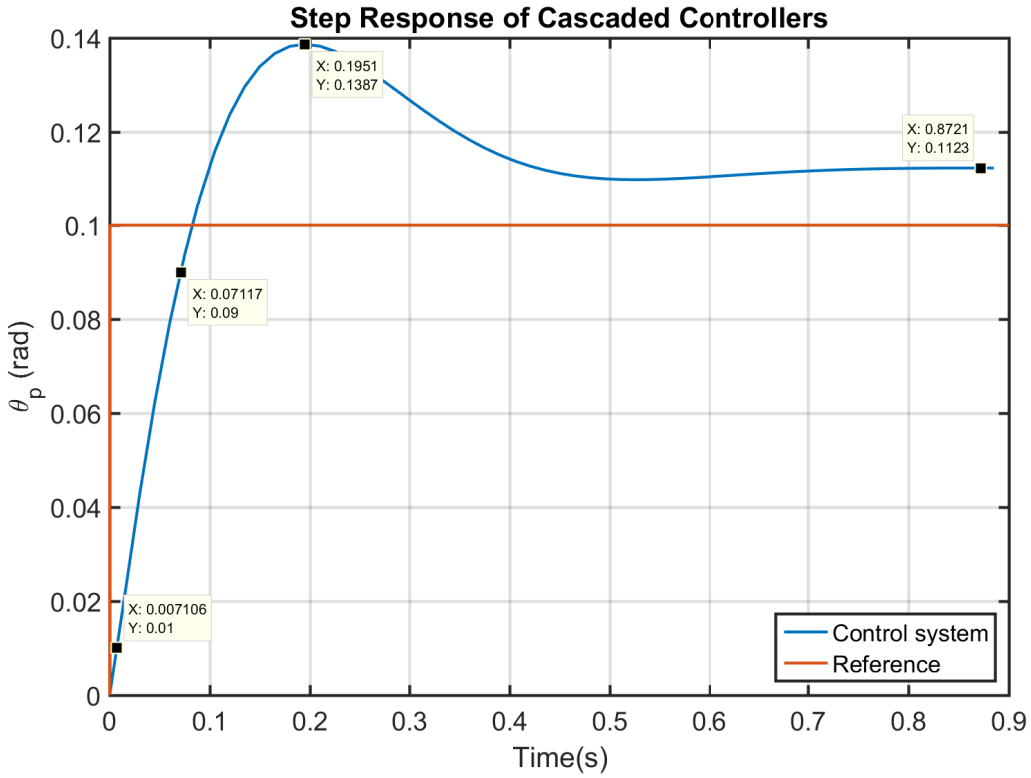


Figure 6.11: Simulation of the step response of the control system consisting of a discrete outer loop controller a continuous inner loop controller and the original model.

From the step response seen in Figure 6.11, the following dynamics are determined:

- Rise time: $t_r = 0.064s$
- Steady-state error: $e_{ss} = 12.3$ %
- Overshoot: MP $= 23.5$ %
- The system never settles at 1 in amplitude, it does however reach a steady state at approximately 0.7 s. Thus $t_s \approx 0.7s$.

From this it can be seen that two of the requirements are met and two are not met, where, for the linearised model, only the requirement for the steady-state error were not met. This may be due to differences between the linearised model and the model. Since these differences most likely arise from the problems mentioned in parameter estimation of the inverted pendulum model, this is not considered an issue as it is assumed that tuning can correct this.

6.4 Implementation of Cascade Controller

After implementation of the controllers on the segway, it is seen that the segway overreacts to errors. Therefore the controller coefficients for the outer loop controller are adjusted, resulting in the following final implementation:

$$D(z) = \frac{3.8 - 2.56 \cdot z^{-1}}{1 - z^{-1}} \tag{6.16}$$

The reason for the substantial adjustment compared to the original implementation in Equation 6.15, might be due to either that the model is missing something critical or that the effects of the saturation is not negligible as anticipated. If this controller is applied to the linear model, the simulation of a step on the reference results in an unstable response, which is not surprising since the adjustment is quite substantial. This could show that the model does not describe reality as well as expected, possibly due to a sign error or an implementation error in the model. This will however not be further investigated due to time constraints.

With the implemented controller, the system has become stable. Filters can now be designed to reduce sensor noise.

7 | Digital Filters

The signals from the encoders and sensors contain uncertainties and noise, which are disturbances in the system. One way to minimize the disturbances is to implement filters which attenuate the noise and measurement uncertainties. Generally, there are two types of filters that can be implemented either analog or digital filters. Analog filters work in continuous time and digital filters in discrete time. Where the continuous time filter has a particular value of a signal at any given time, the discrete time only has defined values at specific points in time. These points are called samples and the time interval between each sample is the sampling period T_s with a sampling frequency $f_s = \frac{1}{T_s}$. Since the hardware platform of the segway is provided, it might be inconvenient to extend the platform with additional hardware, i.e. if a high order filter is needed, more components are needed to realize the filter. Another issue is to design and implement analog filters, as it is not as flexible as a digital filter, since changing the filter characteristic requires changing components. A disadvantage of digital filters is the demand of computational power to realize the filters. However since a digital filters are more flexible to work with, digital filters are chosen to be implemented. Two digital filters will be designed and three implemented, since the segway has two encoders and one angle measurement. The chapter is divided into the following sections:

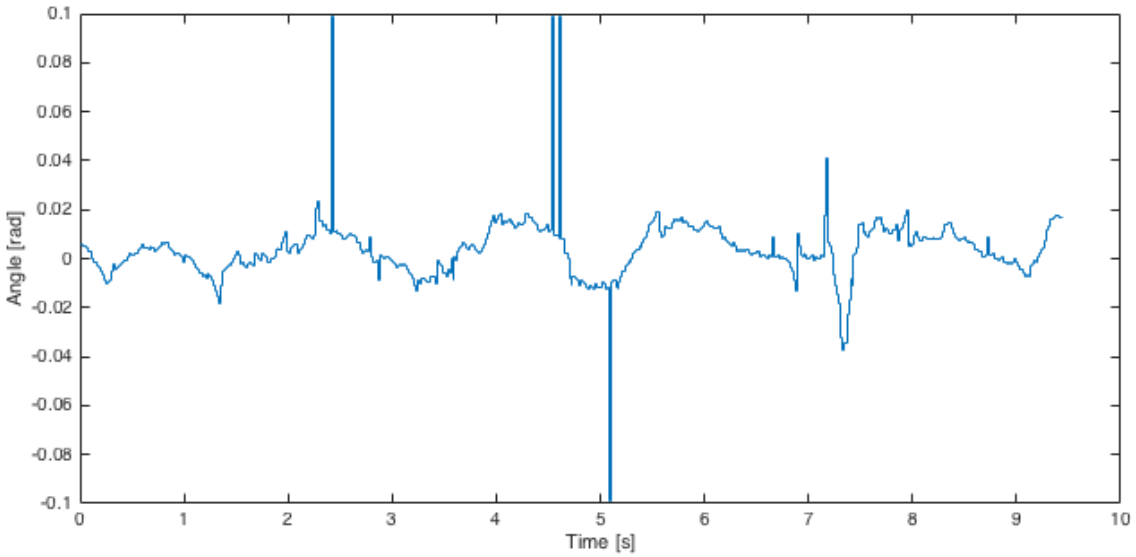
- **Spectrum analysis**
  This section analyse the measurements from the encoders and the angle derived from the gyroscope and accelerometer sensors. The encoders for both wheels are used for measuring the speed while the gyroscope and accelerometer sensors are used to obtain the angle.
- **Requirements and specifications for the digital filters**
  From the spectrum analysis it is possible to set up requirements and specifications for the digital filters.
- **Designing the digital filters**
  The requirements and specification are in this section used to design the digital filters.
- **Implementation of the digital filters**
  The digital filters from previous section are implemented in the microcontroller in this section.

The first part of this chapter is the spectrum analysis, where the requirements and specifications are derived.

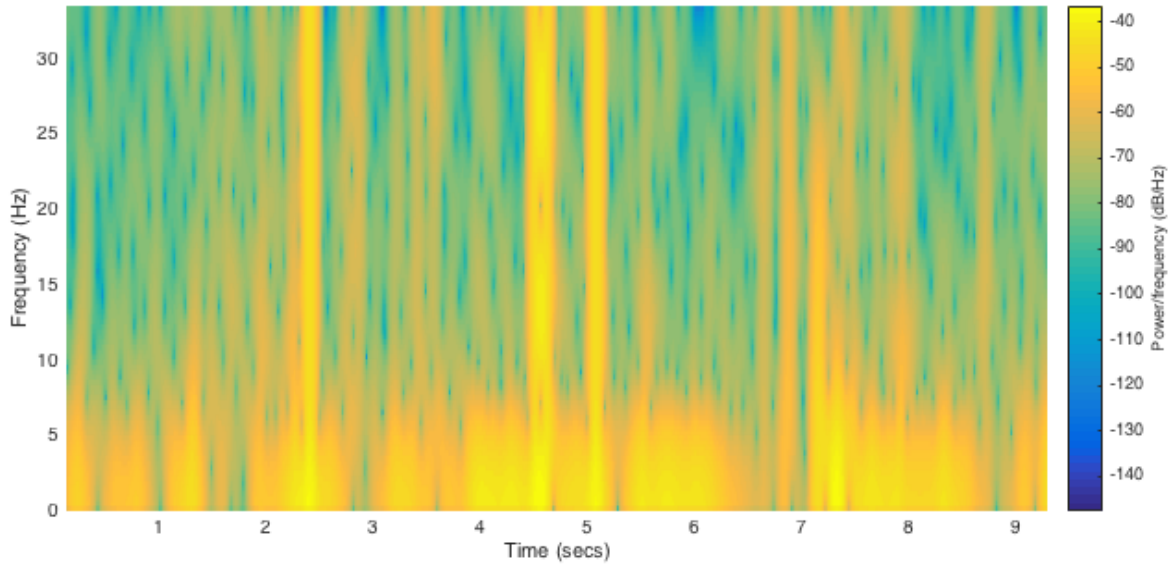
7.1 Spectrum Analysis

Before designing the filters, the requirements for the digital filters need to be specified. It is decided that digital filters will be implemented on both the encoders and the sensors for the angle. Requirements and specifications will be based on a spectrum analysis of the measurements

from the encoders and these sensors. In this chapter the sensors used for calculating the angle, will be referred to as "sensors". The first analysis is based on the measurements from the sensors measured with the controller from a previous section implemented. The sampling period T_s is set to 15 ms and the output plotted in Figure 7.1a is the angle derived from the sensors.



(a) Angle measurements. The measurements are taken while the segway stabilises itself in an upright position with the controller from previous section implemented.



(b) Spectrogram of the angle measurements. The sampling frequency f_s for both the gyroscope and the accelerometer is 66.67 Hz, window size is 20 samples and overlapping is 19 samples.

Figure 7.1: Measured angle and spectrogram of the measured angle.

Figure 7.1a shows the angle when the segway, from upright position, is stabilising itself. Error

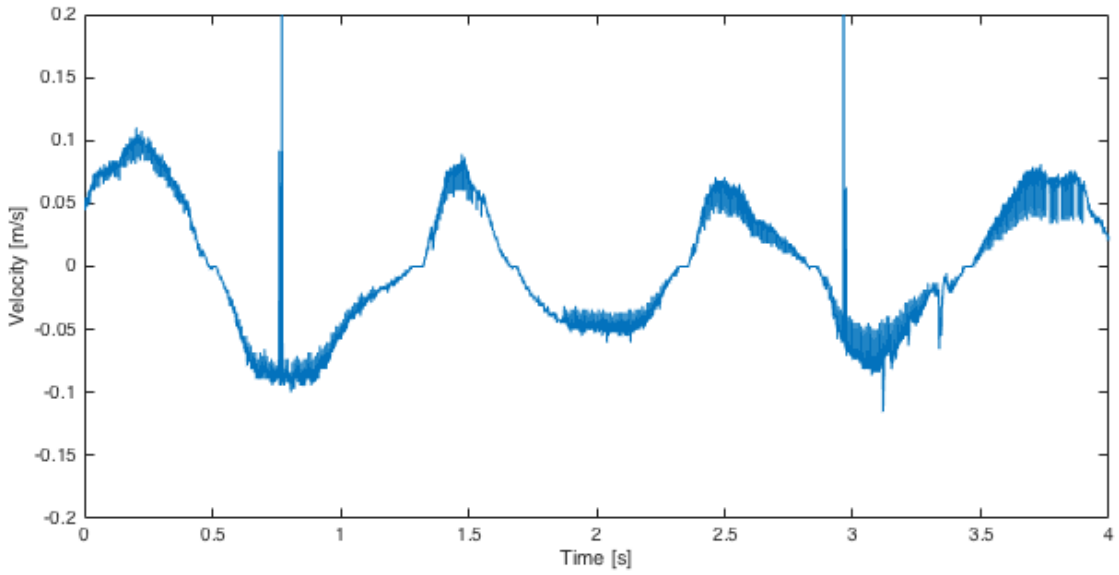
measurements occur when the angle suddenly increases or decreases, see Figure 7.1a at 2.4 seconds, and attenuating these errors are desirable. From the angle measurements, it is seen that the segway moves forward and backwards with a period on approximately two seconds. After seven seconds the segway quickly moves forward and backwards, because a disturbance might have caused it to overreact. Designing a filter to the angle measurements, the attenuation must not be too high at frequencies where the segway overreacts, which for instance occurs at about 7 seconds in the angle measurements. Another side effect is the measurements being delayed, which is not desirable in a control system. Therefore the specifications for the filters need to be set carefully. To determine the specifications for the passband and stopband frequencies, a spectrum analysis is needed.

The measurements from Figure 7.1a are now spectrum analysed in the spectrogram, see Figure 7.1b. The spectrogram is generated from a MATLAB function which calculates multiple Discrete Fourier Transformations (DFT) of a signal. The spectrogram shows the spectrum analysis of the measurements. Because the measurements are time discrete the DFT is used instead of the Continuous Fourier Transformation (CFT). The spectrogram consists of multiple processed DFTs. The number of DFTs is determined by the window size of each DFT and the overlapping for each DFT. The window size is the length of the signal that is processed and the overlap is how much of the signal from the previous DFT is included in the next DFT. The window size is set to 20 samples, the overlap to 19 to increase the resolution of the spectogram. The sampling frequency of the measurements is 66.67 Hz.

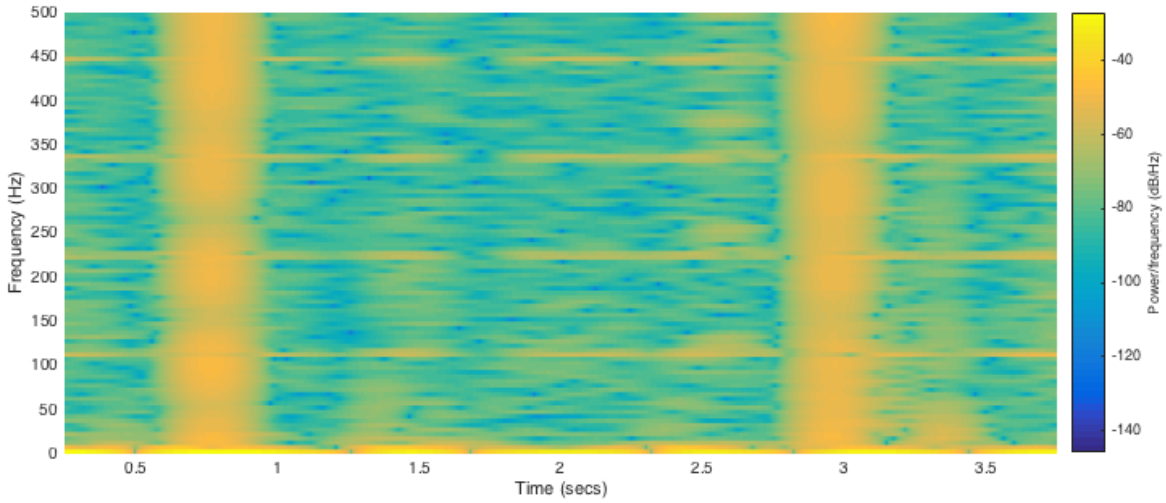
A low-pass filter can be implemented to attenuate noise above 15 Hz. This is derived from comparing the amplitude plot of the measurements with the spectrogram between the 7th and 8th second. It is approximated that most of the valid measurements are between 0 to 15 Hz. To remove noise and measurement errors the cutoff frequency is set to 15 Hz.

7.1.1 Spectrum Analysis for Encoders

The specifications of the digital filter for the encoders are derived from a spectrum analysis of measurements as well with the controller implemented. However the measurements are made independently from the previous measurement for the angle sensors. The measurements are made with a sampling frequency, f_s, of 1 kHz and are seen in Figure 7.2a. The speed of the motors is in- and decreasing with a period of approximately 1 Hz, which suggests that the segway is moving forward and backwards to stabilise itself. Since it is not desirable to attenuate valid measurements, the attenuation at the passband frequency Ω_p is set to maximum 1 dB at 5 Hz.



(a) Velocity measured from the encoders. The encoders measure the rotational speed of the motors in m/s.



(b) Spectrogram of the encoder measurements. The sampling frequency f_s is 1000 Hz, window size is 500 samples, and overlapping is 499 samples.

Figure 7.2: Measured velocity and spectrogram of the measured velocity.

From the spectrum analysis of the encoders and sensors, it is now possible to set the requirements and specifications for the digital filters. Since many of the specifications are estimated, it requires design and implementation iterations, until the filters both attenuate noise and measurement uncertainties, but without affecting the system too much, making it unstable.

7.2 Requirements and Specifications for Digital Filters

From the spectrum analysis in the previous section the requirements and specifications for the digital filters may be specified. Both filters need to be low-pass filters. The chosen filter type for the low-pass filters are Butterworth filters, since they have the desired frequency response i.e. no passband ripple and high attenuation at high frequencies.

Two of the main issues when determining the specifications for the digital filters are attenuation and delaying of valid measurements. It should be avoided in a control system to attenuate and delay the measurements, since this will disturb the control system. To minimize the disturbances introduced by the digital filters, the attenuation must not be high at lower frequencies where valid measurements are present. The cutoff frequency should also be moved one decade higher to minimize the phase shift.

Encoder Specifications

- The attenuation at 5 Hz is max 1 dB.
  The 5 Hz attenuation is estimated from the spectrum analysis, since the filter should not affect lower frequencies.
- The attenuation at 60 Hz is 80 dB.
  The 80 dB attenuation at 60 Hz is approximated, since there is no specific requirement regarding the stopband frequency.

The Nyquist frequency is 500 Hz since the Nyquist rate is defined as[**?**, p. 170]:

$$\frac{2\pi}{T_s} = 2\Omega_N \Leftrightarrow \Omega_N = \frac{2\pi f_s}{2} \tag{7.1}$$

Where:

T_s	is the sampling period	[s]
Ω_N	is the Nyquist frequency	[Hz]
f_s	is the sampling frequency	[Hz]

The highest frequency that can be represented is 500 Hz because of the Nyquist frequency. Higher frequency cannot be represented due to aliasing. Since 500 Hz corresponds to π, a frequency of 5 Hz for instance is equal to $\frac{5}{500}\pi = \frac{1}{100}\pi$. The specifications for the digital filter can be rewritten as:

$$0.89125 \leq |H(e^{j\omega})| \leq 1, \qquad 0 \leq |\omega| \leq \frac{1}{100}\pi \tag{7.2}$$

$$|H(e^{j\omega})| \leq 0.01, \qquad \frac{3}{25}\pi \leq |\omega| \leq \pi \tag{7.3}$$

Where:

ω	is the discrete time frequency	[Hz]

The specifications for the sensors may be derived like the filter for the encoders. However the sampling frequency is 66.67 Hz and the specifications, in contrast to the specifications for the encoders, will be chosen to.

Sensor Specifications

In contrast to the specifications for the encoders, the specifications for the angle sensors will be based on a cutoff frequency and a fixed filter order.

- Cutoff frequency at 15 Hz or $\omega_c = 0.45\pi$.
  As stated in previous section, the cutoff frequency is set to 15 Hz.
- Filter order is $N = 3$.
  The filter order is approximated to 3, to limit the amount computations by the microcontroller.

An issue with implementing digital filters in a small embedded system, like the microcontroller, is the limited computational power. Higher order digital filters have to process more data compared to lower order filters. This cause is explained in the implementation section. Instead of specifying the attenuation at specific frequencies, it is easier to minimize the amount of computation by setting the order $N = 3$ and the cutoff frequency $\omega_c = 0.45\pi$. Another issue is the phase response which introduces a delay at specific frequencies in the measurements. For a continuous time filter, a phase-shift will occur one decade before the cutoff frequency. The amount of phase shifted per decade depends on the filter order. A higher order filter will result in more phase shifting than a lower order.

Considerations about implementing the filters as either an Infinite Impulse Response (IIR) filter or a Finite Impulse Response (FIR) for the digital filters, is also important. However this chapter will not discuss the detailed difference between the two digital filter types. Implementing FIR filter requires generally higher order filters to be realized compared to IIR filters[**?**]. The amount of computation needed, is thus lower for the IIR filter and therefore the IIR filter is chosen.

7.3 Designing Low-pass Butterworth Digital Filter

The purpose of this section is to discuss different design methods and techniques. The first part is an analysis of the impulse invariance and bilinear transformation method for transformation a continuous time filter into a discrete time filter. Afterwards, a prototype of the continuous time filter will be designed and transformed into a discrete time filter using one of the transformation methods.

7.3.1 Impulse Invariance and Bilinear Transformation

When designing digital IIR filters, there are two methods for transforming a continuous time filter into a discrete time filter. The first method is impulse invariance and the second is bilinear transformation. Using the impulse invariance method, the impulse response of a continuous time filter is sampled into a discrete time filter. This necessarily results in the impulse response of the continuous time filter being proportional with the discrete time filter [**?**, p. 522]:

$$h[n] = T_s h_c(nT_s) \tag{7.4}$$

Where:

$h[n]$	is the discrete time impulse response	[1]
$h_c(nT_d)$	is the continuous time impulse response	[1]
T_s	is the sampling period	[s]

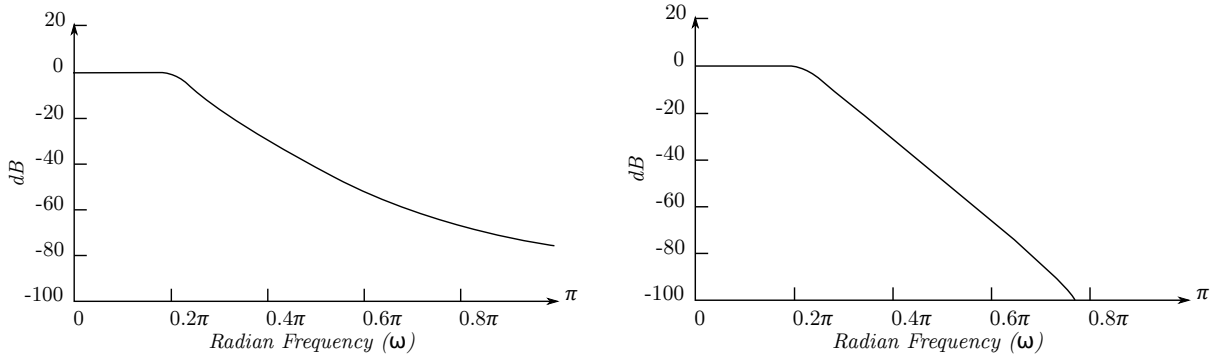
As seen in Equation 7.4, the impulse response of the discrete time filter is the sampled version of the continuous time filter, thus it is possible to keep the characteristic of the impulse response when transformed from s-plane to z-plane. Also, the frequency axis for the discrete time filter is scaled by a factor of T_s compared to the continuous time filter[**?**, p. 522]:

$$\omega = \Omega \cdot T_s, \qquad |\omega| \leq \pi \tag{7.5}$$

Where:

Ω	is the continuous time frequency	[rad/s]

The frequency response for a discrete time filter using impulse invariance is shown in Figure 7.3a. An issue with impulse invariance is aliasing, since the frequency response extends to the area $\pi \leq |\omega| \leq 2\pi$ as seen in Figure 7.4 which is undesirable, since this introduces aliasing distortion.



(a) Frequency response using impulse invariance method[**?**].

(b) Frequency response using bilinear transformation[**?**].

Figure 7.3: Comparison between the impulse invariance method and bilinear transformation.

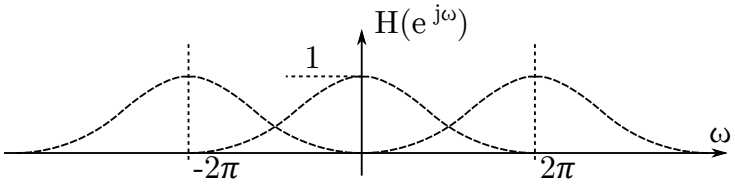


Figure 7.4: Aliasing when using the impulse invariance transform.

The second method for transforming from the s-plane to the z-plane is the bilinear transformation. The design approach is to design a continuous time filter and then map the imaginary $j\Omega$-axis of the s-plane into the unit circle of the z-plane [**?**, p. 528]. This means, that all frequencies in continuous time, $-\infty \leq \Omega \leq \infty$, are mapped into $-\pi \leq \omega \leq \pi$ in discrete time as shown in Figure 7.5.

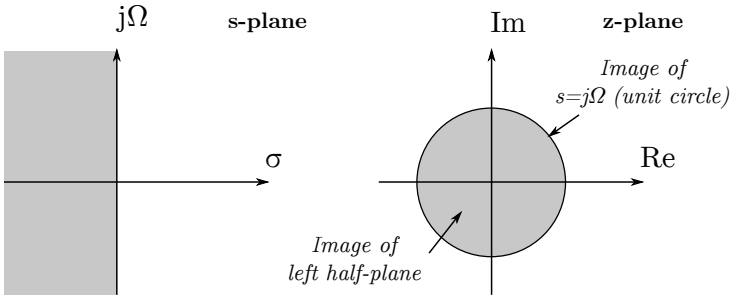


Figure 7.5: Mapping the s-plane into z-plane[**?**, p.528]. The left side of the of s-plane is mapped within the unit circle of the z-plane, while the $j\Omega$-axis is mapped onto the unit circle.

When bilinear transform is used, the relation between the frequency axis for the continuous time filter and the frequency axis for the discrete time filter, is no longer linear compared to the impulse invariance. This is shown in Figure 7.6.

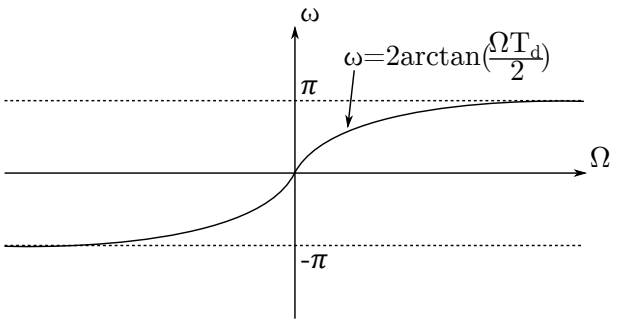


Figure 7.6: Frequency wrapping between continuous time and discrete time [**?**, p.530].

The relation between the frequency axis for the continuous time and discrete time filter is given

as:

$$\omega = 2 \cdot \arctan(\frac{\Omega T_s}{2}) \Leftrightarrow \Omega = \frac{2}{T_s} \tan(\frac{\omega}{2}) \tag{7.6}$$

Comparing Equation 7.6 to Equation 7.5, where the frequency axis ω has a linear scaling, the ω-axis when using the bilinear transformation is not using a linear scaling. As a result, the frequency response of the digital filter is shown in Figure 7.3b. For the bilinear transformation there is no aliasing problem, and the filter keeps its characteristics. It has been chosen that the bilinear transformation will be used to transform from continuous time to discrete time.

7.3.2 Designing Low-pass Filter for Encoders

The digital filter for the encoders will now be designed. To determine the order, N, and the cutoff frequency, Ω_c, of a Butterworth filter, the magnitude-squared function for a Butterworth filter is used[**?**, p. 525] to derive N and Ω_c.

$$|H(j\Omega)|^2 = \frac{1}{1 + (\frac{\Omega}{\Omega_c})^{2 \cdot N}} \tag{7.7}$$

Where:

Ω	is the frequency with the $\|H(j\Omega)\|$ attenuation	[rad/s]
Ω_c	is the cutoff frequency	[rad/s]
N	is the order of the filter	[1]

By using the values from the specifications for the encoders and sensors the order and cutoff frequency can be solved. However since the filter is designed in continuous time, the frequencies specified from the specification needs to be transformed into continuous time. To do so, the frequency is warped from discrete time to continuous time using frequency warping [**?**, p. 529], where the s component is substituted with an expression for z. Frequency warping is given as:

$$\Omega = \frac{2}{T_s} \cdot \tan(\frac{\omega}{2}) \Leftrightarrow \Omega = 2 \cdot f_s \cdot \tan(\frac{\omega}{2}) \tag{7.8}$$

Frequency warping is now applied to the frequencies from the specification:

$$\Omega_p = 2 \cdot 1000 \cdot \tan(\frac{\frac{1}{100}\pi}{2}) = 31.418\,\text{rad/s} \tag{7.9}$$

$$\Omega_s = 2 \cdot 1000 \cdot \tan(\frac{\frac{3}{25}\pi}{2}) = 381.52\,\text{rad/s} \tag{7.10}$$

Where:

Ω_p	is the passband frequency	[rad/s]
Ω_s	is the stopband frequency	[rad/s]

The passband frequency and stopband frequency from Equation 7.10 are inserted in equation Equation 7.7 yielding:

$$0.89125^2 = \frac{1}{1+(\frac{31.418\text{rad/s}}{\Omega_c})^{2\cdot N}}, \quad 0.0001^2 = \frac{1}{1+(\frac{381.52\text{rad/s}}{\Omega_c})^{2\cdot N}} \tag{7.11}$$

The equations are solved with respect to N and Ω_c.

$$N = 3.96 \quad \text{and} \quad \Omega_c = 37.26\,\text{rad/s} \tag{7.12}$$

The result yields that the minimum order of the filter is $N = 4$ and the cutoff frequency is $\Omega_c = 37.26\,\text{rad/s}$. It is important to state that since it is a 4th order filter, a phase shift of 180° will occur at the cutoff frequency, which is undesirable. To minimize the phase shift at 37.26 rad/s, the cutoff frequency must be increased by one decade to $\Omega_c = 372.6\,\text{rad/s}$. This trade-off ensures a low phase shift at 37.26 rad/s but lowers the amount of attenuation.

The next step is to determine the pole locations of the filter. The poles for a Butterworth filter can be found as [**?**, p. 1041]:

$$p_k = \Omega_c \cdot e^{\frac{j\pi}{2N}\cdot(2k+N-1)}, \quad k = 0, 1, ..., 2N-1 \tag{7.13}$$

Where:

p_k	is the pole location	[1]

When inserting the values of k and N, the location of the poles can be found. However since a stable filter is desired, only the poles located in the left half plane of the s-plane are used. Since the order of the filter is $N = 4$, k is equal to 7. The pole locations are now found.

$$p_1 = \Omega_c \cdot e^{j\frac{5}{8}\pi} \tag{7.14}$$

$$p_2 = \Omega_c \cdot e^{-j\frac{5}{8}\pi} \tag{7.15}$$

$$p_3 = \Omega_c \cdot e^{j\frac{7}{8}\pi} \tag{7.16}$$

$$p_4 = \Omega_c \cdot e^{-j\frac{7}{8}\pi} \tag{7.17}$$

It is now possible to obtain the general transfer function of the continuous time filter from the pole locations. The general form of the transfer function for a Butterworth filter is given as:

$$H_c(s) = \frac{G}{\prod_{k=1}^{N}(s-p_k)} \tag{7.18}$$

Where:

G	is the gain of the transfer function	[rad/s]
$H_c(s)$	is the continuous time transfer function	[rad/s]

Since the order of the filter is even, no poles are located on the real axis, and all poles have a pole pair because of the symmetric placement of the poles around the imaginary axis [**?**, p.

1041]. Due to the symmetry, it is possible to write p_2 as p_1 conjugated which is noted as p_1^*. This applies as well for p_3 and p_4.

$$H_c(s) = \frac{G}{(s-p_1)\cdot(s-p_1^*)\cdot(s-p_3)\cdot(s-p_3^*)} = \frac{G}{(s^2-p_1s-p_1^*s-p_1p_1^*)\cdot(s^2-p_3s-p_3^*s-p_3p_3^*)} \tag{7.19}$$

The poles from Equation 7.17 are inserted in Equation 7.19.

$$H_c(s) = \frac{G}{(s^2-\Omega_c e^{j\frac{5}{8}\pi}s-\Omega_c e^{-j\frac{5}{8}\pi}s+\Omega_c^2 e^{j\frac{5}{8}\pi}e^{-j\frac{5}{8}\pi})\cdot(s^2-\Omega_c e^{j\frac{7}{8}\pi}s-\Omega_c e^{-j\frac{7}{8}\pi}s+\Omega_c^2 e^{j\frac{7}{8}\pi}e^{-j\frac{7}{8}\pi})} \tag{7.20}$$

Because of the conjugated pole pairs, the mathematical rules $e^{j\omega}e^{-j\omega}=1$ and $e^{j\omega}+e^{-j\omega}=\cos(\omega)$ are applied to Equation 7.20, which simplifies the transfer function.

$$H_c(s) = \frac{G}{(s^2-2\Omega_c\cos(\frac{5}{8}\pi)s+\Omega_c^2)\cdot(s^2-2\Omega_c\cos(\frac{7}{8}\pi)s+\Omega_c^2)} \tag{7.21}$$

The final step is to determine the DC gain, G. The components are set to zero because a DC gain has a frequency of 0.

$$|H_c(s)|\Big|_{s=0} = 1 = \frac{G}{\Omega_c^4} \tag{7.22}$$

$$G = \Omega_c^4 \tag{7.23}$$

The transfer function of the filter in continuous time is now derived. The frequency response of the continuous time filter is now plottet:

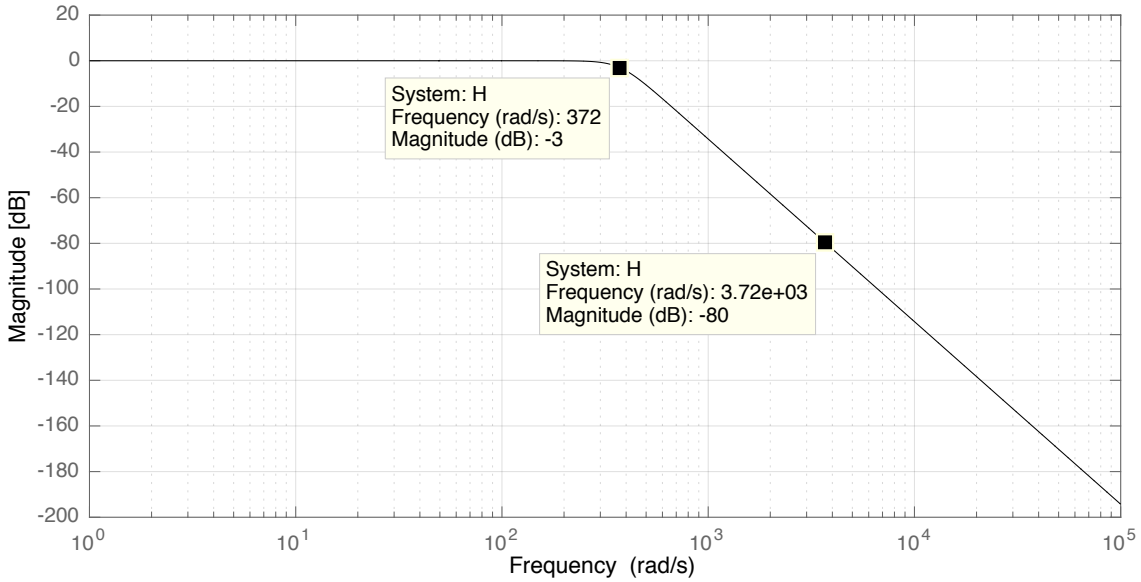


Figure 7.7: Frequency response of continuous time filter for the encoders.

As seen in Figure 7.7 the continuous time filter has the desired cutoff, pass band, and stop band frequency and complies with the minimum requirements. The continuous time filter will now be transformed into discrete time.

By using the bilinear transformation, the s-plane are mapped into the z-plane. The expression for the bilinear transformation is given as [**?**, p. 528]:

$$s = \frac{2}{T_s} \cdot \frac{z-1}{z+1} \tag{7.24}$$

Applying the bilinear transformation on the transfer function yields:

$$H_{enc}(z) = \frac{\Omega_c^4}{((\frac{2}{T_d}\frac{z-1}{z+1})^2 - 2\Omega_c \cos(\frac{5}{8}\pi)\frac{2}{T_d}\frac{z-1}{z+1} + \Omega_c^2) \cdot ((\frac{2}{T_d}\frac{z-1}{z+1})^2 - 2\Omega_c \cos(\frac{7}{8}\pi)\frac{2}{T_d}\frac{z-1}{z+1} + \Omega_c^2)} \tag{7.25}$$

Where:

$H_{enc}(z)$	is the discrete time transfer function of the digital filter for the encoders	[Hz]

The values for the variables are inserted and the expression of $H_d(z)$ is afterwards reduced to a standard form where the first denominator coefficient, a_0 is equal to 1 [**?**, p. 456]. The form is more efficient in the matter of implementation, which will be discussed later.

$$H_{enc}(z) = \frac{\sum_{k=0}^{M}(b_k z^{-k})}{1 - \sum_{k=1}^{N}(a_k z^{-k})} \tag{7.26}$$

Reducing the expression from Equation 7.25 to the form in Equation 7.26, the transfer function for the digital low-pass Butterworth filter is:

$$H_{enc}(z) = \frac{0.000742 + 0.002969z^{-1} + 0.00445z^{-2} + 0.002969z^{-3} + 0.000742z^{-4}}{1 - 3.039809z^{-1} + 3.554178z^{-2} - 1.881894z^{-3} + 0.379401z^{-4}} \tag{7.27}$$

The magnitude of the frequency response is now plotted to examine if the frequency response has the desired characteristic.

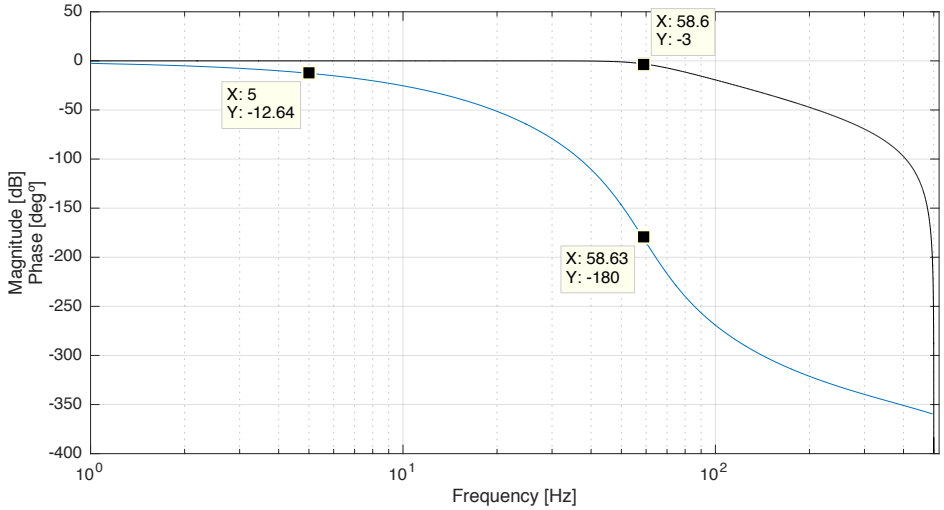


Figure 7.8: Frequency response of the 4th order low-pass discrete digital filter for the encoders. The black line is the frequency response and the blue line is the phase response.

As shown in Figure 7.8 the magnitude frequency response has the desired characteristics with 3 dB attenuation at 58.6 Hz and only -12.64° phase shift at 5 Hz compared to 180° phase shift at the cutoff frequency.

7.3.3 Low-pass Filter for Sensors

Designing the low-pass Butterworth filter for the sensors is similar to the filter for the encoders. However instead of determining the cutoff frequency and order, they are set to 15 Hz and 3rd order as stated in the requirement and specification section. After applying frequency warping the cutoff frequency is increasing by one decade to minimize the phase shift at 15 Hz. The cutoff frequency is determined to be $\Omega_c = 1138.7\,\text{rad/s}$. Since the order is known, the poles of the 3rd order low-pass filter is determined by Equation 7.13.

$$p_1 = \Omega_c \cdot e^{j\frac{2}{3}\pi} \tag{7.28}$$

$$p_2 = \Omega_c \cdot e^{-j\frac{2}{3}\pi} \tag{7.29}$$

$$p_3 = \Omega_c \cdot e^{j\pi} = -\Omega_c \tag{7.30}$$

The poles are now inserted in Equation 7.18. Since one of the poles is on the real axis, the pole does not have a complex conjugated pole pair around the real axis.

$$H_c(s) = \frac{G}{(s+\Omega_c)\cdot(s^2 - 2\Omega_c\cos(\frac{2}{3}\pi)s + \Omega_c^2)} \tag{7.31}$$

The DC gain is determined by setting s to zero. The gain is then derived to be Ω_c^3.

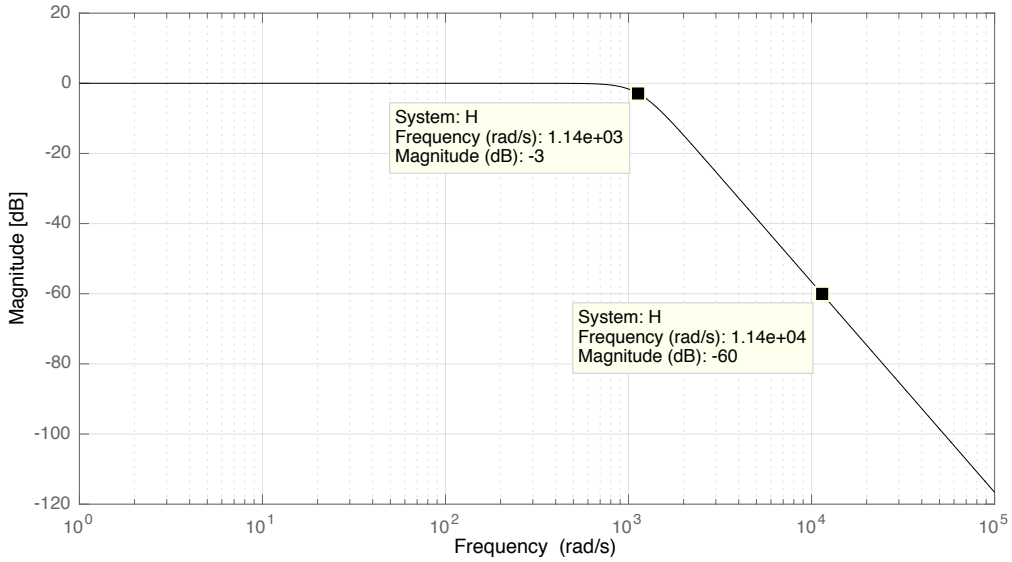


Figure 7.9: Frequency response of continuous time filter for the sensors.

The filter has the desired cutoff frequency, since the calculated frequency warped cutoff frequency was determined to $\Omega_c = 1138.7\,\text{Hz}$. To transform the continuous time filter into discrete, the bilinear transform is used with the inserted cutoff frequency. Afterwards the expression is reduced to the standard form seen in Equation 7.26.

$$H_{sen}(z) = \frac{0.790712 + 2.37213z^{-1} + 2.37213^{-2} + 0.790712z^{-3}}{1 + 2.53247z^{-1} + 2.16800z^{-2} + 0.625224z^{-3}} \tag{7.32}$$

Where:

$H_{sen}(z)$	is the discrete time transfer function of the digital filter for the encoders	[Hz]

Plotting the frequency response of the transfer function, reveals that the cutoff frequency is located at 31 Hz with a phase shift on 135°. The phase shift at 15 Hz is 11.46°.

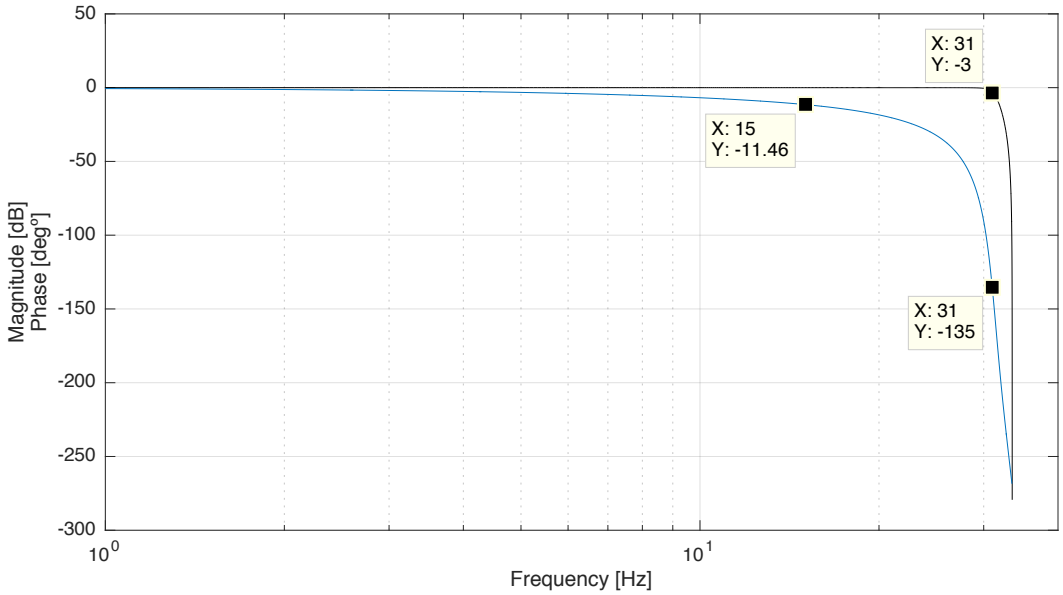


Figure 7.10: Frequency response of the 3rd order low-pass digital filter for the sensors. The black line is the frequency response and the blue line is the phase response.

In the next section an analysis of how to implement the derived transfer functions of the digital filters in a microcontroller follows.

7.4 Implementation of Digital Filters

The derived transfer functions for filters need to be implemented in a microcontroller. Since the microcontroller only works in time domain where the input of the system is $x[n]$ and output $y[n]$, the filters need to be transformed back to discrete time domain from the z-domain. The primary purpose of this section is to analyse how to do the inverse z-transformation and implement the filters efficiently.

In discrete time domain, the output of a system is given as the convolution of the input with the system impulse response:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = x[n] * h[n] \tag{7.33}$$

When applying the z-transform in Equation 7.33, the convolution is substituted with a multiplication in the z-domain.

$$y[n] = x[n] * h[n] \overset{Z}{\leftrightarrow} Y(z) = H(z)X(z) \tag{7.34}$$

Where:

$y[n]$	is the discrete time output	[1]
$x[n]$	is the discrete time input	[1]

$h[n]$ is the discrete time impulse response [1]

The expression of $H(z)$ is previously found to be the discrete time transfer function of the digital filter which is written as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M}(b_k z^{-k})}{\sum_{k=0}^{N}(a_k z^{-k})} \Rightarrow \sum_{k=0}^{N}(a_k z^{-k})Y(z) = \sum_{k=0}^{M}(b_k z^{-k})X(z) \tag{7.35}$$

Since the coefficient, a_0, for both derived transfer functions are $a_0 = 1$, because Equation 7.26 are used to derive the transfer functions, the following difference equation may be written.

$$Y(z) = -\sum_{k=1}^{N}(a_k z^{-k})Y(z) + \sum_{k=0}^{M}(b_k z^{-k})X(z) \tag{7.36}$$

Because of the coefficient $a_0 = 1$, the equation may be written as Equation 7.36, where $k = 1$ for the first summation of a_k. If $a_0 \neq 1$ the difference equation should be divided by a_0 on both sides of the equation.

One of the properties of transforming between discrete time domain and the z-domain is the time-shifting property [**?**, p. 131]. The time-shifting property is basically that a factor of z^{-k} will correspond to a delayed sample in discrete time domain.

$$x[n-k] \overset{Z}{\leftrightarrow} z^{-k}X(z) \tag{7.37}$$

Equation 7.37 also applies for $Y(z)$. The inverse z-transform from Equation 7.34 are therefore now applied in Equation 7.36 with the time-shifting property from Equation 7.37. This yields the following difference equation of the output $y[n]$.

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k] \tag{7.38}$$

To determine the output of the filters, Equation 7.35 is first applied to the transfer function of the filter. Afterwards the expression is rewritten to the difference equation in Equation 7.36. Lastly the inverse z-transformation is used to transform the equation from z-domain to time domain with the end result seen in Equation 7.38.

When using the procedure above, for the transfer functions for the digital filters, the following difference equations are derived. For the encoders:

$$\begin{aligned} y_{enc}[n] = &-(a_1 y[n-1] + a_2 y[n-2] + a_3 y[n-3] + a_4 y[n-4]) + b_0 x[n] + b_1 x[n-1] \\ &+ b_2 x[n-2] + b_3 x[n-3] + b_4 x[n-4] \\ &\Downarrow \\ y_{enc}[n] = &-(-3.039809 y[n-1] + 3.554178 y[n-2] + (-1.881894) y[n-3] \\ &+ 0.079401 y[n-4]) + 0.000742 x[n] + 0.002969 x[n-1] \\ &+ 0.004453 x[n-2] + 0.002969 x[n-3] + 0.000742 x[n-4] \end{aligned} \tag{7.39}$$

Where:

$y_{enc}[n]$	is the discrete time output of the digital filters for the encoders	[1]

The same procedure is applied for the digital filter for the sensors.

$$\begin{aligned} y_{sen}[n] =& -(a_1 y[n-1] + a_2 y[n-2] + a_3 y[n-3]) + b_0 x[n] + b_1 x[n-1] \\ &+ b_2 x[n-2] + b_3 x[n-3] \\ &\Downarrow \\ y_{sen}[n] =& -(2.53247 y[n-1] + 2.16800 y[n-2] + 0.625224 y[n-3]) \\ &+ 0.790712 x[n] + 2.37213 x[n-1] + 2.37213 x[n-2] + 0.790712 x[n-3] \end{aligned} \tag{7.40}$$

Where:

$y_{sen}[n]$	is the discrete time output of the digital filters for the sensors	[1]

As the difference equation of the output $y[n]$ for both the encoders and sensors are derived, the next step is to realize the equations in the microcontroller.

7.4.1 Filter Structure Optimization and Code Implementation

There are different ways to implement the difference equation of the output $y[n]$ derived from the previous section. This section primarily discusses different filter structures to implement in the microcontroller. It is possible to implement Equation 7.39 and Equation 7.40 directly to the microcontroller. However doing so is not necessarily computational efficient for the microcontroller compared to other alternatives. The IIR filter structures are illustrated as block diagrams which eases the readability.

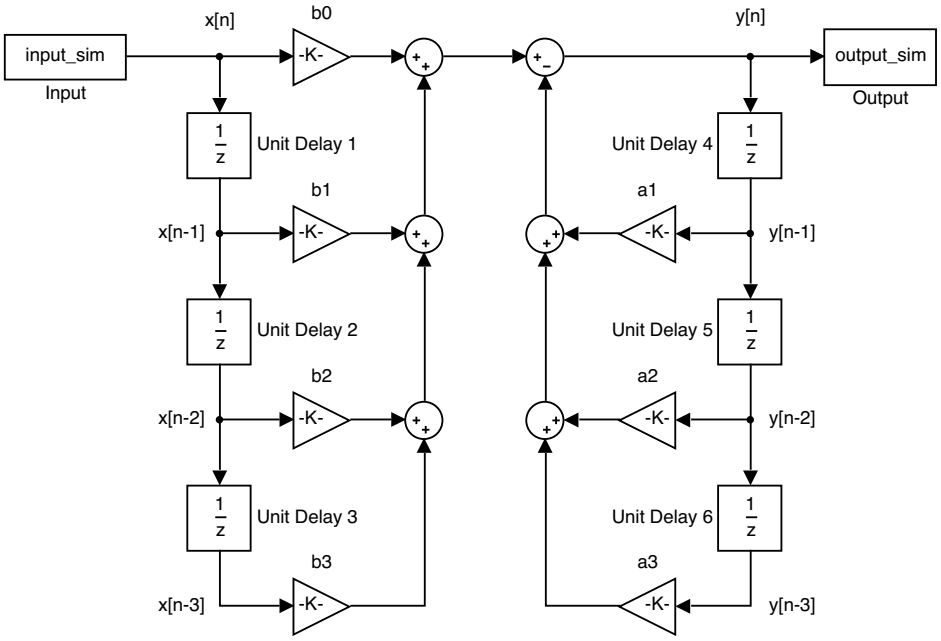


Figure 7.11: Filter structure of the IIR 3rd order digital filter for the sensors. There is a feedback loop at the output, which can make the filter unstable if not designed carefully.

If the difference equation from Equation 7.40 is implemented directly in the microcontroller, the filter structure of the equation is shown in Figure 7.11. The type of structure is called direct form I [**?**, p. 399] and implementing such a structure requires six variables to store the values of $x[n-1]$, $x[n-2]$, $x[n-3]$, $y[n-1]$, $y[n-2]$ and $y[n-3]$. An alternative way of implementing the filter is using direct form II. The output of this structure is equivalent to direct form I, but needs only four variables to store values. The filter for direct form II is seen in Figure 7.12.

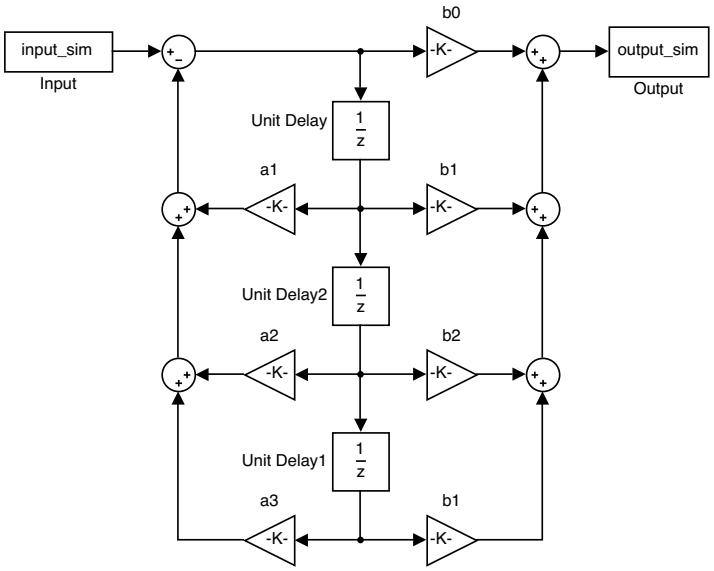


Figure 7.12: Filter structure of the 3rd order IIR filter for the sensors. Direct form II.

The direct form II is used for the digital filter for the sensors. For the direct form II, variables are inserted before and after every delay ending with a total number of variables of four. This saves memory and the processor spares at least two additional instructions for copying the value in a variable to another, corresponding to a delay. The code implementation of the filter structure is seen in Listing 7.2.

```
1  double IIR_LP_ANG(double angle){
2      static double BUF0, BUF1, BUF2, BUF3 = 0;   // Initialize the buffers of static doubles
3      static double angleFiltered = 0;            // Initialize the output variable
4
5      BUF0 = angle - (a1*BUF1 + a2*BUF2 + a3*BUF3); // Calculate the first buffer
6      angleFiltered = b0*BUF0 + b1*BUF1 + b2*BUF2 + b3*BUF3; // Calculate the output
7      BUF3 = BUF2; // Shift the buffer by 1 sample. Corresponds to a delay.
8      BUF2 = BUF1;
9      BUF1 = BUF0;
10
11     return angleFiltered;
12 }
```

Listing 7.1: Code implementation of 3th order low pass filter for the sensors. Direct form II.

The variables are initialized as static doubles, since values can be large and the values need to be preserved when the function is returned. The "static" for the angleFiltered is however done to prevent allocating memory every time the function is called. In the next two lines calculations for the filter are processed. The algorithm is derived from the filter structure in Figure 7.12. Afterwards the values in the variables are shifted, which correspond to a delayed sample.

The filter for the encoders needs to be implemented as well. However instead of implementing the filter using direct form II with four delays, the structure is rearranged into biquad cascade

form, which consists of two 2nd order low-pass filters in direct form II. The filter structure of the filter is seen in Figure 7.13.

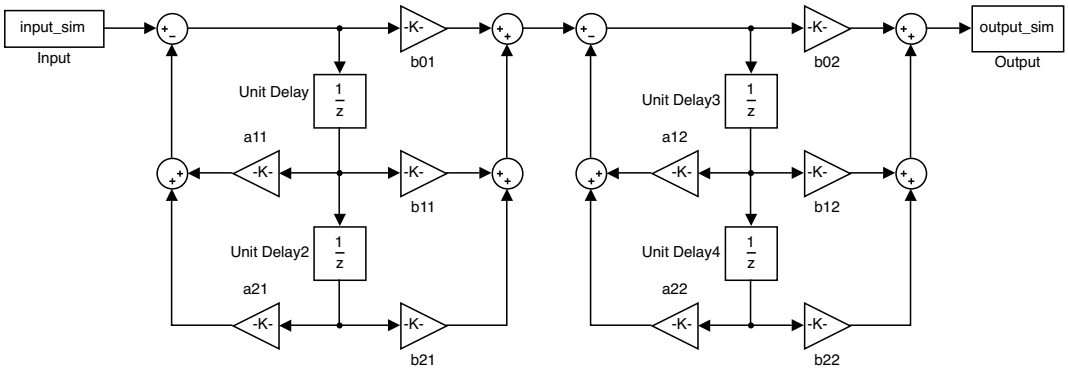


Figure 7.13: Filter structure of the 4th order IIR filter for the encoders.

Rearranging the filter structure from direct form I to biquad cascade direct form II requires that the transfer function is rewritten into the following form[**?**, p. 409]:

$$H_d(z) = \prod_{k=1}^{N_s} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}} \tag{7.41}$$

Since the order of the filter is 4, $N_s = 2$. Using Equation 7.41 the transfer function may be split into two 2nd order transfer functions.

$$\begin{aligned} H_{enc}(z) &= H_{enc1}(z)H_{enc2}(z) \\ &\Downarrow \\ H_{enc}(z) &= \frac{0.027244 + 0.054488z^{-1} + 0.027244z^{-2}}{1 - 1.400000z^{-1} + 0.50069z^{-2}} \cdot \frac{0.027244 + 0.054488z^{-1} + 0.027244z^{-2}}{1 - 1.639811z^{-1} + 0.757754z^{-2}} \end{aligned} \tag{7.42}$$

Where:

$H_{enc1}(z)$	is the 2nd order transfer function of the first biquad	[1]
$H_{enc2}(z)$	is the 2nd order transfer function of the second biquad	[1]

With the transfer function from Equation 7.42 the a_k and b_k coefficients are now derived. The procedure for implementing the filter is to use the coefficients and design the algorithm from the filter structure in Figure 7.13. The implemented code is shown in Listing 7.2.

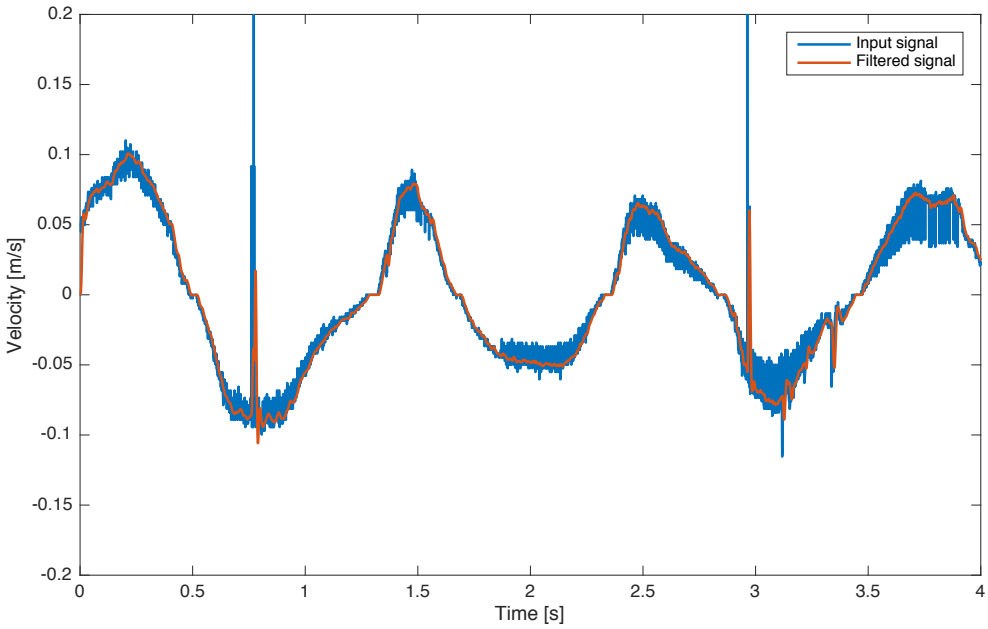
```
double IIR_LP_ENC(double vel){
    static double BUF01, BUF11, BUF21 = 0; // Initialize buffers for first biquad
    static double BUF02, BUF12, BUF22 = 0; // Initialize buffers for second biquad
    static double W, vel_filt = 0;      // W is the output for the first biquad

    BUF01 = vel - (a11*BUF11 + a21*BUF21); // Algorithm for first biquad
    W = b01*BUF01 + b11*BUF11 + b21*BUF21;
    BUF21 = BUF11;                      // Shift values in buffers
    BUF11 = BUF01;

    BUF02 = W - (a12*BUF12 + a22*BUF22); // Algorithm for second biquad
    vel_filt = b02*BUF02 + b12*BUF12 + b22*BUF22;
    BUF22 = BUF12;
    BUF12 = BUF02;

    return vel_filt;
}
```

Listing 7.2: Code implementation of 4th order low pass filter for the encoders.

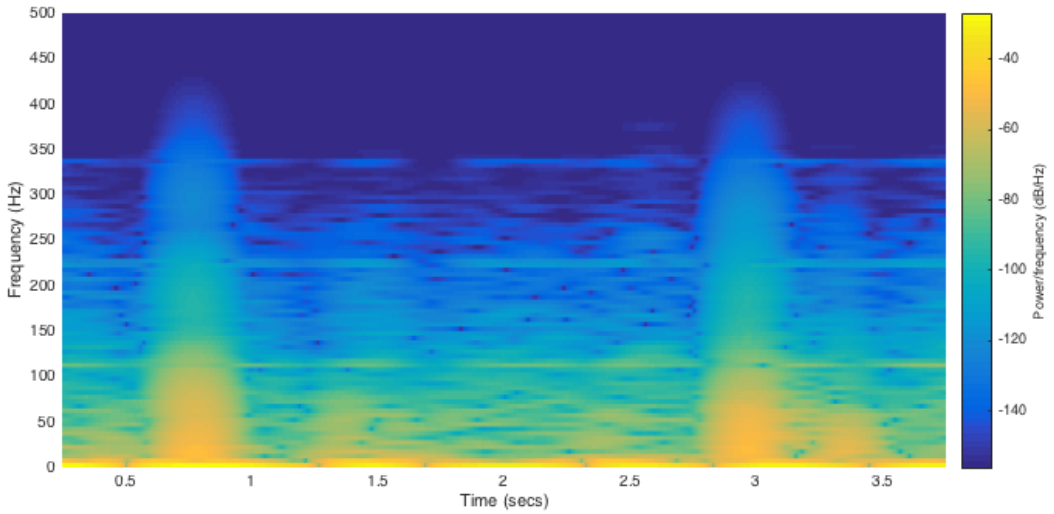
As shown in Listing 7.2 the implementation of the cascade form is to process two 2nd order filters in direct form II, where the output of the first filter, W, is the input of the next filter.

7.5 Results of Digital Filters

The Butterworth filters are now designed and can be implemented in the microcontroller. Before implementing the code in the microcontroller, the algorithm for the filters are applied for the measurements to see their impact on the measurements. In Figure 7.14a the comparison between the unfiltered and filtered signal from the encoder is shown. The digital filter attenuates measurements, which appears to be noise, greatly. The spectrogram in Figure 7.14b also shows the spectrum of the filtered signal. When comparing the spectrogram in Figure 7.14b and Figure 7.2b, it is seen that the filter attenuates signals above the cutoff frequency.



(a) Comparison between the original and the filtered encoder measurements. The noise is reduced greatly in the filtered version.



(b) Spectrogram of the filtered signal. Almost all frequencies above 300 Hz are filtered away.

Figure 7.14: Comparison of the unfiltered and filtered signal and spectrogram of the filtered signal.

A test shows that the segway is still able to stabilise itself in an upright position, after applying the filters on the segway. The digital filters for the encoders are therefore working as expected.

The digital filter for the angle sensors need to be applied on some measurements as well. The comparison of the unfiltered and filtered signal for the sensors are shown in Figure 7.15.

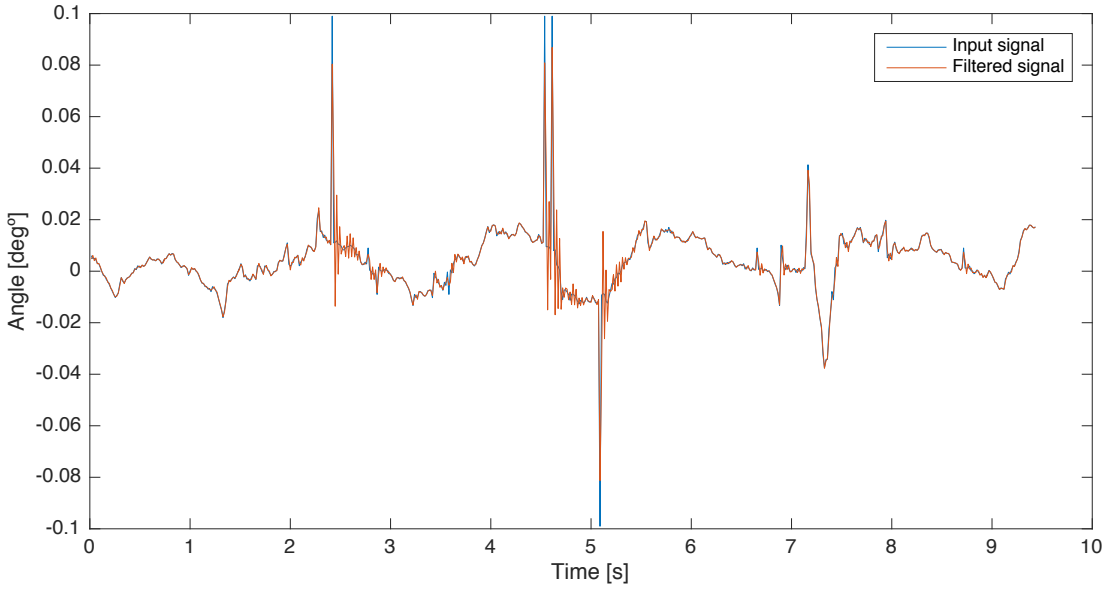


Figure 7.15: Comparison between the original and the filtered angle measurements.

As shown in Figure 7.15 the implemented digital filter does not improve the measurements very much. Therefore it is chosen to not implement the digital filter for the angle.

If the filters are implemented, they do not allow the wireless communication to be executed. Since the wireless communication is of higher priority in the project, the digital filters will not be implemented as they take up too much computation time. The design and implementation of the remote controller for the segway follows in the next chapter.

8 | Remote controller

To meet the requirement of being able to remote control the segway wirelessly and receive data from the segway wirelessly, see section 3.2, a remote controller is to be designed.
A wireless communication platform is needed, if data between the segway and a remote controller needs to be exchanged. As previously stated, it is desirable to remote control the segway, with respect to defining the speed and direction of the segway. Also, measurements from the segway should be possible to transmit to a remote controller where the state of the segway can be monitored. Figure 8.1 shows the data exchanges that are desired between the segway and the remote controller.

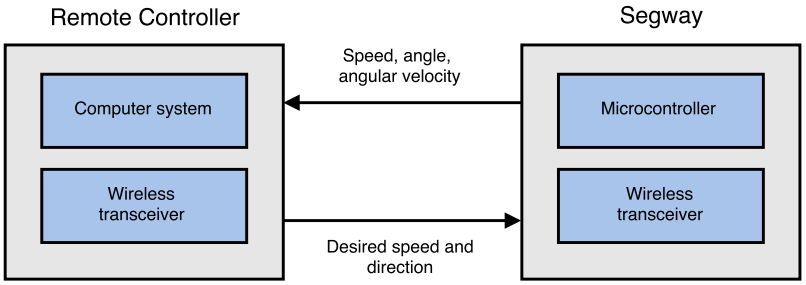


Figure 8.1: Overview of the communication link between the remote controller and the segway.

The remote controller should transmit the desired speed and desired direction when the user wishes to change these. To receive the current speed, inverted pendulum angle and angular velocity of the segway, the remote controller has to transmit a request to the segway, i.e. the communication will be request-based.

8.1 Requirements and Specifications for the Remote Controller

The topic of this chapter will primarily be the development of a communication protocol for data exchange between the segway and the remote controller. The requirements and specifications are thus only for the communication protocol. Based on the requirements set in section 3.2, the remote controller must be able to do the following through the communication protocol:

- Request the current speed, angle and angular velocity of the segway.
- Send the desired speed and direction to the segway.

Related to this, the segway must be able to:

- Reply to requests for the current speed, angle and angular velocity.
- Set the speed and direction of the segway when receiving a new reference.

In the following section, a communication protocol will be designed.

8.2 Design of Communication Protocol

To facilitate the communication between the remote controller (RC) and the segway, a communication protocol is needed. This protocol is to ensure that data is received correctly without errors, and ensures that the communication between the two devices has a certain set of rules to act within, so messages are not misunderstood.

It is decided to make the communication between the RC and segway request-based, meaning that the RC is to request some data from the segway before it is sent. This is done to reduce the computation time needed on the microcontroller on the segway for the communication, since the microcontroller will only have to handle the communication when a request is made.

8.2.1 The OSI-model

Before designing the communication protocol, the Open Systems Interconnection (OSI) model needs to be described, as this is used for identifying the functionalities of the protocol. The OSI model is used to describe various layers of a networking platform. The networking platform is in this case the communication between a segway and a remote controller. The OSI-model does not describe how the various communication protocols should be designed, but describes the purpose and function of each layer [**?**].

Application
Presentation
Session
Transport
Network
Data link
Physical

Table 8.1: The OSI model.

The OSI-model consists of seven layers, each describing the functionalities and responsibilities associated with the layer. The description of each layer will now be discussed whereupon the

design choice of each layer will be clarified.

Physical Layer

In the physical layer, the data transmission of physical signals are defined. The physical layer could concern electrical signals, frequencies or other physical sizes. The physical layer has been facilitated, since the APC220 modules used for the communication uses a wireless channel, with a frequency of about 434 MHz [**?**, p. 3]. These modules are capable of sending data using Gaussian Frequency Shift Keying (GFSK) modulation.

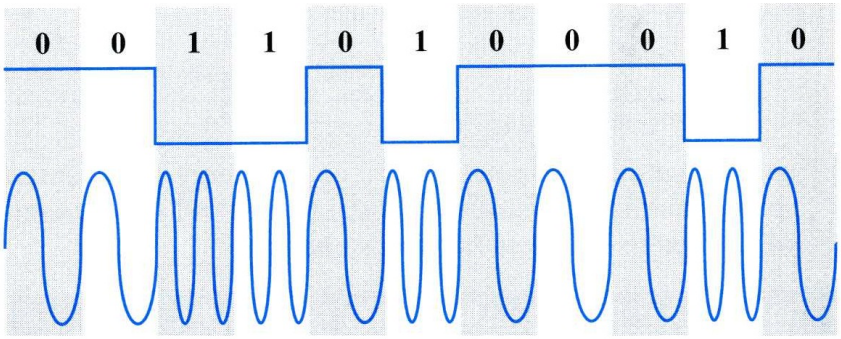


Figure 8.2: FSK modulation, where the frequency is shifted dependent on the binary data sent. The difference between GFSK and FSK is the gaussian filter applied on the GFSK [**?**].

GFSK means that the signal has a certain center frequency, in this case around 434 MHz, however, a slightly different frequency is used to transmit either a binary '0' or '1'. So, to send one binary data value, the center frequency is used, and to send the other binary data value, a different frequency is used. This frequency deviation can be changed in the APC220 modules, from 418 MHz to 455 MHz [**?**, p. 2]. An example of how GFSK data looks can be seen in Figure 8.2, where the data is shown on top, and the GFSK signal in the bottom. Gaussian Frequency Shift Keying is in this way similar to regular Frequency Shift Keying, FSK. The Gaussian in the name refers to a gaussian filter that is applied to the signal to smoothen the signal and thus reduce the rapid frequency changes seen in the FSK signal. This reduces the bandwidth of the signal, which is desired [**?**].

Data Link Layer

The data link layer provides services to the network layer. It is responsible for the link between nodes in the network, and also handles error control. As a part of this, the data link layer is responsible for allocating the channel to avoid collisions. In this case, collision avoidance and channel allocation are not implemented. This is done because the protocol is request based. Thus, data will not be sent by the RC and segway simultaneously, since the RC will request data, wait for a reply, and then request new data. The functionality regarding error control is however implemented in the protocol, in the form of a checksum on the data transmitted.

An alternative design approach, is to make the segway transmit regularly. In contrast to the

request based, the segway do not wait for a request but transmits data, for instance, every 10 ms. However this requires that segway must use more computation time at the communication compared to the request based approach. Therefore the request based approach is chosen to be designed.

Network Layer

The network layer is responsible for routing, i.e. sending packages from one hub in the network to another, to make it reach its destination. The network layer is also responsible for congestion control, i.e. making sure that the channel is not overloaded with data. The features of the network layer are not implemented in the protocol. This is because only two units are connected, and thus there is no reason to have routing.

Transport Layer

The transport layer is an end-to-end layer, meaning that units talking on the transport layer are not seeing the entire network, but only the other end. The transport layer features are somewhat implemented in the protocol, however it can be discussed if it is implemented in the transport layer or the application layer.

In the case of the internet, there are two typical transport layer protocols used, namely TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). The main difference between the two is that TCP is connection-based, i.e. a connection between hosts is set up, and then data can be sent back and forth. On the other hand, UDP is a connection-less protocol, meaning that it is individual packages that are sent back and forth. TCP has acknowledgements and handles retransmission, to ensure that all package are received correctly, whereas UDP does not facilitate this. This means that UDP is simpler than TCP, and also has less overhead. This can be seen based on the header size, where UDP has a header size of 8 bytes, whereas TCP has a header size of at least 40 bytes [**?**], meaning that for short conversations, UDP can be more efficient.

The protocol designed is somewhat similar to UDP. It is decided to make the protocol similar to UDP over TCP, to reduce the overhead, since the amount of data transmitted is not very big. Therefore, having a protocol with multiple handshakes and acknowledges is not desired, since it will require an undesirably large amount of computation from the segway.
Instead, a rather simple protocol with a small overhead is chosen, to minimize computation power and the amount of data sent on the channel. Another feature that is implemented in the transport layer is timeouts. The timeouts make sure that if a packet get lost under transmission, the segway or RC does not wait forever on the lost packet, but return after a set time without any response. The timeout is basically a timer that starts to count when the first byte of the packet is received. If the next packet is not received within the timeout, the function stops waiting for the next bytes and exits. Timeouts are seen as something that takes place in the transport layer, as this is the case for TCP, which will close the connection after a certain time with no data.

Session and Presentation layers

The session and presentation layers are not used in most applications of the OSI model, since their responsibilities are not used in typical networking applications. The same is the case here, and these layers are therefore not described further.

Application layer

The application layer is the layer that can be used directly by the user. Here, the application layer corresponds to the actual remote controller, and the segway. This layer should see the lower layers as a black-box, meaning that to the application layer, simple functions such as *RequestAngle()* and *sendDesiredAngle()* should be available, without the user (in this case the programmer) thinking about how the lower layers handle these functions.

8.2.2 Package design

Now that the OSI model has been described, the actual protocol design can begin. First, the structure of the data packages sent is determined.

It is decided that a package in the communication protocol designed consists of three overall fields: A header, the data, and a footer, consisting of a checksum. This can be seen in Figure 8.3.

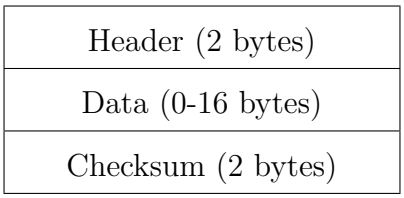


Figure 8.3: The overall contents of a package.

It is optional to include data in the package, depending on whether or not this is needed - an example of this will be shown later. If no data is included in the package, the checksum-footer is not transmitted either.

The package design is inspired by the UDP header, which can be seen in Table 8.2.

Byte offset	Bit 0	Bit 32
0	Source port (16 bits)	Destination Port (16 bits)
4	Length (16 bits)	UDP Checksum (16 bits)
8	Data (if any)	

Table 8.2: The UDP header & data field.

From Table 8.2, it is seen that the UDP header contains 4 fields, each with a length of 16 bits: A source, a destination, the data length and a checksum.

The contents of the header in the designed protocol can be seen in Figure 8.4. The header can be seen as consisting of two parts: The first byte containing flags, and the second byte containing the data length in bytes, i.e. one bit in the data size field represents one byte of data.

Field 1	Reply	v_s	θ_P	ω_P	θ_D	ϕ_D	Reserved for future use	Reserved for future use
Value	0	0	0	0	0	0	1	1

Field 2	Data size

Figure 8.4: The header contents of a single package. Field 1 is the first header byte, while field 2 is the second header byte.

The flags are somewhat similar to the destination and source fields in the UDP header, as they describe what the receiver, i.e. the segway, should do with the data, e.g. reply with the measured angle or set a new reference angle. The UDP length is also included in the protocol, by means of the second header byte. The UDP checksum is also included in the package, but is made as a footer instead as a part of the header. The checksum is therefore only added to the data, and not the header, to reduce the overhead, since the checksum is only sent if data is sent. This is a fair choice to make, because the risk of bit errors in the header is smaller than that on the data, since the header is only two bytes long.

The first bit of the header is a reply field. When the bit is set to '0' the RC transmits a request to the segway. The segway then performs the requested actions and afterwards responds by transmitting a packet to RC, where the first bit is set to '1'. The next three bits are flags, showing which data is either requested or not. The options are the segway speed, v_s, the segway tilting angle, θ_p and the angular velocity of the segway, ω_p. Each of these fields can be set, to indicate the data requested/transmitted. If for example the segway speed is requested, v_s, the second bit in the first header is set to '1'. When the segway is replying with the data values, the data size field is updated accordingly. The segway tilting angle, θ_p, and the angular velocity of the segway, ω_p, both have a size of 4 bytes, since the value transmitted is a float. The segway speed takes up 8 bytes, as it is two floats that are transmitted, namely the speed of both left and right motor. The values sent could be converted into a 16-bit integer, as all data in the microcontroller on the segway is sampled using at maximum 16 bit, to reduce the amount of data sent. However, for simplicity, the numbers are kept floats as this is the data type that the values are used as in the microcontroller.

The next two header bits are the desired speed θ_D, and the desired turning angle of the segway, ϕ_D. These fields are set by the RC, if the user wants to change these reference values for the segway, to either make the segway go forward, backwards or turn it. Each of these values have a size of 4 bytes, a float, and when transmitted, the data size field must be updated accordingly. Finally, the last two bits of the first header byte are not used, and are set to a '1'.

Looking at Figure 8.3, after the header is transmitted, which describes which data is sent and the length of that data, the actual data is transmitted. The data will have a maximum size of 16 byte, which occurs if the segway is to send both v_s, θ_P and ω_p.

Finally, a 16 bit cyclic redundancy check (CRC-16) checksum is added in the end of the frame, to verify that the data sent has been received correctly. To determine if the package do not contain errors, the checksum is compared to the received data. The details of the method behind the CRC-16 code will not be explained here, but the process of generating the checksum can be seen as a polynomial division, where the remainder is the checksum. The CRC16-code generator and CRC-16 checker can be found here [**?**]. If the CRC-16 code does not match up with the data received, the package is discarded, and it is up to the user to either ask for the data again or just ignore the lost data. The datasheet for the APC220 states, the module has a high efficiently error correction implemented [**?**, p. 8]. However the checksum has been chosen to be implemented for better error detection.

An example of a the headers sent during a transmission consisting of a request and reply can be seen in Figure 8.5.

Remote controller requesting angular position and angular velocity.

Field 1	Reply	v_s	θ_P	ω_P	θ_D	ϕ_D	Reserved	Reserved
Value	0	0	1	1	0	0	1	1

Field 2	0

Segway replying with angular position and angular velocity.

Field 1	Reply	v_s	θ_P	ω_P	θ_D	ϕ_D	Reserved	Reserved
Value	1	0	1	1	0	0	1	1

Field 2	8

Figure 8.5: The headers of a data transmission consisting of a request and a reply.

In the example above, the user wants to receive data regarding the segway's angle and angular velocity. Bit 3 and 4 in the first header is therefore set to '1' to indicate so. The second header byte is set to 0 since no data from the user needs to be transmitted to the segway. When the segway receives the request, it responds by transmitting two header bytes where the first header byte indicates it is a reply and the second header byte that the incoming data is 8 bytes long, respectively 4 bytes for the angle and 4 bytes for the angular velocity. After transmitting the header bytes, the 8 data bytes are transmitted, followed by the 2 byte footer, which are now shown. It is possible that a byte gets lost under transmission. To avoid the segway waiting forever on a lost byte, a timeout is implemented which exits the process, if a certain time has expired without any reply has been received. I.e. if the segway waits on the second header byte but the byte is lost, the timeout ensures that the read function returns a 1 meaning the package is lost, whereas a 0 is returned if the package was received successfully. This way, the functions calling the read function can handle the received data or return an error if no data is received.

Examples of two transmissions can be seen in Figure 8.6 and 8.7. In Figure 8.6, it is seen how the remote controller requests for θ_p and is sent over the channel. When the segway receives the request, it replies with transmitting θ_p, which is the request angle to the user.

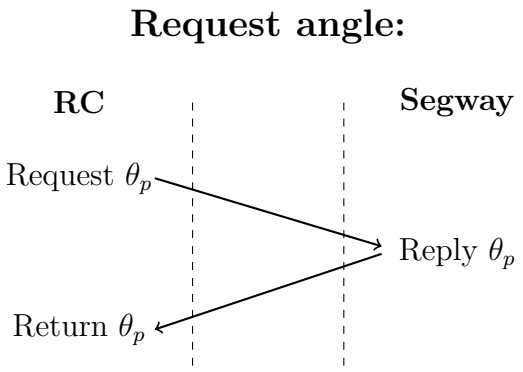


Figure 8.6: The communication for when the RC requests the current segway angle.

In Figure 8.7 the RC requests ω_p, but the reply is lost. This causes the RC to timeout, as it is waiting for a reply within a certain time. Since the reply was not received, the function timeouts, and returns an error message, 0xFFFF, to show that no reply was received. The user can then choose to either ignore this, or try to request the data again.

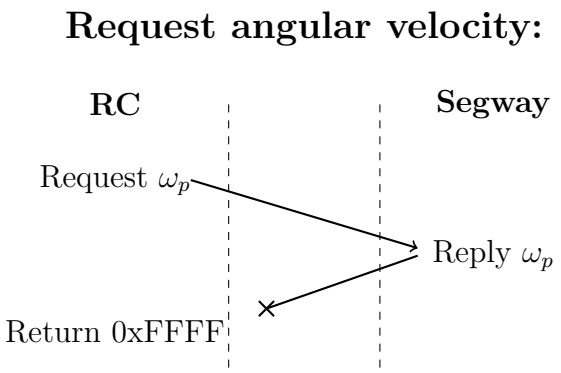


Figure 8.7: The RC requests the segway angular velocity, but the reply is lost.

8.3 Implementation of wireless communication

The implementation of the designed communication protocol and remote controller is explained in this section. The structure of the following section is to first explain the low level implementation and then move to higher level implementation.

The APC220 modules, i.e. the radio modules used for the communication, use UART to serial communication with the segway and RC. The default baud rate for the UART module on the APC220's is 19200 baud. Therefore, the baud rate on the UART modules on the segway and RC are set to 19200 as well. It should be mentioned that the serial communication between the APC220 and the segway is handled by an external UART module on the XMEGA128 chip, and not by the microcontroller itself.

The first step of implementation is to establish communications between the APC220 transceiver module on the RC with the segway. Two basic driver functions for the APC220 are both implemented on the segway and RC. These functions ensure receiving and transmission of data, and are named *APC220_SEND()* and *APC220_READ()*. The implemented code can be seen in Listing 8.1.

```
1  void APC220_SEND(uint8_t APC220_SEND_DATA){
2      uartWriteChar(APC220_SEND_DATA);    // Transmit a byte through UART TX
3  }
4
5  uint8_t APC220_READ(uint8_t *APC220_READ_DATA){
6      int timer = 0;
7      while (uartAvailable() == 0 && timer++ < TIMEOUT); // Timeout counter
8      if(timer < TIMEOUT){
9          *APC220_READ_DATA = uartRead(); // Read data from UART RX
10         return 0;
11     }
12     return 1;
13 }
```

Listing 8.1: APC220 driver for segway.

When the UART RX buffer on the segway has received data from the APC220 module, an interrupt service rutine (ISR) is triggered to set a flag high, stating there is data in the UART RX buffer. The main code will check if this flag is set, and if that is the case, it will handle the information received. A similar APC220 driver is implemented on the RC. With the implemented APC220 UART drivers, it is now possible to transmit and receive bytes through the APC220 modules. These driver functions will be used to transmit and receive the header bytes and data bytes. With the send and read driver functions for the APC220 working, it is now possible to proceed with implementing the communication protocol.

The implemented communication protocol can be seen in Listing 8.2. The structure of the protocol is split into high level and low level functions.

```
//      HIGH LEVEL FUNCTIONS
float requestSpeed();
float requestAngle();
float requestAngularVelocity();
void sendDesiredAngle(float desiredAngle);
void sendDesiredTurn(float desiredTurn);

//      LOW LEVEL FUNCTIONS
int ReceivePackage(void);
void extractDatafromSignal(uint8_t *header, uint8_t *inData);
void handleTransmitRequest();
void sendSpeed(uint8_t dataIndex);
void sendAngle(uint8_t dataIndex);
void sendAngularVelocity(uint8_t dataIndex);
int transmitPackage();
int dataReceivedHandler();
void handleTransmitRequestSegway();
```

Listing 8.2: Functions used for the communication protocol.

The high level functions are used by the user, and can be called from the main code. Through the high level functions it is, for instance, possible to request the current speed of the seqway by calling the *requestAngle()* function. The function then returns the speed of the segway when the RC has received a response from the segway. The low level functions are handled by the high level functions, and should not be called by the user. The contents of the low level functions are primary extracting information from the received packages, and making sure that transmissions comply with the protocol. The high level functions perform call to the lower level drivers, i.e. by transmitting a request and returning the value that was replied from the segway.

The last part in the implementation is how to handle the high level function from the main file for the RC and segway. It is straight forward to handle the high level function from the RC, since the only thing required is to simply call one of the high level functions from Listing 8.2. On the segway, a flag is used to indicate if there is data in the UART RX buffer. If the UART RX buffer is not empty the function *dataReceivedHandler()* is called to read the package. Afterwards, the function *handleTransmitRequestSegway()* is called to perform the desired action from the RC, e.g. to transmit the tilting angle or set a new speed reference. Listing 8.3 shows the implemented code main code for the communication on the segway side.

```
if(RECEIVE_FLAG()){
    if(dataReceivedHandler() == 0){
        handleTransmitRequestSegway();
    }
    SET_RECEIVE_FLAG(0);
}
```

Listing 8.3: Read and handle the data from the UART buffer on the segway.

The communication protocol is now designed and implemented and can now be used for communicating between the segway and RC.

9 | Acceptance test

To ensure the segway complies with the functional requirements, see the requirements in section 3.2, it is necessary to perform an acceptance test of the developed system. The purpose of the chapter is therefore to do validation and verification of the system. To do so, an acceptance test procedure will be developed and used to test the segway. The acceptance test will only concern the functionality requirements since the performance requirements already have been verificated in chapter 6.

9.1 Functionality overview

The segway has been developed through the design chapters.
From the requirement section the design of the segway should ensure that following functionalities are met:

1. The segway must be able to stabilise itself in an upright equilibrium state.
2. The segway must be able to drive forward and backwards without causing the segway to fall over.
3. The segway must be able to turn without causing the segway to fall over.
4. The segway may not fall over when pushed lightly.
5. The user must be able to make the segway drive by a wireless controller.
6. The user must be able to make the segway turn by a wireless controller.
7. The user must be able to receive data from the segway on a wireless controller.

The turning feature has not been implemented in the system, due to time constrains. It is therefore not possible to fulfill requirement 3 and 6.

9.2 Pass/fail criterias

There are three assessment degrees on how the developed system satisfies the requirements. The degrees are ✓ for a pass, (✓) for a partial pass and ✗ for a fail.

1. The segway must be able to stabilise itself in an upright equilibrium state.
   - ✓ The segway stabilises itself in an upright position.
   - (✓) The segway stabilises itself in an upright position, but falls over at rare occasions. A rare occasions is set to be max 1 fail every 10 test.

✗ The segway does not stabilise itself and falls down.

2. The segway must be able to drive forward or backwards without causing the segway to fall over.

   ✓ The segway drives forward or backwards without causing the segway to fall over.

   (✓) The segway drives forward or backwards, but falls over at rare occasions. A rare occasions is set to be max 1 fail every 10 test.

   ✗ The segway falls over when driving forward or backwards.

3. The segway must be able to turn without causing the segway to fall over.

   ✓ The segway turns without causing the segway to fall over.

   (✓) The segway turns but falls over at rare occasions.

   ✗ The segway falls over when turning.

4. The segway may not fall over when pushed lightly.

   ✓ The segway do not fall over when pushed.

   (✓) The segway do not fall over when pushed, but falls over at rare occasions. A rare occasions is set to be max 1 fail every 10 test.

   ✗ The segway falls over when pushed.

5. The user must be able to make the segway drive by a wireless controller.

   ✓ The remote controller is able to control of the segway wirelessly. It is possible drive forward and backwards.

   (✓) The remote controller sends a request to make the segway drive, but the segway does not perform the request action.

   ✗ The remote controller is unable to control of the segway wirelessly.

6. The user must be able to make the segway turn by a wireless controller.

   ✓ The remote controller is able to control of the segway wirelessly. It is possible to turn the segway.

   (✓) The remote controller sends a request to make the segway turn, but the segway does not perform the requested action.

   ✗ The remote controller is unable to control of the segway wirelessly.

7. The user must be able to receive data from the segway on a wireless controller.

   ✓ The remote controller is able to receive different types of data from the segway wirelessly.

   (✓) The remote controller only able to receive one type of data from the segway wirelessly.

   ✗ The remote controller is unable to receive data from the segway wirelessly.

9.3 Test

Setup of the acceptance test and how it is carried out, are described in this section. Each functional requirement is tested individually.

9.3.1 Date and location

The acceptance test was performed the 10. December 2015 at Aalborg University, Fredrik Bajers Vej 7, Aalborg Øst.

9.3.2 Equipment

To perform the tests, following test equipments are needed:

Equipment	**Version**
Segway	AAU 3061
Arduino Uno	Rev. 3
Arduino IDE	1.6.0
APC220 - 2 devices	1
Computer	-

Table 9.1: Test equipment used for the acceptance test.

The Arduino Uno is used as the segway's remote controller and is programmed and monitored through Arduino's IDE. Through the serial monitor it is possible to transmit user inputs and receive data from the segway. In the following section, the test procedures for the acceptance are explained.

9.3.3 Procedure for requirement 1

For the first requirement, the reference angle for the segway is set to 0. When set to 0, the segway will try to stabilise itself and hold its position. When the segway is turned on, the reference angle is set to 0 per default. The starting position of the segway should be in upright position. Observe the segway, and determine if the segway is stable or unstable over a period of 10 minutes.

9.3.4 Procedure for requirement 2

The second requirement is the segway driving forward and backwards. Like the previous test, the default reference angle is set to 0. The reference angle for the segway can be set to a positive angle if it is desired to make the segway drive forward, and a negative value if the segway should drive backwards. Therefore, the remote controller is used to change the reference angle to the desired one, to make the segway drive. For the second test, the following test procedure is used:

1. Put the segway in an upright position and turn the segway on.
2. After the segway is stabilised, change the reference angle to 0.02 by using the RC to transmit a request to the segway.
3. Observe if the segway drives forward.
4. Change the reference angle to -0.02.
5. Observe if the segway drives backwards.
6. Stop the segway by changing the reference angle to 0.

Because the second bullet requires that it is possible to transmit and receive data between the segway and the RC, the 5'th requirement, regarding the RC, should be tested before proceeding to this requirement.

9.3.5 Procedure for requirement 3

The third requirement is the segway turning. Like the previous test, the default reference angle is set to 0. For the second test, the following test procedure is used:

1. Put the segway in an upright position and turn the segway on.
2. After the segway is stabilised, change the turning angle to 90 degrees by using the remote controller.
3. Observe if the segway turns in the counter clockwise direction.
4. Change the reference angle to -90 degrees.
5. Observe if the segway turns in the clockwise direction.

Because the second bullet requires that it is possible to transmit and receive data between the segway and the RC, the 6'th requirement, regarding the RC, should be tested before proceeding to this requirement.

9.3.6 Procedure for requirement 4

In this test, the segway is pushed lightly and should afterwards recover from the push by stabilising itself.

1. Put the segway in an upright position and turn the segway on.
2. After the segway is stabilised push the segway lightly to one direction.
3. Observe if the segway stabilises itself, in its upright position.

9.3.7 Procedure for requirement 5

The fifth test is to verify, if it is possible to control the driving of the segway from an RC.

1. Put the segway in an upright position and turn the segway on.
2. After the segway is stabilised, change the reference angle of the segway to 0.02, by using the RC.
3. Observe if the segway moves forward.
4. Set the reference angle to 0.

9.3.8 Procedure for requirement 6

The sixth test is to verify if it is possible to turn the segway from a RC.

1. Put the segway in an upright position and turn the segway on.
2. After the segway is stabilised, change the turning angle of the segway to 90 degrees by using the RC.
3. Observe if the segway turns.

9.3.9 Procedure for requirement 7

The seventh test is to verify if it is possible to control the segway from a RC. From the RC, it should be possible to change the reference angle, and get informed about the measured angle, angular velocity and speed of the segway.

1. Try to independently request the angle, angular velocity and the speed of the segway.
2. Observe if the segway replies by transmitting the data back.

9.4 Results

The results of the tests are listed in Table 9.2.

Requirement	**Result**
The segway stabilises itself in an upright equilibrium state.	✓
The segway drives forward and backwards without the segway falls over.	(✓)
The segway turns without the segway falls over.	✗
The segway does not fall over when pushed.	✓
The segway is able to drive from being remote controlled wirelessly.	✓
The segway is able to turn from being remote controlled wirelessly.	(✓)
The user is able to receive data from the segway on a wireless controller.	✓

Table 9.2: Acceptance results.

The segway is able to stabilise itself in an upright position and hold the position for 10 minutes. The segway is also capable of driving forward and backwards successfully, but falls over when the segway is set to drive forward or backwards after a period. The segway falls over because it needs to hold its tilting angle and therefore needs to accelerate in the tilting direction in order to hold the angle. At some point the segway cannot accelerate more and falls over. When pushed lightly, the segway stabilises itself successfully without falling over. Also, the segway is able to be remote controlled wirelessly by the user from a computer, and the user can receive data from the segway wirelessly. However, the turning functionality has not been implemented.

Based on this, it is concluded that the segway partially fulfills the requirements set for it.

10 | Conclusion

The aim of this project has been to design a controller which would enable a segway to be balancing in an upright position, as the segway can be seen as an inverted pendulum - a classical control problem that is inherently unstable. In the project, a remote controller has also been made, to be able to drive the segway wirelessly.

To design the cascade controller, a model of the system has been derived. This model includes a model of the motors and wheels, as well as a model of the inverted pendulum. The models have been combined and linearised, after which transfer functions for the system are derived. The system model has been verified through tests. From these it is seen that the fit of the motors and wheels model is 84 %, while the fit of the inverted pendulum model is 70%. From these models, two controllers are designed, with the purpose of controlling the motors and stabilising the inverted pendulum. The controller for the motors and wheels is implemented as a P-controller, while the inverted pendulum controller is a PID-controller. The cascade controller, has through simulation, been proven to stabilise the model of the system, with an overshoot of 23.5 %, whereas the requirement is 10%. The requirements for the settling time and rise time are fulfilled. In the same simulation, a steady-state error of 12.3 % is seen, which does not comply with the requirement of 0% steady-state error.

To obtain data from the segway regarding the velocity of the wheels, encoders are used to measure the speed of the motors. To obtain the angle and angular velocity of the inverted pendulum, a gyroscope and an accelerometer are also used. The data from these three sensors are filtered using digital filters, to remove high-frequency noise. The filters are designed in such a way that the phase shift at low frequencies is small, in order to reduce the delay on the measurements at low frequencies. The filters are designed as low-pass Butterworth filters, which are then transformed to the z-domain using bilinear transformation. The filters work as designed for, but the computation time needed to run the filter code decreases the computation time needed for the wireless communication, and thus the filters are not implemented.

A remote controller is also included in the project, which makes it possible to drive the segway forwards and backwards, by sending commands from a PC. The remote controller also allows the user to receive data from the segway, regarding the tilting angle, the angular velocity and the speed of the segway. To facilitate this, a communication protocol has been designed, in which the data package structure and package header are defined.

In the acceptance test, it is seen that the designed controller after some parameter tuning makes the segway stable i.e. makes the inverted pendulum stand upright. The system is also able to re-stabilise after a light push. The segway cannot turn, but through the remote controller, the segway can be put to drive back and forth. Furthermore data can be sent from the segway to the remote controller.

11 | Discussion

Based on the system designed in this project, there are a wide range of things which can be improved upon to increase the performance of the system.

Regarding the main functionality of the system, balancing in an upright position, different issues can be improved. In the project, a P-controller has been made for the motors and wheels, and a PID-controller is used for the inverted pendulum. This is sufficient to stabilise the system, but to further increase the performance, several cascaded control loops can be utilized. For instance, an inner controller for the angular velocity of the inverted pendulum, ω_p can be implemented, as it is known that cascaded control loops increase performance, assuming that the inner loops' responses are sufficiently fast.

Improving the system model may also be of considerable effect, as it has proved necessary to tune the controller parameters with a factor of nearly 50, to make the designed controller work on the real system. This implies an error in the model or the implementation of the controller. The turning functionality shall also be implemented, and the remote controller can be developed, such that a physical joystick can be used to remote control the segway, in order to make it more intuitive for the user to drive the segway.

Regarding the digital filters, the CPU time available for the filtering shall be increased, in order to make it possible to run the filter code on the segway. A faster sampling frequency can also be used for the angle measurements, since the sampling frequency currently is too low to filter the noise noticeably. A hardware I²C module can also be implemented in the microcontroller, since it requires less computation time to transmit and receive data from the sensors, freeing CPU time for other tasks.

After designing the protocol used for the communication between the remote controller and the segway, it is seen that this can be improved to become more efficient. For instance, the *reply* flag is not needed, as it is implied by the data transmitted to the segway, if a reply is to be sent or not. Removing the checksum can be done as well, as the radio modules already include a checksum. This will also reduce the computation time of running the communication code.
In the remote controller, the timeout is currently implemented as a busy-wait, meaning the remote controller cannot perform other tasks until a reply is received or a timeout is achieved. This can be improved by changing it to an interrupt-based implementation.

It is advantageous to change the type of the radio modules, as it has been observed that the data loss over the channel is rather big. A consequence being that data is requested multiple times before a reply is received.

In the microcontroller, a real-time operating system (RTOS) can also be implemented, instead of having multiple timer interrups, which is currently used. This will make the implementation

of multiple controller loops easier, since it allows for all control loops to run on the same timer, in independent tasks, instead of having a timer interrupt for each controller. It will also ease the job of prioritizing the tasks, as this can be done rather easily in a RTOS. However, additional computation time will be added when using a RTOS, due to context switching.
Finally, the feasibility of the implemented code can be investigated in order to ensure that all deadlines are met, with respect to the different controllers, filters and communication handling.

Implementing these suggestions will make the system perform better and result in an overall better product.

Appendix

A | Segway parameters

The purpose of this test is to measure the physical parameters describing the segway, namely the dimensions and masses of various parts of the segway.

Equipment

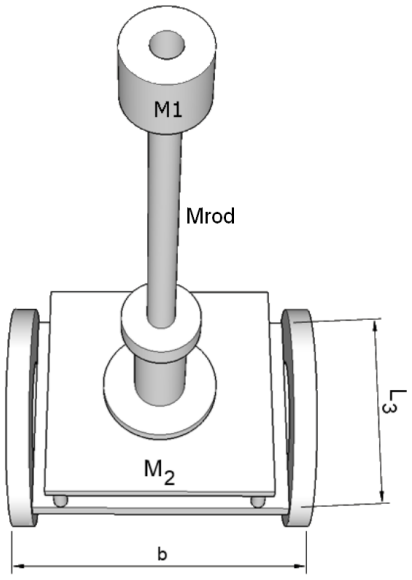
In Table 11.1, the equipment used for performing the measurements is listed.

Name and type	AAU No.	Remarks
KERN - FCB 12K1 - Electronic	86759	1 g precision [?, p. 5]
Ruler	N/A	1 mm division
Caliper	N/A	0.1 mm division

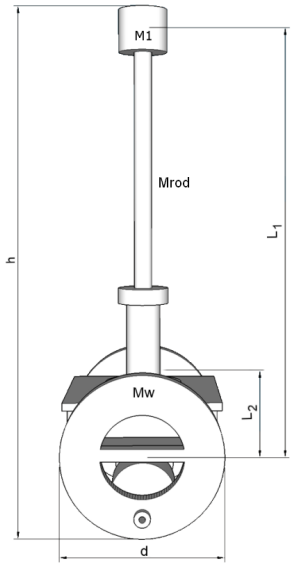
Table 11.1: Equipment used for the test.

Procedure

A blueprint of the segway with the measurement points can be seen in Figure 11.1. The length measurements are performed using the ruler and calliper, where the values are read manually.



(a) The segway blueprint seen from the front.



(b) The segway blueprint seen from the side.

Figure 11.1: A blueprint of the segway with parameter definitions [?, p. 74].

Results

The measurements of the relevant parameters is listed in Table 11.2.

Symbol	Parameter	Value	Unit
d	Wheel diameter	117	mm
h	Total height	371	mm
b	Total platform length	174	mm
L_1	Length from center of rotation to the center of the mass M_p	318	mm
L_2	Distance from the pivot point to the center of mass	57	mm
L_3	Platform width	90	mm
m_p	Mass of cylindrical block at the top of the segway	278	g
m_c	Mass of the platform	1323	g
m_{rod}	Mass of the rod	42	g
m_w	Mass of each wheel	133	g

Table 11.2: Physical dimensions and massses of the segway parts.

Conclusion

The dimensions and masses of relevant parts of the segway have been measured. Any errors will most likely be due to reading errors of the measurements.

B | Motor measurements

The purpose of these measurements is to determine the motor parameters. These parameters are:

- Motor resistance, R_a
- Motor inductance, L_a
- Generator konstant, K_e
- Motor & wheel friction, B
- Motor & wheel moment of inertia, J

These parameters can be seen in the electromechanical equivalent of a DC-motor, see Figure 11.2. Note that the measurements are performed on the motor with the gear and wheel attached, which are seen as a single mechanical system. Thus, all inertias and dampers measured in the following, are the total values of both the motor, gear and wheel. It is known that the gear ration from motor to shaft is $N_{ms} = 19$, while the gear ratio from shaft to wheel is $N_{sw} = \frac{25}{90} \approx 0.2778$

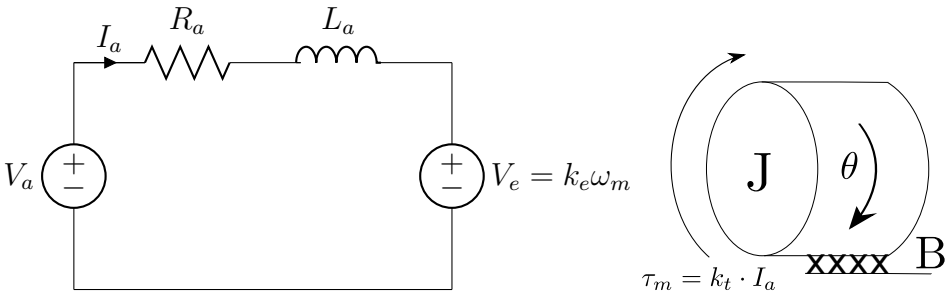


Figure 11.2: The electromechanical equivalent of a DC motor.

Equipment

For these measurements, the following equipments are used:

Name	**AAU No.**	**Type**
HAMEG HM7042-3	AAU60774	Power supply
Fluke 37	AAU33019	Multimeter
Fluke I30S	AAU78550	Amp-meter
Agilent DSO6034A	AAU64572	Oscilloscope
-	AAU08246	Analog Tachometer
Compact A2108	AAU77087	Tachometer

Table 11.3: Equipment used to determine motor parameters.

B.1 Motor resistance, R_a

Setup

The rotor is fixed, so the velocity is 0. In steady state, the motor current and the motor voltage are measured for current from 0 A to 0,9 A. The voltage is measured with a Fluke 37 (AAU33019) multimeter, and the current is measured using the power supply. The current was tried to be measured using a Fluke 37, but the series resistance in the Fluke might have affected the measurements too much, since the results seen were not realistic. Therefore, the multimeter measuring the current was removed. The current was measured with the power supply after verifying, that the measurements were consistent with the Fluke.

Results

Current [A]	**Voltage [V]**	**Resistance [Ω]**
0.00	0.00	-
0.046	0.493	10.7
0.10	0.981	9.81
0.20	1.518	7.59
0.30	2.00	6.93
0.35	2.50	7.15
0.47	3.06	6.52
0.52	3.52	6.77
0.61	4.05	6.64
0.68	4.54	6.68
0.75	5.07	6.76
0.82	5.50	6.75
0.91	6.02	6.65
	Average:	**6.76**

Table 11.4: Results from the resistance measurement.

Table 11.4 and Figure 11.3 show that the resistance goes towards linearity at higher currents. The fit for the trend line is $R^2 = 0.9956$, which means the trend line fits well. From linear regression the resistance is 6.74Ω.

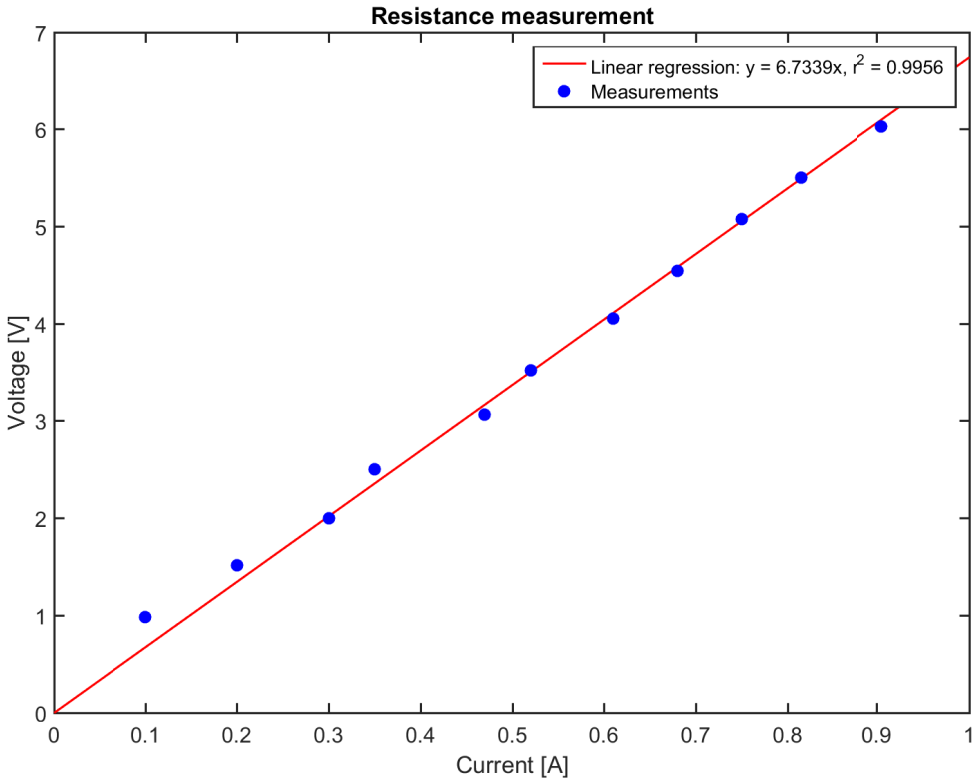


Figure 11.3: Linear regression of voltage-current measurements to determine the resistance.

The reason why the measurements are done in steady state is because in steady state, the model of the motor only consist of a resistor. The inductor and voltage generator are short circuited, making it easy to estimate the resistance.

B.2 Motor inductance, L_a

Setup

The rotor is fixed so the velocity is 0. The current response to a voltage input step is measured. The amp-meter is connected to the oscilloscope, so the current can be measured precisely over time. A voltage step is applied by the PWM drivers with a duty cycle at 200 out of 255.

Results

The current plotted as a function of time can be seen in Figure 11.4.

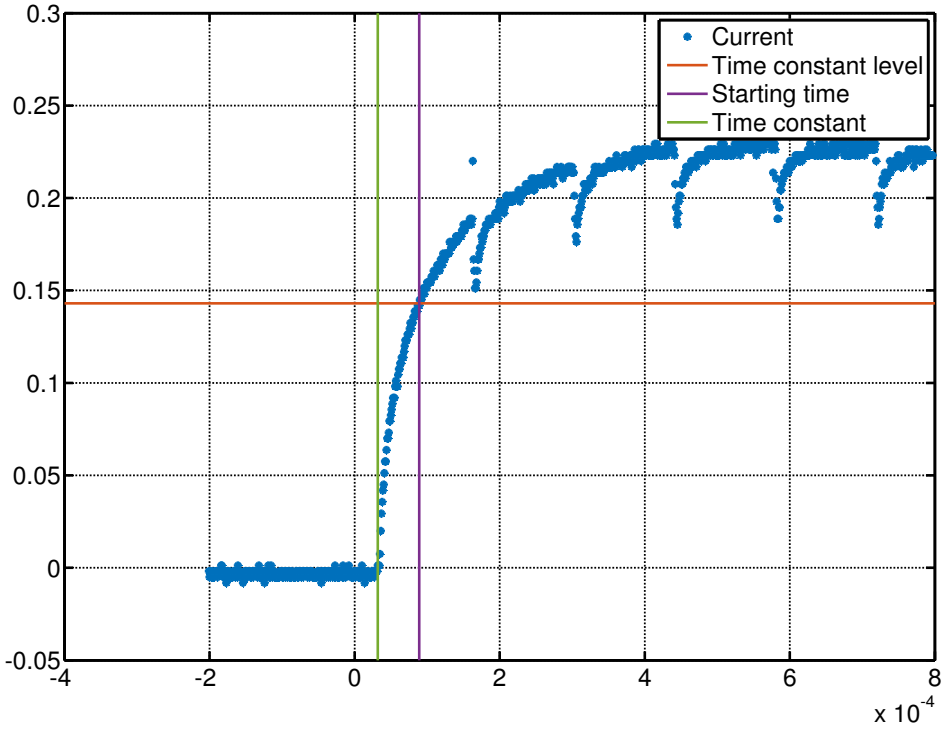


Figure 11.4: Plot of motor current as a function of time, used to find motor inductance.

The transfer function is of 1st order, because there is only one inductor and no capacitors in the motor equivalent circuit. The inductance is found using the time constant of a 1st order circuit. It is known that the time constant can be found as $\tau = \frac{L}{R}$, where $I(\tau) = 0.632 \cdot (I_{max} - I_{min})$. This can be seen in Figure 11.4 as the red line. It is estimated that the voltage step started at the green line. Using this, the time constant is found to be $\tau \approx 57\,\mu s$. Thus, the inductance is found:

$$L = R \cdot \tau = 6.74\,\Omega \cdot 57\,\mu s = 0.384\,\text{mH}$$

The inductance is 0.363 mH according to the datasheet, see [**?**].

B.3 Generator constant, K_e

Setup

In a number of steady state points, the motors voltage and the wheels velocity is measured, using the Fluke multimeter and the onboard encoders. A motor is attached to flexible shaft and to a secondary motor. The motor is fastened using a mounting arm then a current is applied to the secondary motor to make it turn.

Results

The results from the test can be seen in Table 11.5. In the table, the conversion from velocity v to angular velocity, ω_w, are found as:

$$\omega_w = \frac{\mathrm{v}}{r_w} \tag{11.1}$$

$$\omega_m = \frac{1}{N_{ms}} \cdot \frac{1}{N_{sw}} \cdot \omega_w \tag{11.2}$$

Where:

ω_w	is the angular velocity of the wheel	[rad/s]
v	is the translatoric velocity of the wheel	[m/s]
r_w	is the radius of the wheel	[m]
ω_m	is the angular velocity of the motor	[rad/s]
N_{ms}	is the gearing ratio from the motor to the shaft	[1]
N_{sw}	is the gearing ratio from the shaft to the wheel	[1]

Also, K_e can be found using the expression $K_e = \frac{U_a}{\omega}$. Note that the measured velocity has been divided with the gear ratios ($N_{ms} = \frac{1}{19} \approx 0.053$ and $N_{sw} = \frac{25}{90} \approx 0.277$), to obtain the angular velocity of the motor instead of the wheel, as it is the motors angular velocity, ω_m that is used.

$\mathbf{speed_w}$ [m/s]	$\boldsymbol{\omega_w}$ [rad/s]	$\boldsymbol{\omega_m}$ [rad/s]	Voltage [V]	$\boldsymbol{K_e}[\mathrm{V}/\frac{\mathrm{rad}}{\mathrm{s}}]$
0.34	5.81	397.5	4.25	0.0107
0.74	12.65	865.2	9.15	0.0106
0.46	7.86	537.8	5.70	0.0106
0.33	5.64	385.8	4.00	0.0104
0.22	3.76	257.2	2.70	0.0105

Table 11.5: Results from the test to determine the K_e factor.

Averaging the results in Table 11.5, the K_e factor is determined to be:

$$K_e = 10.5 \frac{\text{mV}}{\frac{\text{rad}}{s}}$$

B.4 Motor & wheel friction, B

Setup

In a number a steady state points, the motor current and motor velocity is measured. This is done using the ampmeter and the encoders.

Results

In steady state, i.e. constant angular velocity, the friction torque is $\tau_B = K_t \cdot i_a$. This is true since the resultant torque is 0 when the system is in steady state. Thus, all torque applied by the motor must be countered by an equal torque in opposite direction due to friction.

In Table 11.6 measurements of the angular velocity and the motor current i_a are listed.

ω_m **[rad/s]**	$\mathbf{i_a}$**[A]**
1216.00	0.083
1099.08	0.080
841.85	0.073
502.77	0.066
362.46	0.092
748.31	0.096
1052.31	0.100
1180.92	0.106

Table 11.6: The measurements of the angular velocity and I_a for determining the damper coefficient B.

The motor torque, and thus the damper torque τ_B, can be found using $\tau_B = K_t \cdot i_a$, where $K_t = K_e = 10.5 \frac{\text{mNm}}{A}$

Since it is known that the resultant torque of a damper can be found as $\tau_B = B \cdot \omega$, B can be found since both τ_B and ω are known. The results can be seen in Table 11.7.

$\boldsymbol{\omega_m}[\frac{\boldsymbol{rad}}{\boldsymbol{s}}]$	$\boldsymbol{\tau_B}[\boldsymbol{Nm}]$	**B** $[\frac{\text{Nm}}{\frac{\text{rad}}{s}}]$
1216,00	$875,27 \cdot 10^{-6}$	$0,720 \cdot 10^{-6}$
1099,08	$843,63 \cdot 10^{-6}$	$0,768 \cdot 10^{-6}$
841,85	$769,82 \cdot 10^{-6}$	$0,914 \cdot 10^{-6}$
502,77	$696,00 \cdot 10^{-6}$	$1,384 \cdot 10^{-6}$
362,46	$970,18 \cdot 10^{-6}$	$2,677 \cdot 10^{-6}$
748,31	$1012,36 \cdot 10^{-6}$	$1,353 \cdot 10^{-6}$
1052,31	$1054,54 \cdot 10^{-6}$	$1,002 \cdot 10^{-6}$
1180,92	$1117,81 \cdot 10^{-6}$	$0,947 \cdot 10^{-6}$

Table 11.7: The friction torque and friction coefficient B.

Averaging the results for B in Table 11.7, B is found as:

$$B = 1.22 \frac{\mu\text{Nm}}{\frac{\text{rad}}{s}}$$

B.5 Coulomb friction, τ_c

Setup

In a number a steady state points, the motor current and applied voltage are measured. The voltage is applied by the Hameg power supply, the current is measured with the amp-meter, and the voltage with the multimeter. Then steps are made on the Hameg with approximately 0.1 V until 2.4 V from there the steps are 0.2 V to 5 V.

Results

It is expected that the current can be estimated by two straight lines, because of the coulomb friction. Until $Ia \cdot k_t$ equals the coulomb friction. The current is expected to have a steep slope, and when the motor starts rotating, the slope is expected to be less steep. This can be seen in Figure 11.5, where these slopes and offsets have been estimated using curve fitting.

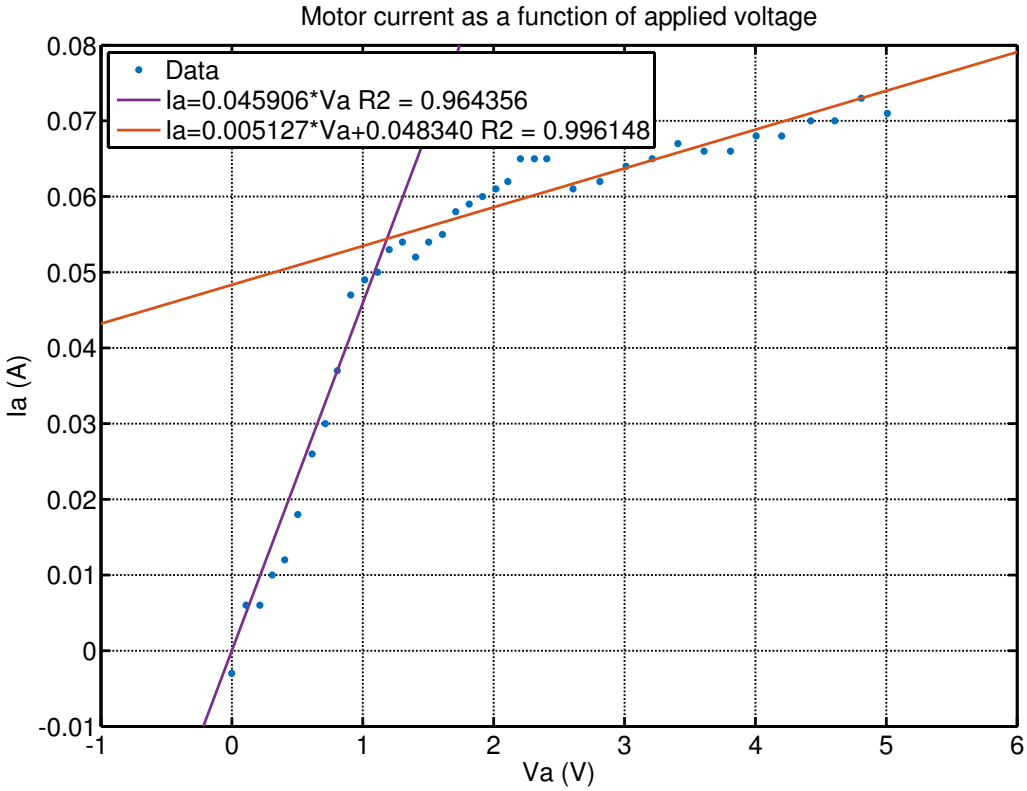


Figure 11.5: Motor current as a function of applied voltage with two curves fitted to the data area before and after wheel started turning.

The coulomb friction can be found based in Figure 11.5 by finding the current in the intersection between the two lines and then multiplying with k_t.

$$0.045906 \cdot Va = 0.005127 \cdot Va + 0.048340 \tag{11.3}$$

$$Va = 1.18541\,\mathrm{V} \tag{11.4}$$

The current in the intersection can be found by inserting Va in any of the equations.

$$Ia = 0.045906 \cdot 1.18541 \tag{11.5}$$

$$Ia = 0.054417\,\mathrm{A} \tag{11.6}$$

The coulomb friction can then be found as:

$$\tau_c = Ia \cdot k_t \tag{11.7}$$

$$\tau_c = 0.054417\,\mathrm{A} \cdot 0.0105 \frac{\mathrm{Nm}}{\mathrm{A}} \tag{11.8}$$

$$\tau_c = 571.4 \mu\mathrm{Nm} \tag{11.9}$$

Note that during another version of this test, the maximum velocity was found to be 1.17 m/s when driven by the segway's PWM signal.

B.6 Moment of inertia, J

Setup

The motor is rotated with a fixed velocity, after which the motor is turned off i.e., making $i_a = 0$ by setting the duty cycle to 0. The motor velocity is measured with a interval of 5 ms. This is done by using the onboard encoders and a timer interrupt.

Results

Looking at a motor's kinematics, the mechanical equation can be expressed as shown in Equation 11.10

$$I \cdot \dot{\omega}(t) = k_t \cdot Ia(t) - B \cdot \omega(t) - \tau_c \tag{11.10}$$

If it is assumed that the motor is running at $\omega(0)$ and the motor then is terminated ($Ia = 0$), the differential equation can then be solved. The result can be seen in Equation 11.11

$$\omega(t) = -\frac{\tau_c}{B} + \left(\omega(0) + \frac{\tau_c}{B}\right) \cdot e^{-\frac{B}{I} \cdot t} \tag{11.11}$$

By knowing this, the output graph seen in Figure 11.6, shows that the rotational speed as a function of time can be interpreted. By inserting the previous found values, a fit has to be found manually, this is the purple graph in Figure 11.6. A better fit can be found if the damper is divided by 8 before fitting the inertia. This can be seen as the orange graph in Figure 11.6.

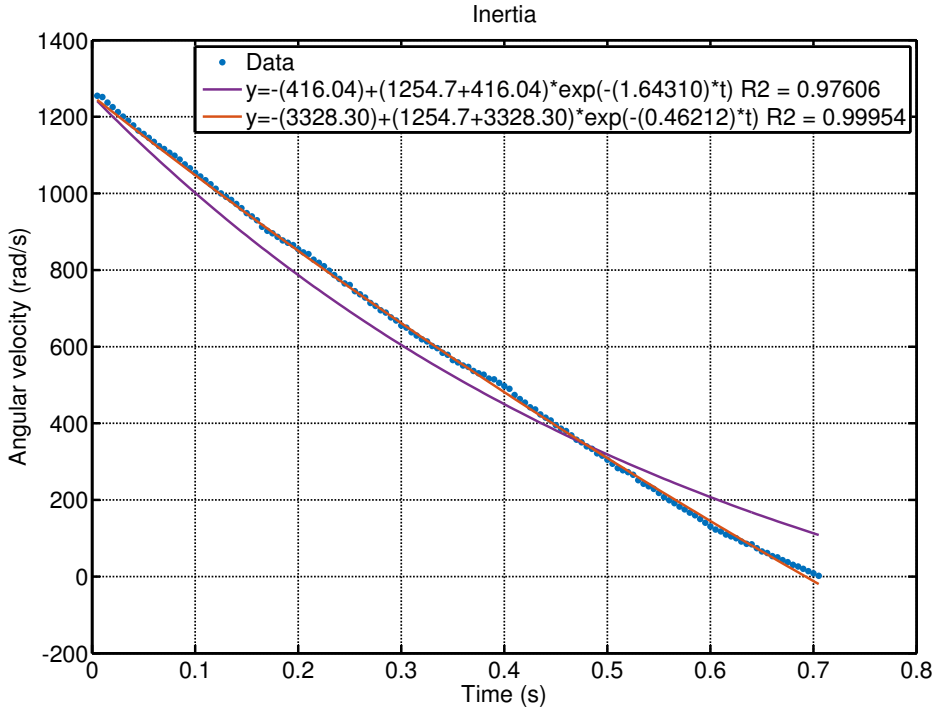


Figure 11.6: Motor velocity as a function of time, when the motor current i_a is set to 0 at time 0.

Because of this measurement, it is decided to divide the damper by a factor of 8 in the model. Based on this, the damper and inertia is found and can be seen in Table 11.8

Damper	$152.5 \cdot 10^{-9} \frac{\text{sNm}}{\text{rad}}$
Inertia	$330 \cdot 10^{-9} \frac{kg}{m^2}$

Table 11.8: The friction coefficient B, and Inertia J, to be used in model.

C | Linearisation with Taylor expansion

In this section the second and third governing equations are linearised, through the use of Taylor expansion, around the operating point $\bar{\theta}_p(t) = 0$. This method of linearisation is described in subsection 5.4.2.

The Taylor expansion is shown in Equation 11.12, where each term has been designated a letter. The reason for this is that these letters will be used for traceability during the linearisation of Equation 11.18 and Equation 11.13.

$$T = \underbrace{f(\bar{\theta}_p(t), \dot{\bar{\theta}}_p(t), \ddot{\bar{\theta}}_p(t))}_{\text{A}} + \underbrace{\frac{\partial f(\bar{\theta}_p(t))}{\partial \theta_p(t)} \cdot \hat{\theta}_p(t)}_{\text{B}} + \underbrace{\frac{\partial f(\dot{\bar{\theta}}_p(t))}{\partial \dot{\theta}(t)_p} \cdot \hat{\dot{\theta}}_p(t)}_{\text{C}} + \underbrace{\frac{\partial f(\ddot{\bar{\theta}}_p(t))}{\partial \ddot{\theta}(t)_p} \cdot \hat{\ddot{\theta}}_p(t)}_{\text{D}} \tag{11.12}$$

Second governing equation

The second governing equation, where the terms that are to be linearised are marked:

$$m_c \cdot r_w \cdot \ddot{\theta}_w(t) = F_F(V_a(t)) - m_p\Big(\ddot{\theta}_w(t) \cdot r_w + \underbrace{l \cdot \sin(\theta_p(t)) \cdot \dot{\theta}_p^2(t)}_{\text{T1}} - \underbrace{l \cdot \cos(\theta_p(t)) \cdot \ddot{\theta}_p(t)}_{\text{T2}}\Big) \tag{11.13}$$

Applying the Taylor expansion to each of the non-linear terms yields:

Term 1 linearised:

$$\underbrace{l \cdot \sin(\bar{\theta}_p(t)) \cdot \dot{\bar{\theta}}_p^2(t)}_{\text{A}} + \underbrace{l \cdot \cos(\bar{\theta}_p(t)) \cdot \dot{\bar{\theta}}_p^2(t) \cdot \hat{\theta}_p(t)}_{\text{B}} + \underbrace{2 \cdot l \cdot \sin(\bar{\theta}_p(t)) \cdot \dot{\bar{\theta}}_p^2(t) \cdot \hat{\dot{\theta}}_p(t)}_{\text{C}} = 0 \tag{11.14}$$

Term 2 linearised:

$$\underbrace{l \cdot \cos(\bar{\theta}_p(t)) \cdot \ddot{\bar{\theta}}_p(t)}_{\text{A}} - \underbrace{l \cdot \sin(\bar{\theta}_p(t)) \cdot \ddot{\bar{\theta}}_p(t) \cdot \hat{\theta}_p(t)}_{\text{B}} + \underbrace{l \cdot \cos(\bar{\theta}_p(t)) \cdot \hat{\ddot{\theta}}_p(t)}_{\text{D}} = l \cdot \hat{\ddot{\theta}}_p(t) \tag{11.15}$$

Linearised second governing equation:

$$m_c \cdot r_w \cdot \ddot{\theta}_w(t) = F_F(V_a(t)) - m_p \cdot \ddot{\theta}_w(t) \cdot r_w + \underbrace{l \cdot \hat{\ddot{\theta}}_p(t)}_{\text{T2}} \tag{11.16}$$

Which is rearranged into:

$$(m_p + m_c) r_w \cdot \ddot{\theta}_p(t) = F_F(V_a(t)) + \underbrace{l \cdot \hat{\ddot{\theta}}_p(t)}_{\text{T2}} \tag{11.17}$$

Third governing equation

The third governing equation, where the terms that are to be linearised are marked:

$$\underbrace{(J_p + m_p \cdot l^2) \cdot \ddot{\theta}_p(t)}_{\text{T1}} = m_p \cdot l \cdot \Big(\underbrace{\sin(\theta_p(t)) \cdot g}_{\text{T2}} + \underbrace{\cos(\theta_p(t)) \cdot r_w \cdot \ddot{\theta}_w(t)}_{\text{T3}} \Big) \tag{11.18}$$

Applying the Taylor expansion to each of the non-linear terms yields:

Term 1 linearised:

$$\underbrace{(J_p + m_p \cdot l^2)\bar{\ddot{\theta}}_p(t)}_{\text{A}} + \underbrace{(J_p + m_p \cdot l^2)\hat{\ddot{\theta}}_p(t)}_{\text{D}} = (J_p + m_p \cdot l^2)\hat{\ddot{\theta}}_p(t)$$

Term 2 linearised:

$$\underbrace{\sin(\bar{\theta}_p(t)) \cdot g}_{\text{A}} + \underbrace{\cos(\bar{\theta}_p(t)) \cdot g \cdot \hat{\theta}_p(t)}_{\text{B}} = g \cdot \hat{\theta}_p(t)$$

Term 3 linearised:

$$\underbrace{\cos(\bar{\theta}_p(t)) \cdot r_w \ddot{\theta}_w(t)}_{\text{A}} - \underbrace{\sin(\bar{\theta}_p(t)) \cdot r_w \cdot \ddot{\theta}_w(t)) \cdot \hat{\theta}_p(t)}_{\text{B}} = r_w \cdot \ddot{\theta}_w(t)$$

Linearised third governing equation:

$$\underbrace{(J_p + m_p \cdot l^2) \cdot \hat{\ddot{\theta}}_p(t)}_{\text{T1}} = m_p \cdot l \Big(\underbrace{g \cdot \hat{\theta}_p(t)}_{\text{T2}} + \underbrace{r_w \cdot \ddot{\theta}_w(t)}_{\text{T3}} \Big) \tag{11.19}$$