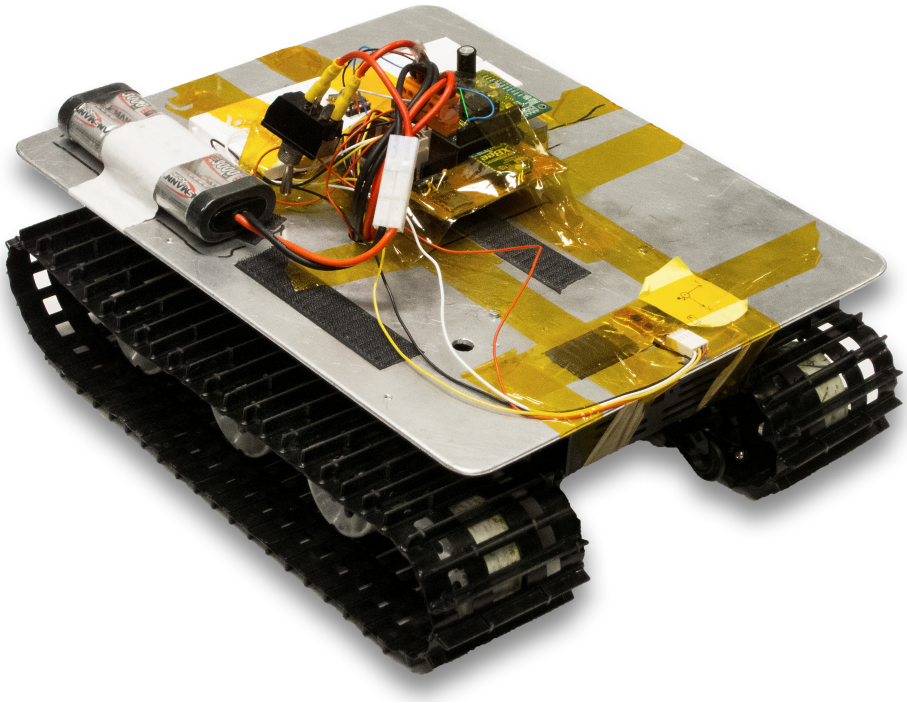
Autonomous Lawn Mower

Utilizing a Local Positioning System



5. Semester Project Report

Group 15gr510

Aalborg University
Electronic Engineering & IT
Fredrik Bajers Vej 7
DK-9220 Aalborg

This report is compiled in LaTeX, originally developed by Leslie Lamport, based on Donald Knuth's TeX. The main text is written in *Latin Modern* pt 12, designed by Bogusław Jackowski and Janusz M. Nowacki. Flowcharts and diagrams are made using Microsoft Visio.

AALBORG UNIVERSITY
STUDENT REPORT

5th Semester
School of Information and Communication Technologies
Electronics and IT
Fredrik Bajers Vej 7B
9220 Aalborg
http://www.sict.aau.dk/electronics-and-it

Title:
Autonomous Lawn Mower

Theme:
Digital and Analog Systems
Interacting with the Surroundings

Project Period:
P5, Autumn 2015
02/09/2015 - 17/12/2015

Project Group:
510

Participants:
Amalie V. Petersen
Julien Brehin
Mads R. Gotthardsen
Niels Skov Vestergaard
Romaric Destremau
Thomas Rasmussen

Supervisor:
Tom Søndergaard Pedersen
Rasmus Pedersen

Prints: 9
Pages: 138
Appendices: 16
Attached: 1 CD
Concluded: 17/12/2015

Synopsis

In order to facilitate the lawn mowing, robotic lawn mowers have been pushed to the market the last few years.
From a given vehicle base, this project intends to improve the easiness of use and reduce installation constraints by use of ultrasound positioning and wireless communication.
Models of this vehicle have been built, upstream from the design and implementation of appropriate controllers to actually make the vehicle autonomous. The prototype is implemented with a positioning system which communicates with an Arduino platform running an open-source Real Time Operating System. Feedback is fed to the controllers from angular and velocity sensors for which filtering has been considered.
Except from one functionality which could not be tested in practice due to one sensor incompatibility with indoor environment. The different parts of the system have been simulated through software and have been proven to work successfully in real life.

Preface

This project aims at the design of an autonomous vehicle prototype, able to run and follow a predefined route, in the perspective of using it as a lawn mower.

The hereby report has been written by a group of students on fifth semester in "Electronics and IT" at Aalborg University.
The reader should have a basic knowledge and understanding in electronics engineering, and more specifically in modelling and control, communication in electronic systems and signal processing. Specific knowledge related to the implemented controllers, communication protocol and digital filter will be described in more details. Code for the implementation of the prototype is written in C, Arduino-specific C++ and in C#. It is assumed that the reader is able to comprehend these programming languages.

Special thanks are addressed to Rasmus Gundorff Sæderup for his contribution in taking professional looking pictures of the vehicle, Associate Professor Henrik Schiøler for guidance on the GoT system, and to Associate Professor Jens Dalsgaard Nielsen for the support he brought the group on his Real Time Operating System. Final thanks are for the group's supervisors Tom Søndergaard Pedersen and Rasmus Pedersen and their guidance throughout the semester.

Reading Instructions

- This report is organized in three parts. The first part describes the design process of the wanted prototype. The second one covers the actual implementation choices made to fulfill the requirements. In the last part, the prototype's characteristics are actually tested. From this, a conclusion on the project is made as well as a discussion on potential improvements of the prototype.
- On the attached CD, the following content can be found: main project code, code, data and Matlab scripts used in the tests, datasheets on components and standards used, schematics, sources used in the project which are not freely available.

Text by:

Amalie V. Petersen

Julien Bréhin

Mads R. Gotthardsen

Niels Skov Vestergaard

Romaric Destremau

Thomas Rasmussen

Contents

Part I Preanalysis **1**

1 Introduction **2**
1.1 Robotic Lawn Mowers . . . 2
1.2 The Games on Track System . . . 4
1.3 Satellite Based Positioning Systems vs GoT . . . 5

2 Design Considerations **6**
2.1 Use-case Design . . . 6
2.2 System Base Description . . . 9
2.3 Prototype Constraints . . . 18
2.4 Prototype, Interfaces and Submodules . . . 20

3 Prototype Requirements **24**

Part II Design & Implementation **25**

4 Hardware and Software Considerations **26**
4.1 Hardware Choice . . . 26
4.2 Hardware Implementation . . . 35
4.3 H-Bridge . . . 40
4.4 Software Choice . . . 43

5 Sensor Implementation **47**
5.1 Hall Sensors . . . 47
5.2 Magnetometer . . . 49

6 Communication **50**
6.1 OSI model . . . 50
6.2 Protocol . . . 51
6.3 Communication Filtering . . . 57

7 Modeling of the Vehicle **59**
7.1 Velocity Model of the Vehicle . . . 60
7.2 Steering Model of the Vehicle . . . 77

8 Control of the Vehicle **84**
8.1 Velocity Controller . . . 84
8.2 Steering Controller . . . 96

9 Digital Filter **108**
9.1 Filter Considerations . . . 108

9.2 Filter Requirements . . . 113
9.3 Design . . . 114
9.4 Implementation . . . 121
9.5 Results . . . 123

Part III Test & Conclusion **127**

10 Acceptance Test **128**
10.1 Test Procedure . . . 128
10.2 Test Results . . . 130

11 Conclusion **136**

12 Discussion **137**

Appendix **140**
A Motor Tests - Armature Resistance . . . 140
B Motor Tests - Armature Inductance . . . 143
C Motor Tests - Tachometer Constant . . . 145
D Motor Tests - Generator Constant . . . 146
E Motor Tests - Motor Constant . . . 148
F Motor Tests - Friction . . . 150
G Motor Tests - Moment of Inertia . . . 152
H Motor Tests - Time Constant and Gain . . . 154
I Sensor Test - Magnetometer Calibration . . . 157
J Vehicle Tests - Friction . . . 162
K Vehicle Tests - Inertia and Time Constant . . . 165
L Vehicle Tests - Velocity Gain . . . 169
M Vehicle Test - Proportional Controller Test . . . 172
N Vehicle Tests - Steering - Steering Gain . . . 175
O Vehicle Tests - Steering - Linear Area for K_p . . . 180
P Schematics . . . 183

Part I

Preanalysis

1 | Introduction

More and more robots appear in everyday life. Automatic vacuum cleaners and floor washers are getting widespread, as technologies become cheaper and better. Those new kind of automatic robots have matured to a level, where they can save an important amount of man-hours.

Outside the walls of our homes lays the next weekly hurdle: mowing the lawn. A known way to handle this, is to pay the neighbour's teenager to do it. Unfortunately they grow up and move out, leaving the lawns in the residential neighbourhoods behind. Luckily engineers have stepped in, and provide a more long-term solution: robotic lawn mowers.

1.1 Robotic Lawn Mowers

Several manufacturers of electrical gardening machines have started selling robotic lawn mowers in the recent years. In general they use one of these two strategies when cutting the lawn [1]:

- Random direction mowers.
- Parallel line mowers.

Mowers utilizing the random direction strategy will drive in a straight line until a guard wire or an obstacle is detected. They will then turn in a random direction, and continue. See *Figure 1.1*.

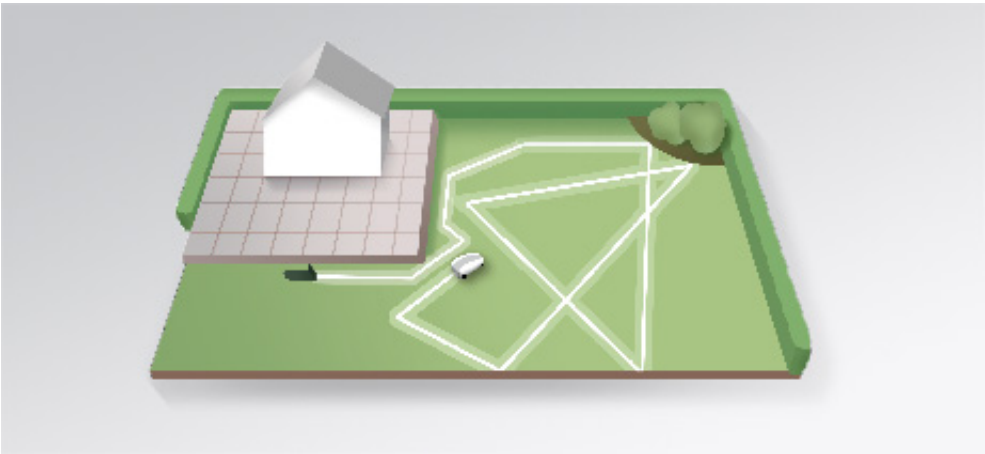


Figure 1.1: Random cut system [1]

When the battery is nearly discharged, the mower will follow the guard wire back to the base station to recharge. Parallel line mowers utilize another control algorithm for mowing. After an initial learning run, following the guard wire around the lawn to be mowed, it will map the lawn, and cut in parallel lines, see *Figure 1.2*. The advantage of this strategy, is that the lawn mower will not run over the same spots more than once. According to Bosch, a given lawn can be mowed up to 30% faster with their Logicut system [1].

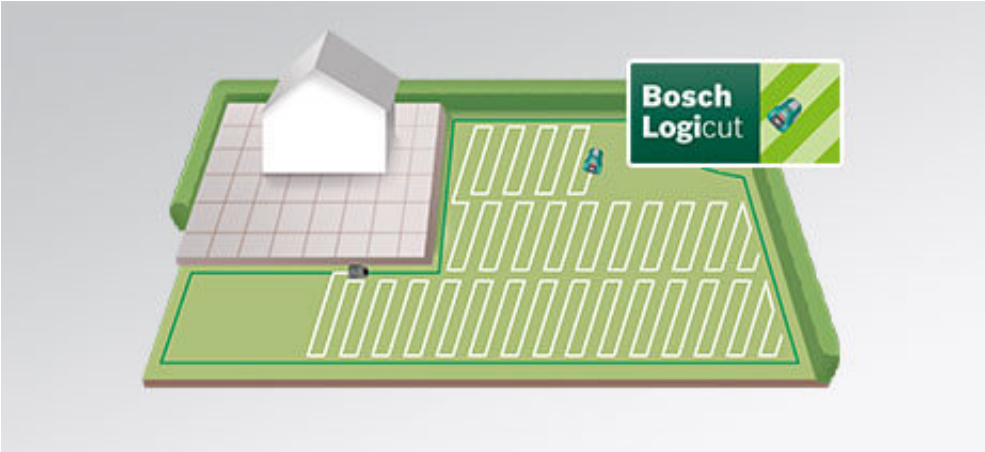


Figure 1.2: Bosch Logicut system [1]

Common for both systems is the guard wire, which has to be placed around the lawn and anywhere the lawn mower is not allowed to go, like flower beds, swimming pools, etc.

Existing Problems

Some commercially available robotic lawn mowers require a guard wire placed around the lawn. It can either be installed at the surface, and be held in place by pegs, or dug down below the surface[2]. The guard wire must be routed around flower beds, bushes etc. as well, see *Figure 1.3*.

The use of the guard wire for guiding the mower back to the charging station presents another potential problem: in a garden with many restricted areas, the guard wire could get very long. Therefore the journey home could be longer, compared to a more direct route. This again uses more battery power, that could have been used for actually mowing the lawn instead.

This will be the motivation for the project: to avoid the work routing a wire around the garden, and as a bonus get more work done on a battery charge, by not wasting power following the wire home.

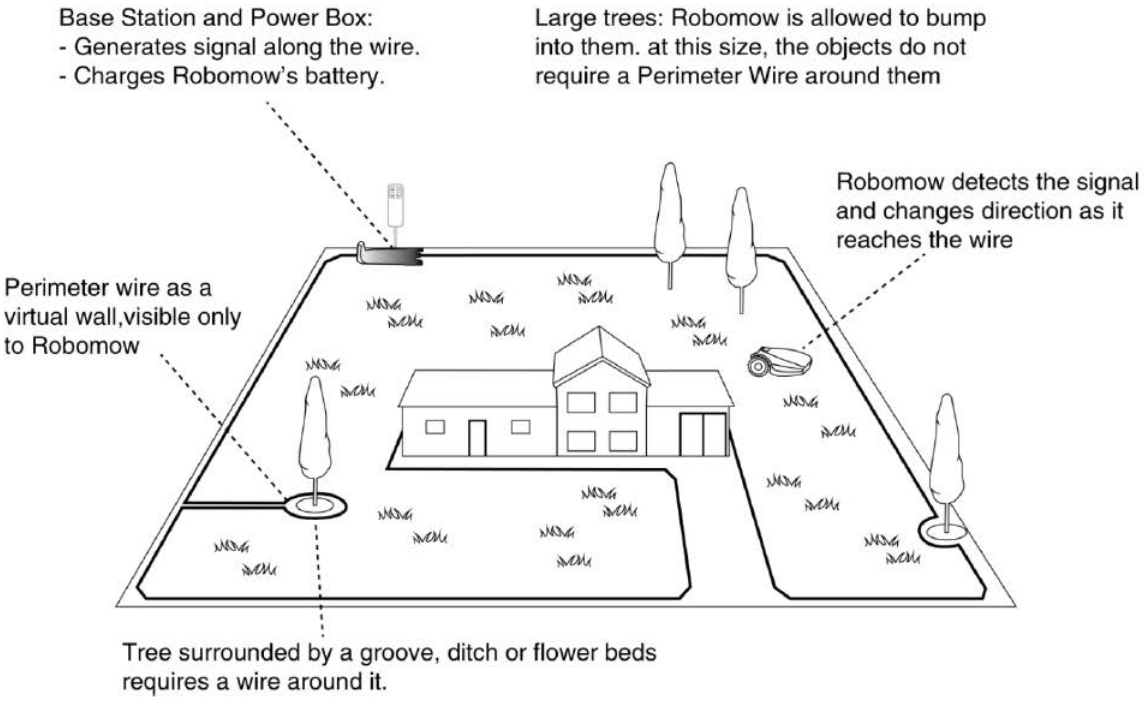


Figure 1.3: Guard wire installation [2]

Then, the next question is: what other solutions could be used to get the lawn mower to go where it has to go? A solution of keeping track of the lawn mower's position in real-time is examined.

1.2 The Games on Track System

One available system is the *Games on Track GT-Position* system, referred to as GoT. The GoT would be able to determine the lawn mower's position in space, see *Section 2.2*. It is composed of three different parts both hardware and software [3]:

- A tracked module, which emits ultra-sound and radio waves. It should be placed on the lawn mower itself.
- Towers which are tracking the tracked module placed on the vehicle are needed. These should be located around the area where the lawn mower will move. There should be at least 3 towers, depending on the terrain, and can be up to more than 20. The more towers, the more accuracy can be obtained to cancel out any ambient noise, and more space can be monitored.
- The master, connected to a computer, receives data from the towers and transmits it to computer through a USB in regular intervals. The distance of the tracked module is then calculated by the computer.

Figure 1.4: Games on Track GT-Position package [source:Games on Track]

The GoT software aggregates the received positions through time, and can be used to draw a map of the lawn, and to determine the absolute position of the tracked module.

GoT was originally designed for train modelling, but it is easily adaptable for any use of position tracking. In the following segment satellite based positioning systems are compared to the discussed GoT system.

1.3 Satellite Based Positioning Systems vs GoT

The reasons why satellites positioning system probably should not be used for autonomous systems like a lawn mower are mainly related to accuracy-over-price ratios and to energy consumption.

Although, there are some cheap standard GPS chips, around 10 USD, these only reach around 1 meter of precision in the most ideal situations [4, 5].

On the other hand, the best GPS chips can achieve precisions up to a few millimeters [4], when combined with different augmentation systems, algorithms for instance, but they end up being highly expensive.

Moreover, when summed up with the bit rates satellites can achieve, the low signal amplifiers on the receiver, plus all the position calculations and potential augmentation systems, the total energy consumption quickly rises.

The autonomy of the vehicle (both in energy and for the navigation), and the overall cost should be considered. The GoT system itself has a cost (around $ 600 per unit for the most basic package) beyond anything a normal customer would probably pay for a lawn mower. But despite that, it appears at first, to be a good solution for a basic autonomous lawn mower. This is in terms of accuracy and energy consumption, compared to systems like GPS which are even more expensive for the same level of accuracy.

These preliminary considerations will influence the design process of an autonomous lawn mower.

2 | Design Considerations

In this chapter the system is designed with a top-down approach, which means an overview of the system is formulated first. Thereafter the system will be broken down into smaller segments.

First a use-case model of the system is given, where the functionalities and the coherent actors are described, in order to give an overview of what the system shall be able to do. Thereafter, a description of the provided vehicle and tracking system is made, to get a knowledge base of the utilized systems. Constraints to the prototype, caused by time limitations as well as a desired focus on the main scope of the project, are considered. After constraints have been considered, the final prototype is described.

2.1 Use-case Design

To give an overview of what the system should be able to do, a use case with coherent actors is utilized. The use case diagram, see *Figure 2.1*, will describe the main functionalities in the use case as well as describing the actors, the external sensors and subsystems, affecting the system.

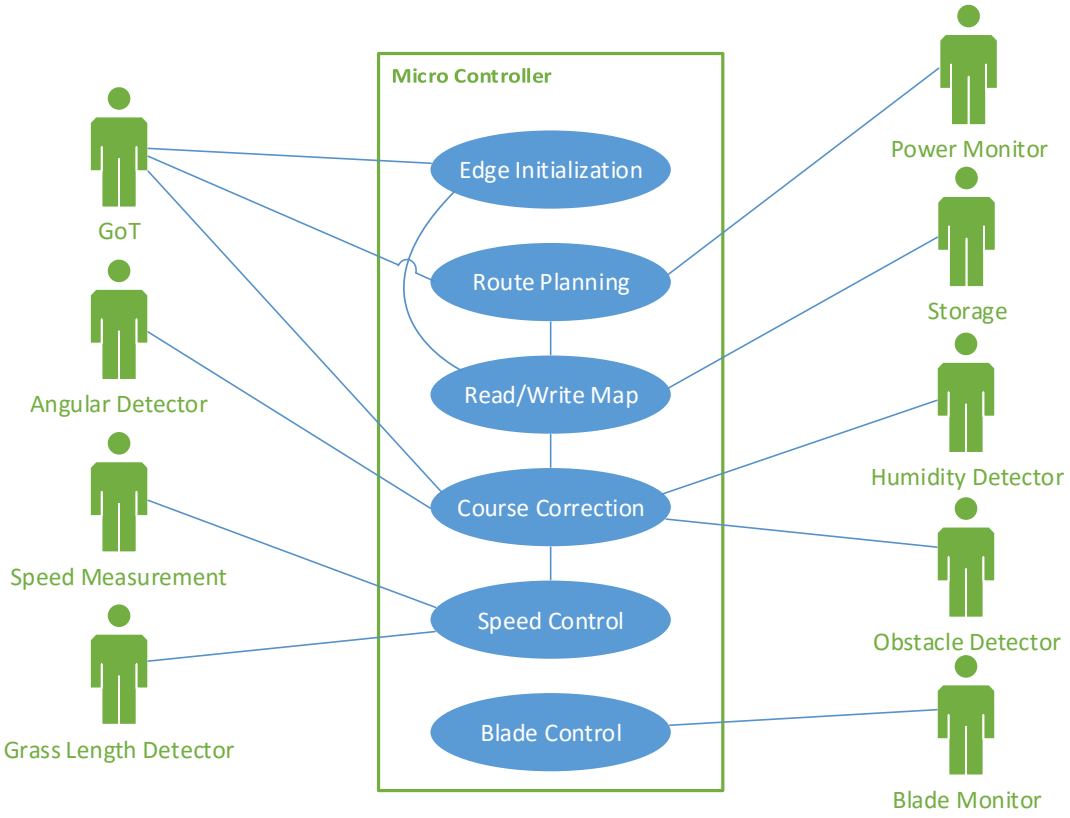


Figure 2.1: Use-Case Diagram

The system, inside the controller, have 6 main functionalities and is affected by 9 different actors.

Main Functionalities

Functionalities are the actions that the system can perform. Each functionality can have more than one function in the system. The functionalities get input from actors and can be in association with other functionalities.

Edge Initialization: The *Edge Initialization* gets input from the *GoT* system and is associated with the *Read/Write Map* functionality. The purpose of this function, is to calibrate the system to a new lawn, by mapping the edge of the lawn. The edges which is mapped should include the area in which the vehicle should stay inside and the areas which it should avoid, e.g. trees, bushes and so on. This is done, by first manually moving the *GoT* system's transmitter along the periphery of the lawn and thereafter along the edges of obstacles placed on the lawn. This could be performed by the use of a button which could make the system record edges when pressed. The information about the edges, an edge map is created and saved through the *Read/Write Map* functionality.

Route Planning: The *Route Planning* gets inputs from the *Power Monitor* and is associated with the *Read/Write Map* functionality. The purpose of this function, is to plan the route for the lawn mower. The route is created by knowing where the lawn mower is located and the edges of the lawn, which are located in *Storage*. The map is loaded from the *Storage* with the *Read/Write Map* functionality. Furthermore it acquires the voltage level of the battery from the *Power Monitor*, to ensure it will return to base before it runs out of battery.

Read/Write Map: The *Read/Write Map* gets inputs from the *Storage* and is associated with the *Route Planning*, *Course Correction* and the *Edge Initialization* functionalities. This functionality is the link between the functionalities and *Storage*. The edge map retrieved from *Edge Initialization* and through the use of the *Read/Write Map* functionality, is stored in the *Storage*. The route which is calculated by the *Route Planning* is also located in *Storage*. Furthermore, the *Course Correction* functionality also needs access to the *Storage* to be able get the calculated route.

Course Correction: The *Course Correction* gets input from the *GoT* system, the *Humidity Detector*, the *Obstacle Detector* and the *Angular Detector* and is associated with the *Read/Write Map* and the *Speed Control* functionalities. This functionality gets both an angle and the humidity in the air as input. Furthermore it gets an input if it encounters an obstacle blocking its path. *Course Correction* takes all these parameters into consideration when navigating from its last registered position, retrieved from the *GoT* system, to the next destination on the calculated route, retrieved from the *Storage* trough *Read/Write Map*. The *Course Correction* is the main function behind the regulation of the vehicle's angular movement, and therefore it can also regulate the vehicle back to the original path if it slips or is moved. The *Course Correction* decides the velocity, according to the calculated route, and passes the decided velocity to the *Speed Control*.

Speed Control: The *Speed Control* gets inputs from the *Grass Length Detector* and the *Speed*

Measurement and is associated with *Course Correction*. The functionality's purpose, is to make sure that the lawn mower adapts the speed depending on the grass length and if the *Course Correction* ask for a different velocity, e.g. if the vehicle is going into a curve. Additionally, if the *Course Correction* ask for a desired velocity from the *Speed Control*, it should be able to go to the desired velocity and maintain it regardless of incoming disturbances.

Blade Control: The *Blade Control* gets inputs from the *Blade Monitor*. The functionality's purpose, is keeping the blade rotational speed the same, to make an even cut grass. With the input from the *Blade Monitor*, the *Blade Control* can regulate the velocity of the blade.

Actors

An actor is a component, that influences the system by giving inputs to the functionalities and/or receiving outputs from them.

GoT: The *GoT* system is connected to the *Course Correction*, the *Route Planning* and the *Edge Initialization* functionalities. This system is used to retrieve the GoT transmitter's location on the lawn. The GoT system is explained in *Section 2.2*.

Storage: The *Storage* is connected to the *Read/Write Map* functionality. It uses a non-volatile storage to store the information about the edge map and route, which is made in the *Edge Initialization* and *Route Planning* functionalities. This will make the *Storage* able to keep the data even if the system is turned off.

Humidity Detector: The *Humidity Detector* is connected to the *Course Correction* functionality. This is a sensor that measures the humidity in the air. The system uses the information about the ambient humidity level to see, whether the grass is too wet for mowing the lawn, since wet grass can not be cut evenly.

Obstacle Detector: The *Obstacle Detector* is connected to the *Course Correction* functionality. This sensor detects, if there are any objects, which blocks the lawn mower's route. This sensor makes sure, that the lawn mower registers objects, like a human or a dog, which could interfere with the lawn mower's route.

Angular Detector: The *Angular Detector* is connected to the *Course Correction* functionality. This sensor measures the angular position and movement of the lawn mower, whether it is intentional movement or if the lawn mower slips.

Grass Length Detector: The *Grass Length Detector* is connected to the *Speed Control* functionality. This sensor measures the grass length, which affects the velocity, at which the lawn mower should drive. If the grass is long, the lawn mower has to drive slower, to make sure that it cuts the grass evenly.

Power Monitor: The *Power Monitor* is connected to the *Route Planning* functionality. This sensor measures how much power is left in the battery. This is needed, so that the lawn mower can drive back to its charging station, before the battery is too low to make the vehicle run, and

thereby not damaging the battery.

Blade Monitor: The *Blade Monitor* is connected to the *Blade Control* functionality. This sensor measures the rotational speed of the blade and sends the information back to the *Blade Control*

Speed Measurement: The *Speed Measurement* is connected to the *Speed Control* functionality. The sensor measures the speed of the lawn mower.

This is a system which could be implemented to be utilized as an autonomous lawn mower. To get a knowledge base on the provided vehicle and battery as well as on the GoT system, a description of these will be given in the following section.

2.2 System Base Description

The technology which has been provided for the prototype is a tracked vehicle, seen on *Figure 2.2*, a battery and a GoT system. A platform with the length and width of the vehicle is mounted on the vehicle to protect the mechanical system. This is utilized to support the rest of the system, which includes PCB boards, the battery, and the sensors used to control the vehicle. When the platform is mounted on, the vehicle is 45 cm long, 29 cm in width, 12 cm in height and weighs 2932 grams.

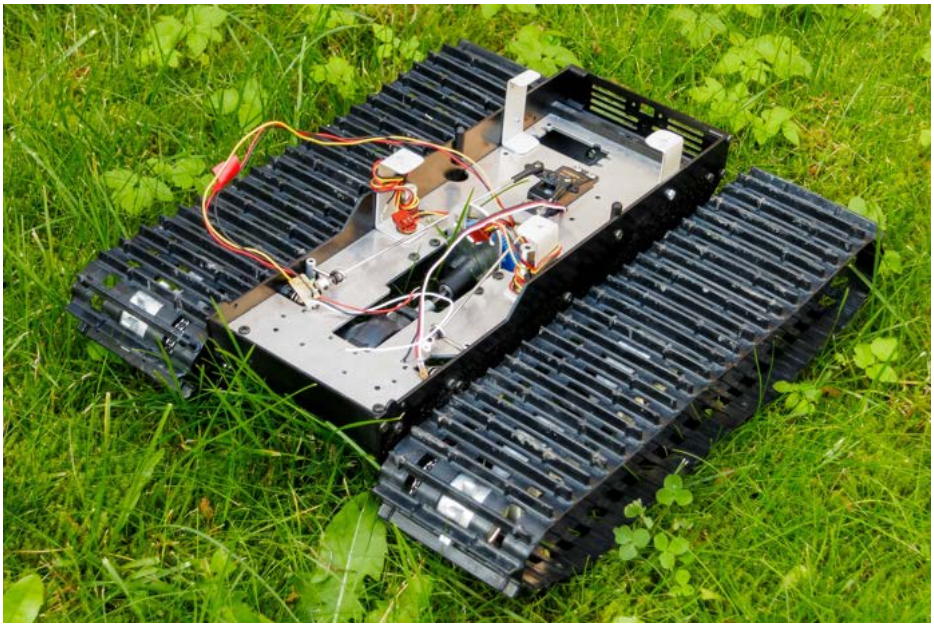


Figure 2.2: The provided tracked vehicle without a platform

To get an understanding of how the motor is used to make the belts rotate, one must comprehend how the drivetrain, see *Figure 2.3*, functions. To make the vehicle turn it is necessary to understand how the servo acts on the brakes, which are connected to the driving wheels on each side of the vehicle. Two Hall sensors are attached to the provided vehicle, along with 4 magnets attached to each drive gear. To run the vehicle a rechargeable battery pack is also provided.

Furthermore it is necessary to understand how the GoT system works to be able to utilize it and thereby be able to track the vehicle.

First a description of the vehicle's drivetrain will be given.

Drivetrain

The drivetrain of a motorized vehicle, is the components that transfer the rotational energy from the motor to the drive wheel of the vehicle. To understand the placement of the different components, a picture of the drivetrain is seen in *Figure 2.3*.

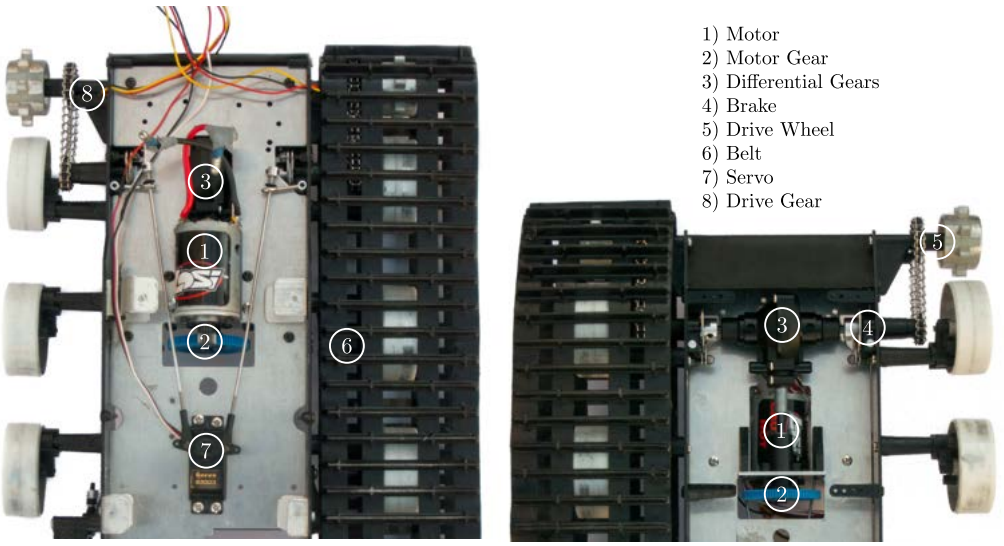


Figure 2.3: Picure presenting some of the elements on the vehicle.

To be able to model this drivetrain, a simplified drawing of the vehicle is made. The drivetrain contains the gear connected to the DC motor, the differential gear box and the gears connected to the belts. The drivetrain is illustrated in *Figure 2.4*.

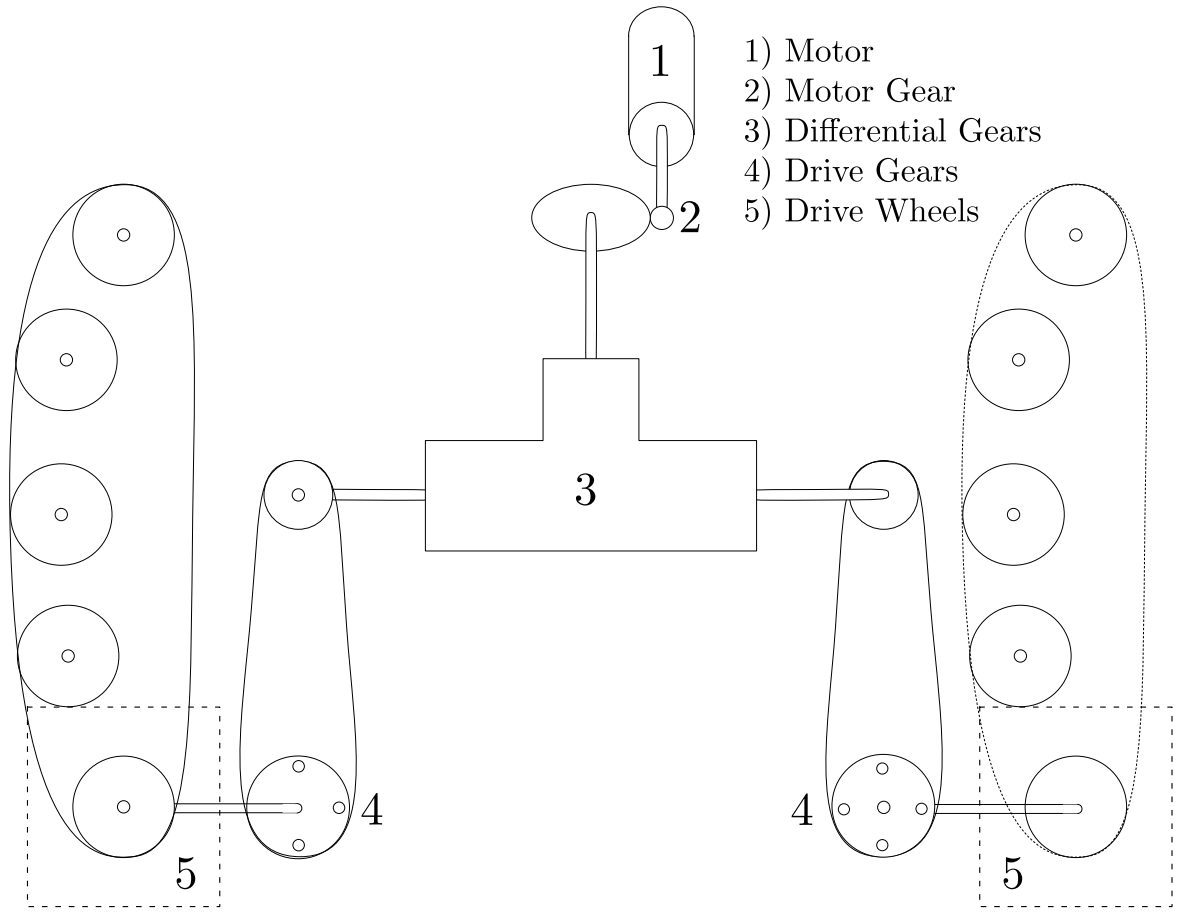


Figure 2.4: Illustration of components included in the drivetrain.

It is known that the motor(1) delivers a force. This force is delivered to the system through a motor-shaft with a connecting gear(2). This gear is connected to the start of the drivetrain. The gear at the start of the drivetrain is connected, through a shaft, to a differential gear box(3). From the differential gears two shafts are connected to a gear, which transfers a rotational velocity to the drive gears(4). From there the rotational velocity makes the drive-wheel(5) turn, thus rotating the belts, which are supported by four free wheels on each side.

In the following segment the differential gears are explained.

Differential Gears

The differential gear ensures that a minimal amount of the energy, received from the motor, is lost when the vehicle is turning. If one of the driving wheels brakes, the differential gears transfer the rotational energy from the braking side to the other side. This will minimize the loss of energy. The extra velocity, transferred to the opposite belt, makes the vehicle turn.

A differential gear system can be seen on *Figure 2.5*.

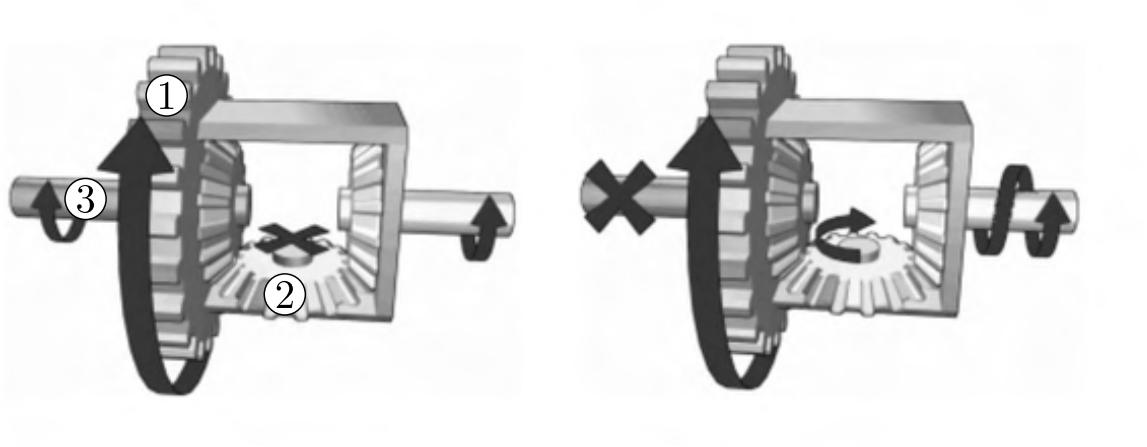


Figure 2.5: Illustration of a differential gear system. On the left, the resistance on each side is equal, so the spider gear(2) does not rotate. On the right, the resistance on the left side is bigger than the one on the right side, so the spider gear rotates and transfers more rotational energy to the right side. [6]

The differential gear system contains a ring gear (1), a spider gear (2), two side gears connected to the rest of the system (3) and a pinion gear (not shown on picture) connected to the ring gear. The pinion gear transfers the energy from the motor to the differential gear.

When the motor is running, the pinion gear transfers a rotational energy to the ring gear. The spider gear, fixed on to the ring gear, begins to rotate around the side gear. The ring gear is statically connected to the frame in which the spider gear is mounted. Therefore the transfer of rotation of the frame happens through the spider gear, which pushes on the side gears. If the resistance on both side gears is the same, see the left side in *Figure 2.5*, the same velocity is transferred to either side gear. Therefore the spider gear will not rotate around its own axis and apply the same rotational energy to each side gear.

When there is a difference in resistance on one side compared to the other, see the right side in *Figure 2.5*, one of the sides brakes and the spider gear will rotate. There is a bigger resistance on one side than on the other, so the side which is not braking will be easier to rotate. The ring gear and the spider gear are rotating at the same velocity around the two side gears. But the spider gear will apply less rotational velocity on the braking side than on the other side. In the case that the resistance on one side is infinite, the side gear on the none braking side will rotate twice as fast, compared to equal transfer of energy to the two sides.

Instead of only having one spider gear, as seen in *Figure 2.5*, there are two on the received vehicle, the functionality is the same, but two spider gears gives more reliability and solidity.

Steering

To steer the vehicle with a certain angle a servomotor is utilized. When the servomotor rotates to a specific angle, which is not the servo's middle position, it will pull on one of the brakes. Thus affecting the output of the differential gear and change the velocity on the belts placed on each side of the vehicle. The difference in belt velocities yields the angle of the moving vehicle.

Brakes

The moving vehicle turns by only pulling on one of it's brakes. if one of the belts is slower than the other it will make the vehicle turn regardless of changes in the motor output torque. A mechanical drawing of one of the brakes can be seen in *Figure 2.6*.

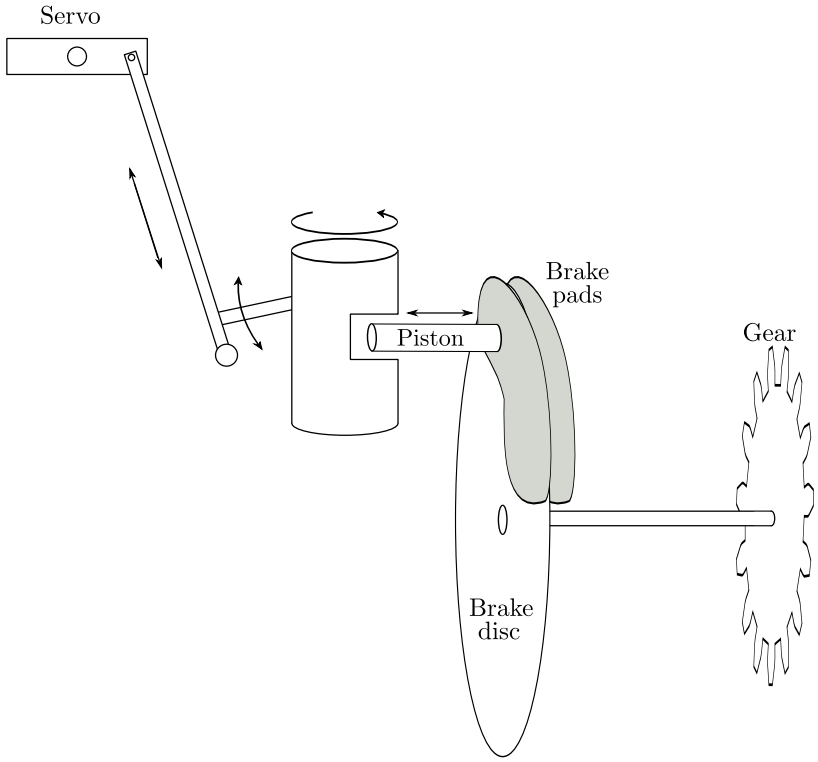


Figure 2.6: Illustration of the servo affecting the brakes and how the brakes are connected to the drive gear.

When the servo rotate to a specific angle, it pulls an arm connected to a rotating cylinder, pushing the piston, seen in *Figure 2.6*, when it rotates. The piston pushes the brake pad closest to it and will thereby add friction to the brake disc, slowing down the rotation.

Servo

The vehicle includes a S3003 servomotor from Futaba [7], which is used for the vehicle's steering. The servo can be seen in *Figure 2.3*. The way it makes the vehicle turn is by braking the gears on either side of the vehicle, and transferring the velocity over to the other side, by utilization of the differential gearbox.

The servo is controlled by a PWM signal with a period of 30000 μs. By applying a specific PWM signal the servo turns to a certain angle, see *Table 2.1*. When the arm rotates one way, it triggers the brake on that side, but does not affect the other side.

Angle	Pulse Width
$-90°$	500 μs
$0°$	1450 μs
$90°$	2500 μs

Table 2.1: Servo positions in relation to PWM signals of period 30000μs

By convention in this project the neutral position ($0°$), of the servomotor, is chosen so that the servo-arm is perpendicular to the vehicle's orientation. This ensures that the servo does not push on the brakes. The associated PWM signal of pulse-width is 1450 μs.
While these angles are those for the servo tested alone, its combination with the rest of the steering plant adds some limitations to the possible servo orientations.

Steering's Behavior

Due to mechanical constraints, the servomotor shall only be used in certain positions. Firstly, the servo can not rotate more than a certain amount in each direction due to the two pulling arms, which are attached to either side. Thus, it can never reach the extreme default positions it would if it were not attached. Secondly, the vehicle's turning is almost negligible and most of all non-linear for small angles of the servo. For these reasons, an offset needs to be applied to jump from the middle position to the edges of the linear areas of steering, see *Figure 2.7*.

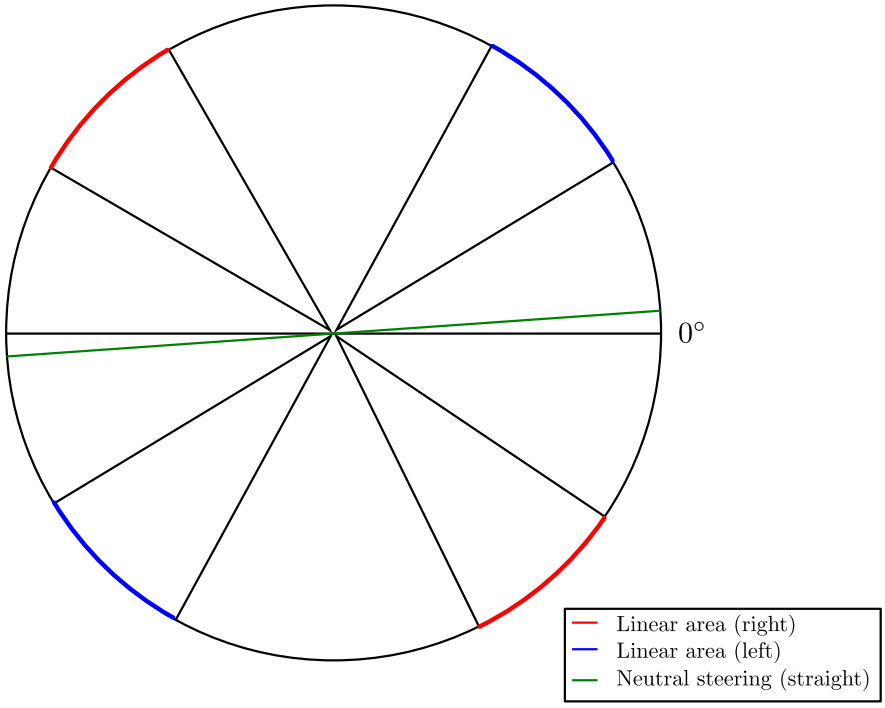


Figure 2.7: A representation of the servomotor's angle positions. Where the red and blue are the linear steering areas and the green is the neutral position, for the vehicle to go straight

The diameters on the circle represent the servo arm which spins around a central position. For example, if the servos arm rotates left, then the left arms pulls back which pushes the brake on the left side, resulting in the vehicle turning left.

The green line, shifted of a few degrees from the 0° is the position allowing the vehicle to go straight. Moreover, the actual pulse width value which allows the vehicle to run straight, is not the same that sets the servomotor to its middle position and may slightly vary through time because of mechanical wear.

The linear areas of steering are represented by the red and blue arcs. To turn to the left, the servo angle has to be increased as well as the pulse width. To go to the right, the servo angle has be decreased together with the pulse width.

Still due to mechanical imperfections on the vehicle, the offset that has to be applied to the servo pulse width, to reach the linear area for steering, is different depending on which way the vehicle is turning. It also varies a bit with the ground texture it is running on, because of the friction and it has to be tuned accordingly. The actual value of this offset is empirically found to be close to ±250 ms.

Hall Sensor

There are two hall sensors implemented on the vehicle, one by each drive gear, illustrated in *Figure 2.4*. Four magnets are placed on each drive gear, with a quarter turn between each. The Hall sensor is illustrated in *Figure 2.8*.

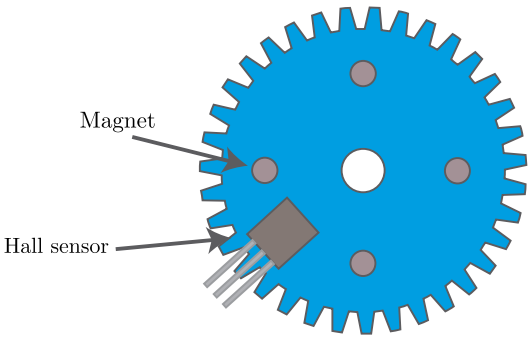


Figure 2.8: Illustration of the drive gear, with the magnets and the Hall sensor. The Hall sensor is placed stationary beside the rotating gear [8].

A Hall sensor is affected by magnetic fields. When the field is significantly larger than the ambient fields a pulse is generated. This gives a voltage output each time one of the magnets is in front of the Hall sensor. Therefore each time the drive gear makes a quarter of a turn a voltage signal is generated.

Battery Pack Specifications

In order to power the motor, the servomotor, and all the electronics, an electrical power source is needed. A NiMH battery pack from Ansmann Racing with $7,2$ V nominal voltage and a 4700 mAh capacity is given with the vehicle [9]. The charge voltage cannot go beyond $9,0$ V and should stay above $6,0$ V.

The provided vehicle and battery have been described. Now it is necessary to understand how the GoT system works, to be able to utilize it to track the vehicle.

GoT Description

The Games on Track GT-Position system, shortened GoT, is a positioning system which uses radio-waves for communication and ultrasound to locate a tracked object in space. The system has a sampling rate of 10 Hz, but it can vary a little because of ultrasound waves propagation. The system is built up from three types of hardware components: the transmitter, the satellites and the master, see *Figure 2.9*. The transmitter and the satellites send the information in all directions, with 360 degrees range signals.
The GoT System does not work if the transmitter is not close enough to the satellites. Furthermore, it can be disturbed if there are some interference by another ultrasound signal or by an object between the satellites.

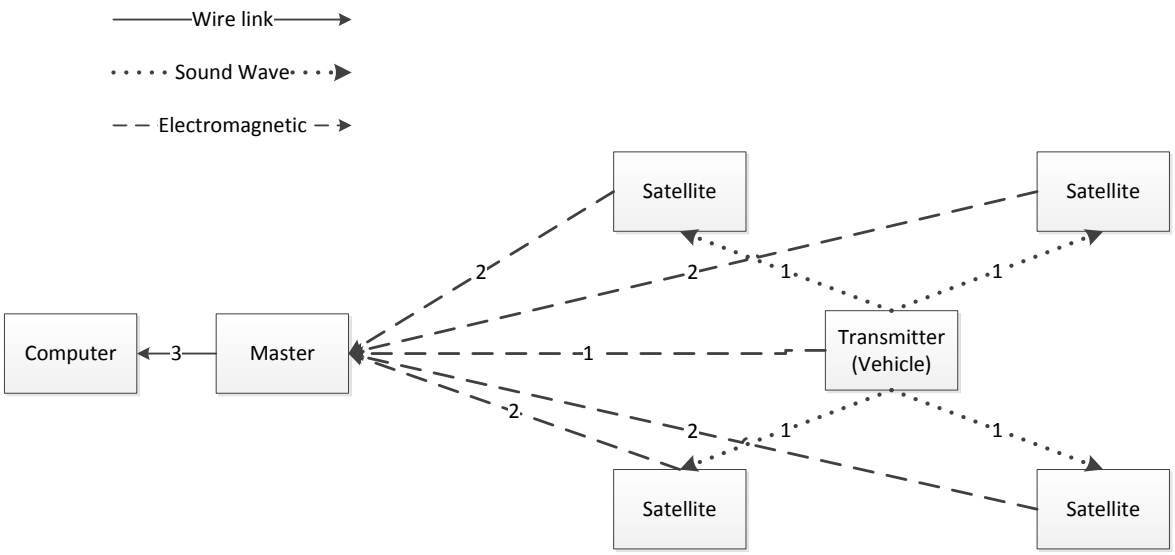


Figure 2.9: Overview of the GoT system, and the placement of hardware components.

Transmitter

The transmitter component is placed on the object which needs to be located. The transmitter sends an ultrasound burst out containing an identification number, that the satellites are

intended to receive. The time at which each ultrasound bursts are sent from the transmitter to the satellites is transmitted by radio waves to the master. An ultrasound burst is sent each tenth of a second, giving a sampling frequency of 10 Hz.
The transmitter component runs on 2 AA-batteries and therefore does not need an external power-source.

Satellites

The satellites components are placed around the area where the object, with the transmitter on it, has to be located. The satellites' assignment is to search for the ultrasound waves, which the transmitter emits, and send a radio wave signal to the master as soon as they receive the burst.

To be able to calculate the exact position of the transmitter and then the object, a minimum of three satellites is necessary. However, more can be added to the system for more reliability, accuracy or to cover a larger area. To work with high efficiency, they should be placed each 1 to 2 meters apart from each other and not on a single line. But to cover a bigger area, they can be placed up to a distance of 5 meters between them, although this would affect the measurement and thereby make it less reliable. Each satellite has a maximum range of 8 meters and the three satellites should be placed in each other's reach. The satellites need between 14 to 20 volts DC, thus making the satellites able to be powered through a computer charger or by a panel solar for example is necessary.

Master

The master is a receiver which is connected to a computer. The master's assignment is to receive the data transmitted from the individual satellites and the transmitter, and to relay it directly to the connected computer. The master is powered through a USB cable, between the master and the computer.

Computer

The program on the computer, which is running in C# and handles the information received from the master, uses the data to calculate the position of the transmitter. This is done with a method call trilateration. Trilateration is a way of calculating a position in a three-dimensional space, from three distances of known locations, with the help of spheres, circles and triangles as seen on *Figure 2.10*.

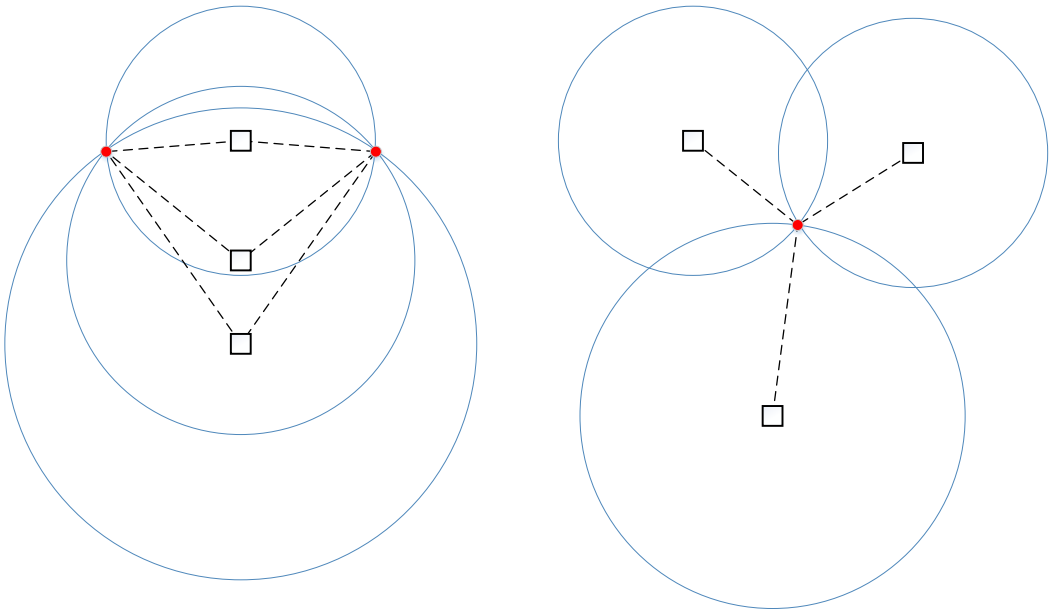


Figure 2.10: To the left, satellites are aligned and will give two possible positions. To the right, satellites are placed in a triangle and will give one possible position

If the satellites have been moved, it is necessary to recalibrate the system. This is done with a calibration triangle. The calibration triangle is made of three points on a flat surface and have a distance of 40 to 200 centimeters between them. One of the points on the calibration triangle is made the origin (0,0,0) of the new coordinate system. Another point on the triangle will then be called (X,0,0), in which the line between the first point and the second point will become the X-axis. The last point will be call (X,Y,0) and will determine in which way the positive Y-axis will go. The surface which the calibration triangle is placed on, will be the XY-plan, where Z will go vertically, defining the (X,Y,Z) coordinates. Thereafter, the transmitter is placed in the three point, with (0,0,0) first and (X,Y,0) last. When the transmitter is placed in a point, the satellites measure the distance between them and the transmitter. From this information, the program can calculate the position of each satellites, with the help of trilateration.

These are the basic elements from which the prototype is built. However, the functionalities suggested in *Section 2.1* need to be refined for the prototype implementation.

2.3 Prototype Constraints

Before the prototype can be established, some considerations have to be made in respect to time limitations and the main scope of this semester. The aim of the project is to create a functional automated proof of concept lawn mower. The following section provides argumentation for eliminated functionalities from the use case in *Section 2.1*.

Grass Length Detection

Detection of the grass length to control the velocity of the lawn mower thus ensuring an evenly cut lawn, is a submodule which can be added at any time. Since it is not fatal for a working system and might even be unnecessary depending on time between each mowing of the lawn, it is decided to exclude this functionality from the initial design.

Humidity Sensor

As the lawn mower is supposed to work outside, it is important to consider that the grass could be humid. Since it is difficult to cut wet grass, a humidity sensor could be used to warn the system about the humidity, thus the system could go back to the charging station. This submodule is not fatal for a functional prototype, so this type of sensor will not be included in the design.

Obstacle Avoidance

The lawn mover's path might not always be clear, e.g. garden tools, tables or moving objects could be in the way. The vehicle should be aware of what is in front of it at any time, to correct its path and be able to get around the obstacle if necessary. To avoid this the sensor could be a pushing button to detect a solid object or an ultrasound detector if the object is fragile. As the aim of the project is to control the path of the vehicle by using angular positioning sensors, a proximity sensor will not be included. Static objects could be registered on the map to avoid these issues.

Furthermore the edge mapping functionality will not be included in the project which instead will focus on a map predefined in the test room.

Blade Control & Blade Monitor

To get the blade to cut the grass evenly over the whole lawn, the blade control and monitor is needed to keep the blade at the same velocity. The blade, and therefore the blade control and monitor, will not be in the prototype. This is because, the focus of the project is on the movement of the vehicle and the blade does not contribute to this and is therefore not relevant.

Route Planning

Route planning is a way of optimizing the lawn mowers time on the lawn, instead of the classic solution with a random cut system, described in *Section 1.1*. The prototype will not have an algorithm calculating a route from the vehicle's position, the battery time and information received from the edge map. The route in which to follow is a predefined route made to illustrate and test the essential part of the prototype.

Testing

The testing will take place in Aalborg University Vicon Room, where the GoT system is installed, see *Section 2.2*, is installed and calibrated with the appurtenant transmitter, which is mounted on the tracked vehicle during test.

A description of the system and arguments for eliminating functionalities from the initial design has been presented in the preceding segment. The prototype can now be established in regards to the equipment available and remaining functionalities.

2.4 Prototype, Interfaces and Submodules

The overall functionalities for the project have been constrained from the Use-Case *Section 2.1*, due to time limitations and to focus on the main scope of the semester. A prototype is therefore made to illustrate the main functionalities necessary to make an automated vehicle, containing principles for lawn mowing.

The final prototype includes a regulator, which will make it possible to follow a path from A to B. It is able to continue if the wireless connection is lost between the prototype and the GoT system for some duration. The outline of the design is shown in *Figure 2.11* to give an idea of the final prototype setup. In this section, the different functional blocks are put forth along with the interfaces describing the interaction between them.

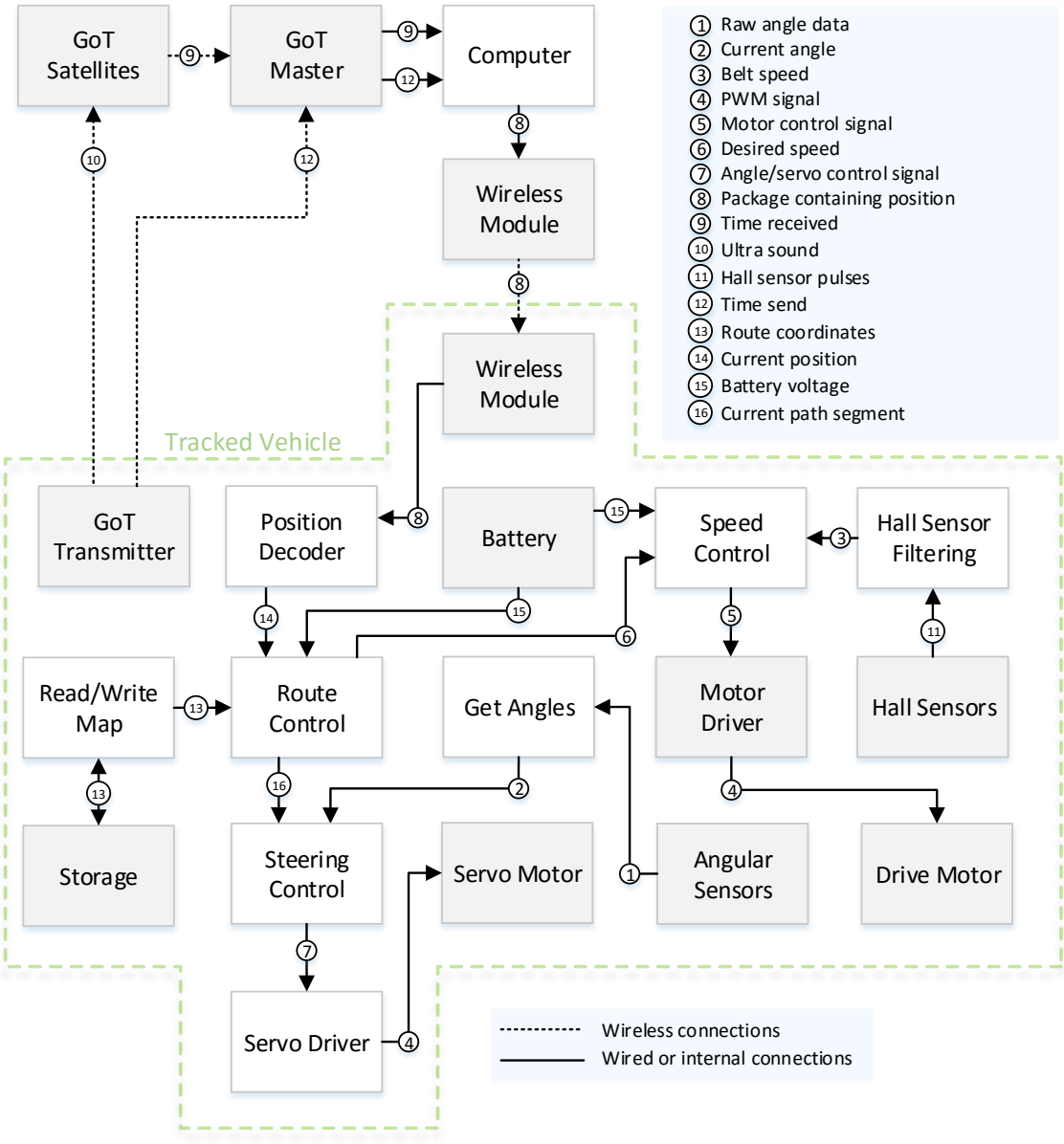


Figure 2.11: Overview of the software and electronic parts of the system prototype, where the gray modules are hardware and the white are software.

Modules

In the following all the modules from the system prototype which needs further explanation are described to obtain a basic understanding of the prototype.

Computer and Position Decoder: The computer handles the calculation of the current position of the vehicle *Section 2.2*. If a packet is disturbed or sent incorrectly it should be

possible to detect it, so invalid data is not used by the prototype. It may occur that the coordinates which are sent from the GoT system is out of sensible range, e.g the coordinate transmitted jumps from one location to a location which is unrealistic for the vehicle to reach in the amount of time between received coordinates. In this event the out of range coordinate should also be disregarded. The computer must compile a package containing the coordinates. The position decoder must be able to decode the transmitted package.

Steering Control: This module makes sure that the vehicle stays on its path, which it receives from storage. To achieve this the steering control also receives the current path segment of the vehicle, which it uses to calculate a desired angle. The desired angle is used in comparison with the current angle in the steering control to regulate course of the vehicle.

Read/Write Map: This module handles the writing of coordinates to the non-volatile storage device, as well as the reading of next desired destination from the stored route.

Speed Control: This module retrieves the speed of the vehicle from the Hall Sensor Filtering module. This is used, through control, to obtain a steady speed. Furthermore it handles any request of change in speed delivered by the course correction.

Interfaces

In this section a high layer interface between the modules will be presented. This provides information for designing each of the adjacent submodules individually.

Package Containing Position (8): The data communicated from the GoT system to the Position Decoder on the vehicle contains the last recorded position of the vehicle. This position is presented in the form of an x- and a y-coordinate, which must be included in the data package. Additionally each package must contain decode information for the receiver, including how to separate each package along with error handling.

Angle/Servo Control Signal (7) and PWM Signal (4): Course correction will ask the vehicle to turn or make small adjustments to the angle. This angle adjustment is realized through the servomotor which brakes on either belt appropriate to the required angle. For the servo to understand the angle, the servo driver must translate it into a PWM signal, which makes the servo turn to a specific position for each pulse width.

Route Coordinates (13), Current Position(14) and Battery Voltage (15): Route coordinates are extracted from the storage and paired with the current position given by the GoT system. This coordinate pair yields a path segment for the vehicle to follow, which is forwarded to steering control.

Raw Angle Data (1), Current Angle (2) and Current Path Segment (16): The raw angle data is translated to a current angle of the vehicle. Current angle is then used to calculate the desired angle using the path segment given by route planning.

Desired Speed (6), Motor Control Signal (5) and PWM Signal (4): Desired speed is

regulated by the route control because the route planner knows when the vehicle must turn, in which case the speed should be lowered, if desired. The speed controller makes sure that the vehicle stays at this speed, until told otherwise, by regulation of the motor control signal. This signal is then translated, by the motor driver, to a PWM signal for the motor.

Hall Sensor Pulses (11) and Belt Speed (3): To determine the speed of the vehicle, two Hall sensors are placed by the drive wheels. These generate pulses transmitted through the filter, which translates it into two separate belt speeds averaged as one for speed control.

Now that the vehicle is described and the prototype defined, requirements for the prototype can be established.

3 | Prototype Requirements

Based on the preanalysis, the use case, system description and prototype, it is possible to establish requirements for the prototype.

1. **It shall be possible for the vehicle to receive its own location wirelessly from the GoT system, through a computer.**

   The vehicle needs to be able to receive its own location, additionally, it should be wireless so the vehicle can drive without cables attached, and thereby be able to move freely, *Section 2.4*

2. **It shall be possible for the prototype to disregard incorrect packets transmitted from the computer**

   To ensure that invalid data is not utilized, *Section 2.4*

3. **The prototype must be able to disregard erroneous coordinates sent from the GoT system**

   If the GoT system delivers an unrealistic coordinate, e.g. created by noise, to the vehicle, *Section 2.4*.

4. **The prototype must be able to access the route, which it has to follow, from a storage space located on the vehicle**

   To simulate the use-case, see *Section 2.1*, the route needs to be stored on the vehicle, and thereby making it accessible for the functionality course correction, *Section 2.4*.

5. **The prototype must be able to shut down, if the battery voltage is below its cut-off specification**

   To ensure the battery is not damaged, *Section 2.4*.

6. **It shall be possible for the prototype to follow a predetermined route**

   This is to simulate a lawn mower, which is able to cut grass by following a route, *Section 2.3*

7. **It shall be possible for the prototype to return to the predetermined route if disturbed**

   E.g. if the vehicle slips or is moved, *Section 2.1*

8. **The prototype shall be able to keep a velocity on 1,4 $m \cdot s^{-1}$, when going up - or downhill and when turning**

   To ensure an evenly cut of the grass, *Section 2.1*

Based on the established requirements it is now possible to design a prototype of an autonomous vehicle.

Part II

Design & Implementation

4 | Hardware and Software Considerations

From the established requirements for the prototype, it is possible to choose which hardware and software should be utilized. Furthermore, the overall design and implementation of the hardware and software are described.

4.1 Hardware Choice

From the prototype description, see *Section 2.4* it is possible to choose compatible hardware for building the prototype. Some of the hardware components are given with the vehicle, see *Section 2.2*, and therefore, are not described in this section. The availability of the components and their compatibility with the system also have to be considered.

Microcontroller

The microcontroller is utilized to control the system and the connected hardware. Thus containing software used for controlling the system by sending and receiving electrical signals as needed.

Requirements for the microcontroller:

- Having I/O connections, both digital and analogue, see *Section 2.2*.
- Having output connections, that can transmit PWM signals, see *Section 2.2*.
- Having 5 free timers. Two for the Hall sensors and one for the Real Time Operating System, motor, servo-motor.

Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller board, which comes with an 8-bit ATmega2560 integrated circuit and extends its I/O ports, see *Figure 4.1* [10].

Figure 4.1: An Arduino Mega 2560 board [10]

The Arduino Mega has 54 digital ports with a 5 volts logic. From these, 15 can be used to generate PWM signals. The CPU runs at 16MHz and the chip has 6 integrated timers, one of which is used by the Arduino itself. There are also 4 UARTs, used in serial communication, as well as a setup for SPI communication and an I^2C bus. The Arduino can be powered though a USB cable or with an external power source.

The Arduino Mega board has been chosen as the microcontroller utilized. The reason is that the connections needed are available. Furthermore, it is possible to find a lot of helpful documentation on the Arduino Mega on the internet. The Arduino is programmable through a serial connection, with the help of avrdude[11] or simply through the Arduino IDE[12]. The IDE can compile C, C++ and Arduino (syntax very close to C and C++) files.

Storage

The storage is used for saving the predetermined route for the vehicle see *Section 2.4*.

An approximation of the needed storage size for the mowing route is made by considering, a lawn of 10 m by 10 m and a simple path with parallel lines which require two points each. Since a route point is defined by two coordinates x and y of size 4 B in total, and the vehicle is 29 cm wide, see *Section 2.2*, the storage capacity is approximated to:

$$\text{Capacity} = \frac{10}{0,29} \cdot 2 \cdot 2 \approx 276 \text{ B} \tag{4.1}$$

This allows to store data points for the lawn mower to cover the whole area without overlapping and with a very simple route. Since this is only a rough calculation, a SD card with a capacity of more than this should be chosen for containment of a predetermined route. Furthermore it would be practical to utilize the SD card when testing. Therefore a SD card with a capacity of at least 1 GB should be chosen.

To find an estimate of the required storage transfer rate when retrieving data, the steering control requirements in terms of sampling frequency are considered, this is found in *Section 8.2*, as this utilizes the route coordinates. Since the servomotor needs to receive data every 30 ms [7], the receiving of data from storage needs to be done in the meantime. The other tasks also needs a certain time to run. This time can be approximated to 2 ms, see *Section 4.4*, and has

to be substracted from the 30 ms. Furthermore, the data transfer during each turn should not take too much time, and it is arbitrary decided to take a tenth of the available time. As each point weighs 2 B, the minimum transfer rate needed is:

$$\text{Transfer rate} = \frac{1}{\frac{0{,}030-0{,}002}{10}} \cdot 2 \approx 714,3 \text{ B} = 7,2 \text{ B} \cdot \text{s}^{-1} \tag{4.2}$$

Requirements for the storage:

- Offering the possibility to retrieve the data after a power cut to the storage.
- Having a storage of at least 1 GB.
- Having a transfer speed greater than $7,2 \text{ KB} \cdot \text{s}^{-1}$.

SD Card

Secure Digital (SD) card is a type of non-volatile flash memory cards. This means that the data will not be lost during a power cut off.

It comes in various storage sizes, from 1 GB to 2 TB, depending on which type of SD card it is[13]. There is 3 types of SD card, the standard capacity (SDSC), the high capacity (SDHC) and the extended capacity (SDXC). The difference between the different types, is the file system utilized on the card and their maximum capacity. With the requirement of a storage size on 1 GB bytes, and a desired need for utilizing it to the contain test data, a SDSC with a capacity of 2 GB is chosen.

The lowest transfer speed for an SDSC card is 2 MB/s[13]. With the requirement of a transfer speed greater than $7,2 \text{ KB} \cdot \text{s}^{-1}$, makes the SDSC card fast enough.

To use the SDSC card and connect it to the microcontroller, 7 connection pins are required, see *Figure 4.2*. The setup of the SD card is different, depended on which mode is used.

Pin	SPI	One bit	Four bit
1	Unused	Unused	Data 2
2	Card Select	Card Detection	Data 3
3	Data In	Command & Response	Command & Response
4	Power	Power	Power
5	Serial clock	Serial clock	Serial clock
6	Ground	Ground	Ground
7	Data Out	Data 0	Data 0
8	Unused	Unused	Data 1

Table 4.1: SD card pinout configuration[14]

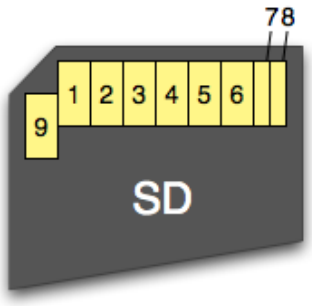


Figure 4.2: Illustration of a micro size SD card.[14]

The power needed for the SD card is 3,3 volts[14]. There are three possible setups for the SD card: SPI, One-Bit and Four-Bit. With the SPI setup, the communication between the SD card and the microcontroller is made on two lines. With the One-Bit setup, the commands between the SD card and microcontroller are sent on one line and the data is sent through another one. With the Four-Bit setup, three more data connections are used.

In this project, the SPI setup is used, since the extra transfer speed from the Four-Bit setup is not needed. Furthermore, the Arduino has 4 pins that can be set up to SPI communication[10].

An SDSC card, with a SPI setup is chosen, since this as has a high enough capacity and transfer speed. Furthermore the data is saved, if the power is cut off and the Arduino Mega has SPI connections.

Motor Driver

The motor driver is the connection between the microcontroller, the power supply and the motor. It is used, to isolate the hardware components functioning in low power, i.e. the microcontroller, from the high power parts, i.e. the motor.

The requirements for the motor driver are:

- Having to be controlled by the microcontroller.
- Offering the possibility to power it through an external power source.
- Being able to handle the battery pack's voltage.
- Allowing to power the motor in both directions.

Pololu Dual VNH5019

The Pololu dual VHN5019 motor driver shield (see *Figure 4.3*) is specially designed for Arduino boards. It can be placed directly on top of the the Arduino board as a shield, with the pins connecting to the right ports on the Arduino.[15]

Figure 4.3: The motor shield on top of an Arduino Uno.[16]

It is possible to power the Arduino from an external power source through the motor shield, which also delivers the power to the motor. It is also possible to have the motor shield and the Arduino powered separately. This is determined by a jumper setting on the shield, see *Figure 4.4*. The motor shield need a power voltage between 7 volts and 12 volts.[15]

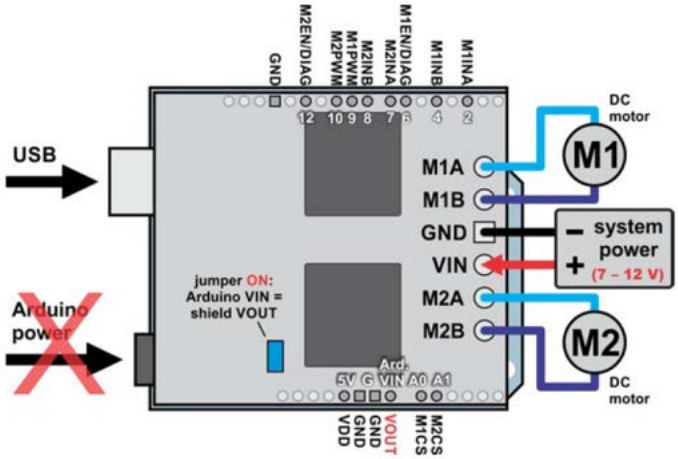


Figure 4.4: Setup for the mort shield.[16]

There are two H-bridges on the board, to control up to two motors. Since only one motor is used in this project, both of the two H-bridges will be used to power the motor, see further details on the configuration in *Section 4.3*. This results in a smaller current through the two H-bridges' transistors and therefore ensures a better protection of the system.[15]

The Pololu dual VHN5019 motor driver shield is used in this project, as it is made for the Arduino and therefore easy to implement. Moreover, it can power the Arduino through an external power source and get a power output high enough for the motor and in each direction.[17]

Wireless communication system

The data from the GoT system should be transmitted to the microcontroller with a wireless communication system, *Section 3*. The transmitter will be located on the computer, utilized for the GoT system, and the receiver on the vehicle. As the prototype will be tested in the control lab, the distance that the wireless communication have to send is less than 10 meters.

The GoT system has a sampling frequency of 10 Hz, see *Section 2.2*. Each time a set of coordinates is measured, the x and y components have to be sent, see *Section 6.3*. Each coordinate consists of 2 bytes, but more bits are needed when communicating, to account for the transmission protocols' data. It is arbitrary chosen to use at least 2 bits for this data. This yields:

$$\text{Transfer rate} = (2 \cdot 8 + 2) \cdot 10 = 180\,\text{Kb} \cdot \text{s}^{-1} \tag{4.3}$$

This is the minimum transfer rate needed for communication between the computer associated to the GoT system and the vehicle.

Requirements for the wireless communication components:

- It possible to control it utilizing the Arduino Mega.
- Having a range greater than 10 meters.
- Transfering at a higher rate than $180\,\text{Kb} \cdot \text{s}^{-1}$.
- Offering the possibility to control it and couple it with the GoT code, located on the computer.

Xbee

Xbees are small radio modules, that are easy to set up. An overview of the Xbee and its pin layout can be seen on *Figure 4.5* and *Figure 4.6*.

Figure 4.5: Two Xbee radio modules[18]

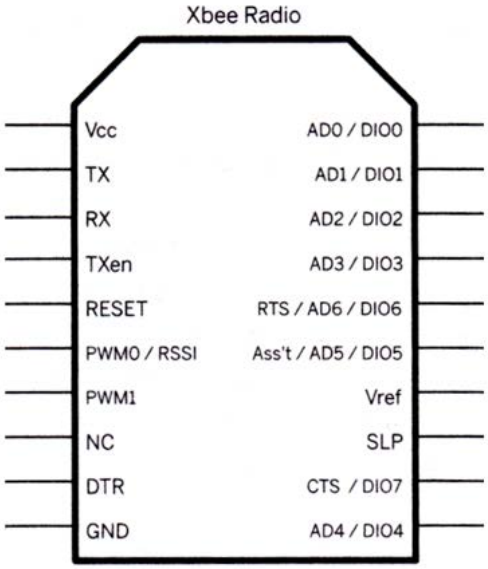


Figure 4.6: Pinout of an Xbee radio module[19]

The Xbee modules communicate through an UART connection (with TX and RX pins)[20]. Since the Arduino has three UART connections, plus the one used to program the Arduino, the Xbee can be connected. The software code for the GoT system is implemented in C# (see in *Section 2.2*) and the computer running has serial ports that can ensure the connection between the software and the Xbee. Therefore, this solution can be used for a wireless connection between the GoT system and the Arduino.

To run the Xbee modules, a $3,3$ V power line, a ground line, and a $3,3$ V logic UART are needed[20]. The modules have a transfer speed of up to $115,2$ kbit/s and can reach up to 100 m indoor and 300m outdoor. It transmits at a $2,4$ GHz frequency with 1 mV (0 dBm) and can receive downto -96 dBm[20]. The Xbee is already ready for communication, with the lower levels of communication, the physical and data link layers, being already implemented. With only two devices, a lightweight combination of transport layer can be added to the protocol directly on top of the data link layer, to add more error handling and create the transmitted packet.

Angular sensor

The angular sensor is used in the feedback for the control system.

The servomotor needs to receive commands every 30 ms, [7]. The measurements from the

angular sensor will therefore also be sampled at this frequency.

$$\text{Sampling frequency} = \frac{1}{0,03} = 33,3 \text{ Hz} \tag{4.4}$$

The requirements for the angular sensor are:

- It is possible to control it by utilizing the Arduino Mega.
- Able to achieve a sampling frequency at 33,3 Hz.

HMC5883L

Sparkfun's "9 degrees of freedom" board[21] comprises a magnetometer integrated circuit, the HMC5883L. A magnetometer will be used as the angular sensor, since it can be set up as a compass and give out a angle compared to north.

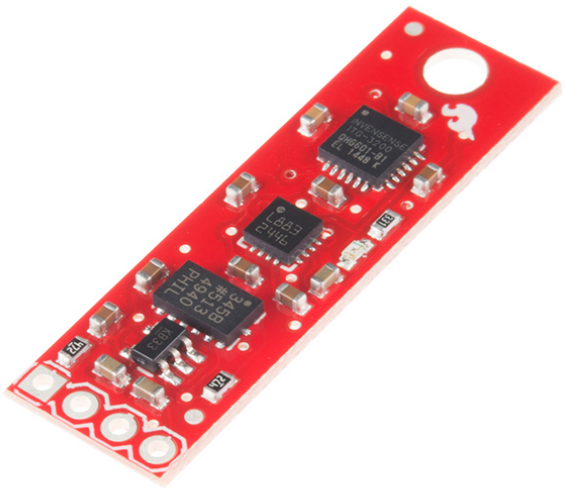


Figure 4.7: Sparksfun's "9 degrees of freedom" sensor stick[21]

The "9 degrees of freedom" sensor stick, see *Figure 4.7*, will be used in the project. Beside the magnetometer, there are also a accelerometer and a gyroscope on the board, but these will not be used. The "9 degrees of freedom" sensor stick is designed to be used with a microcontroller and comes with software example for the Arduino platform. It needs an I^2C bus to communicate with the Arduino, which has that kind of interface.

Power monitor

The power monitor is used to give a feedback to the system about the level of power in the battery pack.

Requirements for the power monitor:

- Can be monitored by the Arduino Mega.
- Measuring the battery pack voltage output downto 6 V, *Section 2.2*.

Voltage divider

A voltage divider is an electronic circuit, that divides the input voltage with a constant depending on the values of the resistors used in the circuit, see *Figure 4.8*.

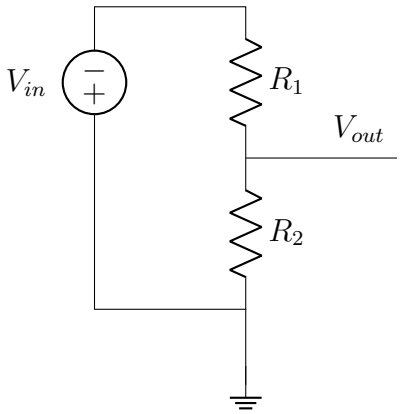


Figure 4.8: Standard voltage divider

The output of the voltage divider can be found with *Equation (4.5)*.

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in} \quad [V] \quad (4.5)$$

Where:

V_{in}	is the battery pack voltage	[V]
R_1, R_2	are two calculated resistors	[Ω]
V_{out}	is the measured output voltage	[V]

The input voltage is set to be the battery pack and the output is measured though an analogue pin on the Arduino. The analogue pin on the Arduino can measure from 0 V to 5 V with a resolution of 10 bits. This gives 1023 steps from 0 to 5 volts and each step is 4, 9 mV. The size of each step and the range for the measuring is multiplied with the inverse constant, that comes from the two resistor, so it is possible to make the range go over 5 V.

A voltage divider is used in this project, as it can be scaled to the interval that it has to measure. The voltage divider will be connected to an analogue pin on the Arduino, which will make the measurement from the output of the voltage divider.

From the preanalysis it has been possible to choose hardware which needs to be utilized. After having chosen the different hardware components for the prototype, the different components have to be implemented in the system.

4.2 Hardware Implementation

With the hardware components being chosen, it is necessary to assemble them, ensure that they are compatible for communication with the microcontroller, and that they receive the appropriate electrical power. Other implementation considerations also have to be considered. e.g , the voltage divider for the power monitor.

Microcontroller peripherals

Before choosing which pins on the Arduino Mega to connect the various hardware to, the peripherals[1] in the microcontroller needs to be evaluated (each peripheral is only available on certain pins). First, the timers are considered. The microcontroller has 6 hardware timers, Timer 0 to Timer 5.

Timer 0 is reserved by the Arduino for it's internal time keeping functions, and not to be touched [22]. This leaves 5 timers, that can be used freely.

It would be advantageous to utilize the input capture[2] functionality of the timers to read the hall sensors, since it will provide a hardware generated timestamp, and be independent of software latency. After examining the Arduino Mega pinout, *Appendix P*, it is seen that only Timer 4 and 5 have their input capture pins available on the board (ICP4 and ICP5). These are therefore chosen for the two hall sensors.

The real-time operating system needs a timer, for running its scheduler. Timer 1 is chosen for this, for no other reason than it being the default setting.

This leaves Timer 2 and 3. According to [23], Timer 2 is an 8 bit timer, and Timer 3 is 16 bit.

Timer 3 is chosen for the PWM signal sent to the servo. This, because the higher resolution makes it easier to generate the short duty cycles required by the servo (500 μs to 2500 μs on-time, in a 30 ms period, see *Subsection 2.2*)

The last timer is chosen for the PWM signal to the DC-motor.

As the timers have now been designated, the next step is to assign the various serial ports. This is done as shown on *Figure 4.9*. Considerations about the powering of the hardware modules is seen in *Section 4.2*.

[1] Hardware modules, such as timers, UARTs, etc.

[2] An input capture timer samples a free running counter, whenever triggered by an external event

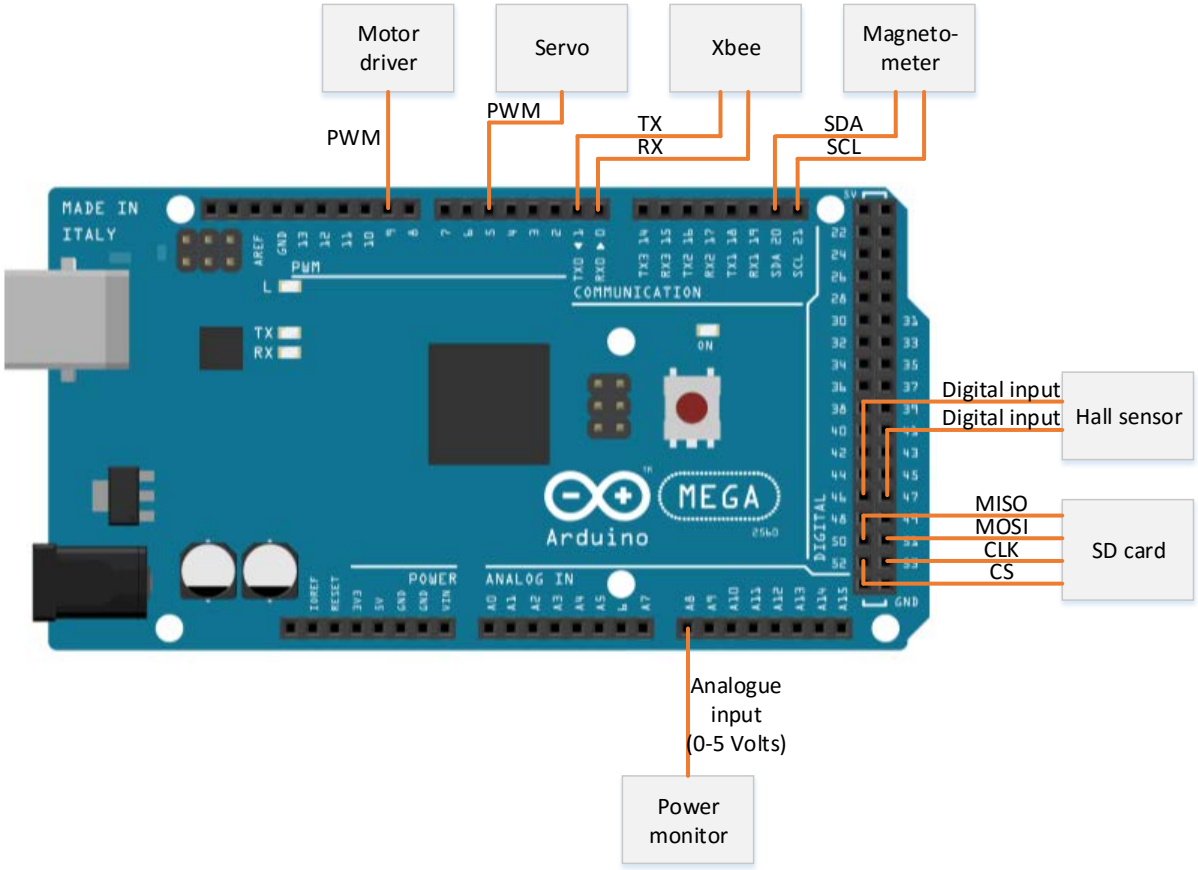


Figure 4.9: Connections between the different hardware components and the Arduino.[24]

The XBee module needs to connect to a serial port, and UART3 is chosen, as it makes it easier to route the PCB traces. UART3 is located at pins 0 and 1.

The SD card needs an SPI connection, and must therefore be connected to the SPI pins 50, 51, 52 and 53.

The "9 Degrees of Freedom" board needs an I^2C connection, and has to be connected to the Arduino's I^2C pins 20 and 21.

The PWM signals sent to the motor and servo can be set to any of the PWM pins. For easier routing, pin 9 for the motor and pin 5 for the servo is selected.

The input from the hall sensors will be measured by the input capture pins (ICP4 and ICP5), to which Timer 4 and Timer 5 are connected. On the Arduino board, it corresponds to the pins 46 and 47.

Lastly, the voltage divider used for the power monitoring is connected to an analogue pin chosen to be pin A9.

As all hardware modules have now been assigned pins on the Arduino, the electrical requirements of each module will now be considered.

Electrical considerations

Before connecting hardware modules to the Arduino pins, signal voltage levels also need to be considered:

- The Arduino Mega 2560 itself uses 5V logic levels.[10]
- The "9 Degrees of Freedom" board needs 3,3 V-16 V supply and 3,3 V logic levels. [25]
- The servo needs 4,8 V to 6 V supply voltage and signal levels.[7]
- The SD card needs 3,3 V supply voltage and signal levels, see *Subsection 4.1*.
- The Hall sensors needs 3,5 V to 24 V supply, and a pull up resistor to define the logic level. [26]
- The XBee module needs 3,3 V supply voltage and signal levels. *Subsection 4.1*

The Arduino has regulated 5V and 3,3 V supply rails available, which will be used to power the respectable hardware modules. As the servo can potentially draw a lot of current, and overload the Arduino, it will be connected to a dedicated 5V voltage regulator, powered directly from the battery.

A low-dropout type is chosen, L4941, to make sure it works, even at low battery voltages. This regulator has a maximum dropout of 700mV at 1A current, meaning that it will work with battery voltages as low as 5,7 V. [27]

All one-directional connections from the Arduino to 3,3 V logic level inputs will be connected to a HEF4050 CMOS buffer. This device can accept input voltages up to 15V, while supplied by as little as 3V supply voltage [28]. This makes it ideal for uni-directional voltage level translation.

One-directional connections from 3,3 V logic level output to Arduino inputs will be connected directly, as 3,3 V is above the minimum high level threshold [23].

$V_{IH,min} = 0,6 \cdot V_{cc}$
$V_{IH,min} = 0,6 \cdot 5$
$V_{IH,min} = 3\ V$

As the I^2C bus is using bi-directional connections, and needs to connect a 3,3 V system to a 5V system, this needs to be addressed as well. One solution could be running the bus at 3,3 V, but this is unfortunately not possible, as the microcontroller needs at least 3,5 V as HIGH voltage, when in I^2C mode [23]. The creators of I^2C, Philips, recommend solving this with two MOSFETs [29], see *Figure 4.10*.

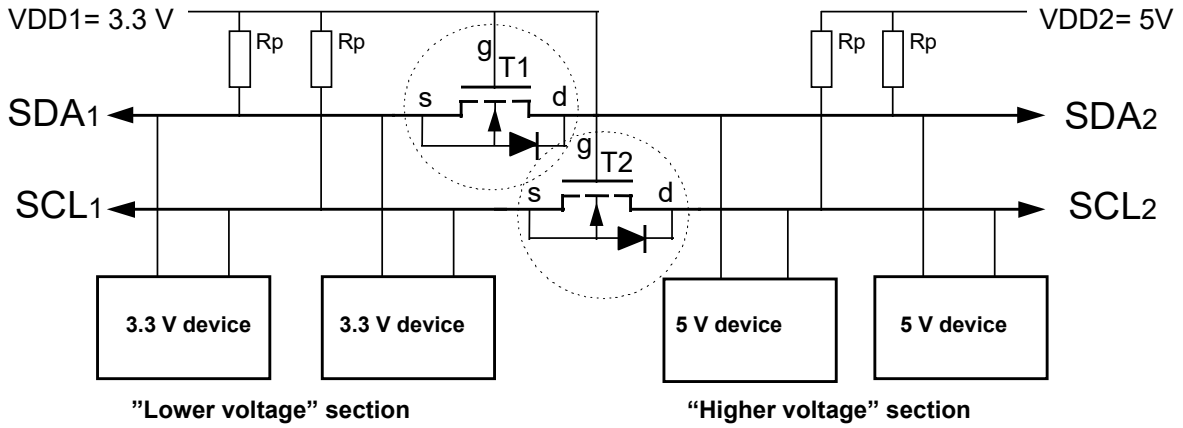


Figure 4.10: I^{2}C Voltage level translator. [29]

In I^{2}C systems, the nodes can only pull the line LOW. To make it possible to signal a HIGH, pull-up resistors are used. If a low voltage system is directly connected to a higher voltage system, the voltage at the input pins of the low-voltage section risk being pulled up to 5V.

To solve this, the translator uses a MOSFET per signal line, with the gate connected to the lower supply voltage, to separate the two systems. Here is a quick explanation, examining just one line (as both lines are implemented identically):
The gate of the MOSFET is connected to 3,3 V. Four states are possible, both sides HIGH, both sides LOW, or one of each. If both ends are in the HIGH state, the voltage on the source pin will be 3,3 V, and the gate-source voltage will therefore be $3{,}3\,\mathrm{V} - 3{,}3\,\mathrm{V} = 0\,\mathrm{V}$. The MOSFET is therefore off, and not conducting. This prevents the 5V supply from reaching the input of the 3,3 V side.
If the 3,3 V section is in the LOW state, the voltage on the source pin will be 0V, and the gate-source voltage will therefore be $3{,}3\,\mathrm{V} - 0\,\mathrm{V} = 3{,}3\,\mathrm{V}$, turning the MOSFET on. This in turn pulls the 5V section to 0V, signalling a LOW.
If the 5V voltage section is in the LOW state, the intrinsic diode in the MOSFET will conduct, pulling the source pin down to 0 V plus a diode drop ($\approx 0{,}6\,\mathrm{V}$). This results in a gate-source voltage of $3{,}3\,\mathrm{V} - 0{,}6\,\mathrm{V} = 2{,}7\,\mathrm{V}$, turning the MOSFET on. This will eliminate the diode drop, and the result will be 0V on the source pin, signalling a LOW to the 3,3 V side.

For this to work, a MOSFET that will turn on at 2,7 V is needed. FDV303N is available and choosen, as it has a V_{ds} threshold voltage[3] of maximum 1,5 V. At 2,7 V, the drain-source resistance is specified to $0{,}6\,\Omega$, meaning that the MOSFET will be on at this voltage[30].

Voltage divider

The power monitor need a voltage divider, see *Section 4.1*. With a measurement interval of 0 to 5 volts in 1023 steps for the analogue pin on the Arduino, the voltage divider has to be designed to be able measure the whole interval for the battery pack. As the battery pack can have a voltage up to $9,0$ V[9], the transfer constant for the voltage divider need to be smaller than 0,59 to measure the whole interval. The smaller the transfer constant, the bigger the interval is and

[3]The V_{ds} threshold is the voltage, where the MOSFET starts to conduct

the bigger each step will be. To make sure that there will not come a bigger voltage than 5 volts on the output from the voltage divider, the transfer constant is set to 0,5: this will give a maximum voltage into the voltage divider of 10 volts before the output voltage goes beyond 5 volts. To get a transfer constant of 0,5, the resistors in the voltage divider have to be the same, see *Equation (4.6)*.
If,

$$R = R_1 = R_2 \quad [\Omega]$$

Then,

$$\mathrm{V_{out}} = \frac{\mathrm{R}}{\mathrm{R}+\mathrm{R}} \cdot \mathrm{V_{in}} \quad [\mathrm{V}]$$

$$\mathrm{V_{out}} = \frac{1}{2} \cdot \mathrm{V_{in}} \quad [\mathrm{V}] \quad (4.6)$$

For the size of the resistors, the output impedance for the voltage divider have to be taken into consideration. For the analogue pins on the Arduino, it is recommended to have a output impedance smaller than 10 kΩ, or the it will begin to disturb the reading on the pin.
To make sure that this does not happen, the resistors in the voltage divider are set to half of the maximum output impedance, 5 kΩ. The closest resistor in the E96 series is $4,99$ kΩ, which will be used for both resistors.

The final voltage divider circuit for the power monitor can be seen on *Figure 4.11*.

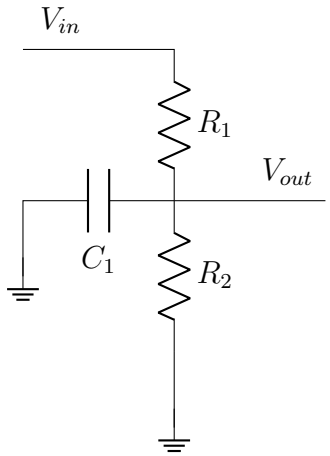


Figure 4.11: The implemented voltage divider

To filter the high frequency noise out, a simple low-pass RC filter is finally implemented with a 1 μF capacitor in the voltage divider. The resulting cut-off frequency is then:

$$\omega_c = \frac{1}{R \cdot C}\mathrm{rad} \cdot \mathrm{s}^{-1} = \frac{1}{2{,}5 \cdot 10^3 \cdot 1 \cdot 10^{-6}}\mathrm{rad} \cdot s^{-1} = 400 \ \mathrm{rad} \cdot \mathrm{s}^{-1}$$

With all the hardware components physically implemented on the vehicle and connected to the microcontroller, the software utilized needs to be determined and designed to enable interactions between the Arduino and all the hardware components.

4.3 H-Bridge

The H-bridge see *Figure 4.12* is a circuit used for motor control. To control the motor some FETs are controlled with a PWM signal. The duty cycle of this PWM signal determines at which velocity the motor will run.[31]

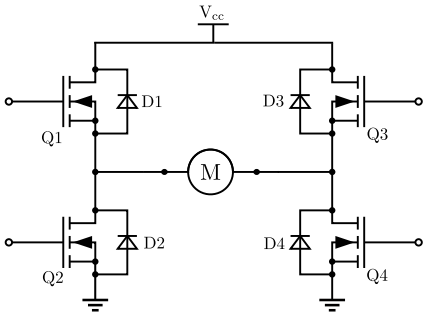


Figure 4.12: Standard H-bridge.
Edited from image by Biezl.[32]

There exists a wide range of configurations in which the H-bridge can be implemented. These configurations determines the modes of operation, where each mode has some different properties. Some of the common modes are described in the following sections.[31]

Regenerative Coast Mode

One mode of operation is the coast mode, where the PWM signal controls 2 of the 4 FETs, Q1 and Q3, as seen on *Figure 4.13*. In the on-period of the PWM signal the Q1 and Q4 are turned on, leading the current from Vcc through the motor and down to ground. This means that the motor is driven by the supply when the PWM goes high. When the PWM goes low, both Q1 and Q4 are shut off. Given that Q2 and Q3 are permanently off, the current generated by the motor in its armature coils will be driven through the intrinsic diodes of Q2 and Q3. So when the PWM goes low, the battery is charged.[33]

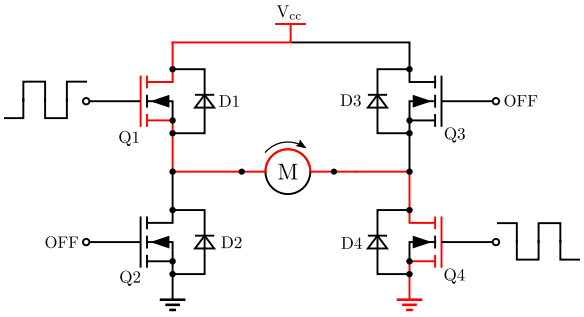


Figure 4.13: Clockwise coast in on-state.
Edited from image by Biezl.[32]

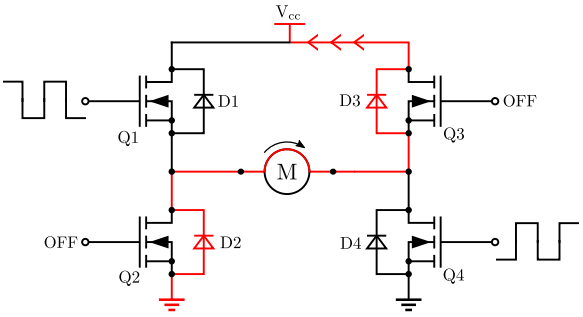


Figure 4.14: Clockwise coast in off-state.
Edited from image by Biezl.[32]

4Q Mode

In this mode of operation the PWM signal is imposed on all 4 FETs, as seen on *Figure 4.15* and *Figure 4.16*. On *Figure 4.15*, Q1 and Q4 are turned on by the PWM signal, allowing current to pass through the motor from Vcc to ground. Although the same PWM signal is imposed on the 4 FETs, only two can be turned on at the time, due to the NOT-gate placed on the gate of Q2 and Q3. Comparing *Figure 4.15* and *Figure 4.16*, it is seen that the polarity across the motor is switched between the up- and down-period of the PWM.[33]

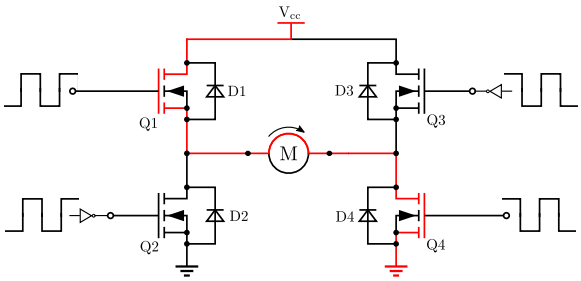


Figure 4.15: Clockwise 4Q operation.
Edited from image by Biezl.[32]

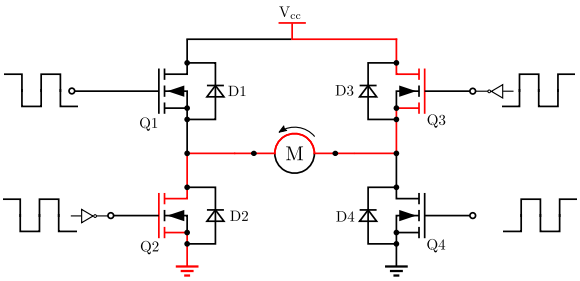


Figure 4.16: Counterclockwise 4Q operation.
Edited from image by Biezl.[32]

With this mode of operation a duty cycle of 50% makes the motor stand still, less than 50% will make it turn one way and more than 50%, the other way.

Brake Mode

This mode of operation imposes the PWM signal on one side of the H-bridge, as seen on *Figure 4.17* and *Figure 4.18*. The PWM signal in collaboration with the NOT-gate shifts between Q1 and Q2, which provides the two current paths as shown. On *Figure 4.17* the motor is driven by the supply, connected through Q1 and Q4. On *Figure 4.18* the motor is short-circuited to ground through Q2 and Q4, which brakes on the motor. In this example the configuration brakes to ground, however the principal is the same when braking to to Vcc.[33]

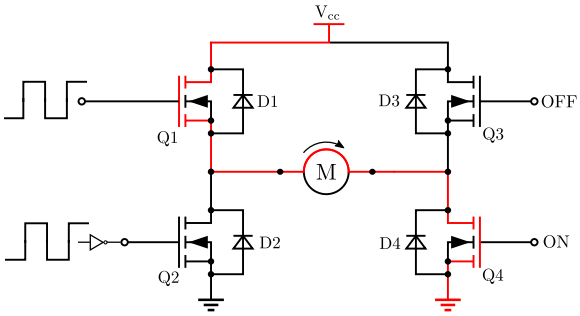


Figure 4.17: Clockwise brake in on-state.
Edited from image by Biezl.[32]

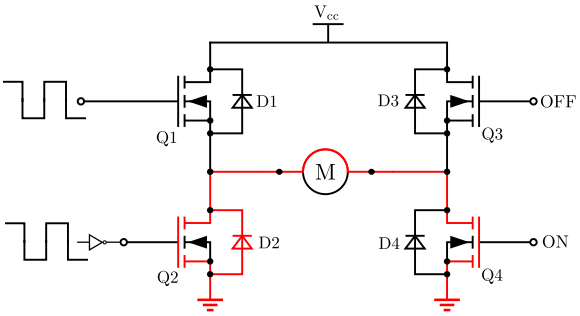


Figure 4.18: Clockwise brake in off-state.
Edited from image by Biezl.[32]

There multiple ways to configure an electrical break operation of an H-bridge, in this example the PWM signal is imposed on Q2, rather than only using the intrinsic diode, D2, when in OFF-state, *Figure 4.18* [31]. This method creates less heat dissipation in the FET, however weather it is necessary depends largely on the specifications of the FETs and the current draw of the load.

Double H-bridge Brake Mode

Implemented on the vehicle is a double H-bridge, which is configured to run in brake mode as illustrated on *Figure 4.19* and *Figure 4.20*.

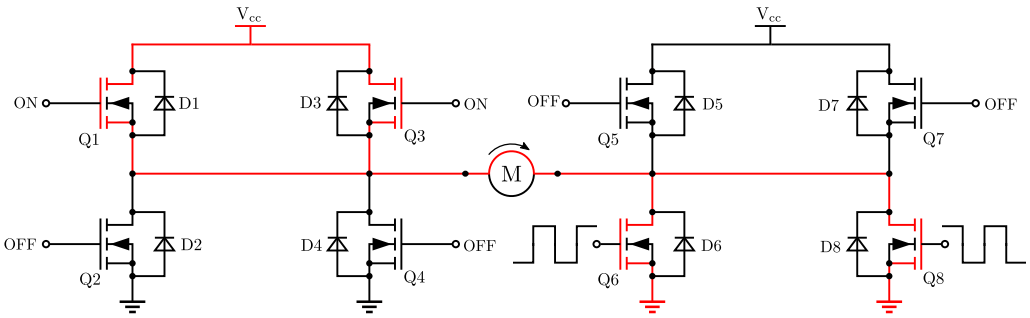


Figure 4.19: Double H-bridge brake operation in on-state.
Edited from image by Biezl.[32]

In this configuration the PWM signal is only imposed on Q6 and Q8. So instead of activating Q5 and Q7 in the OFF-period, as in *Figure 4.18*, it utilizes the intrinsic diodes of Q5 and Q7, as shown on *Figure 4.20*. This can sometimes cause too high heat dissipation in the intrinsic diodes, however this H-bridge has an absolute maximum rating of ±30 A, and in this configuration has two FETs in parallel, which is more than sufficient[17].

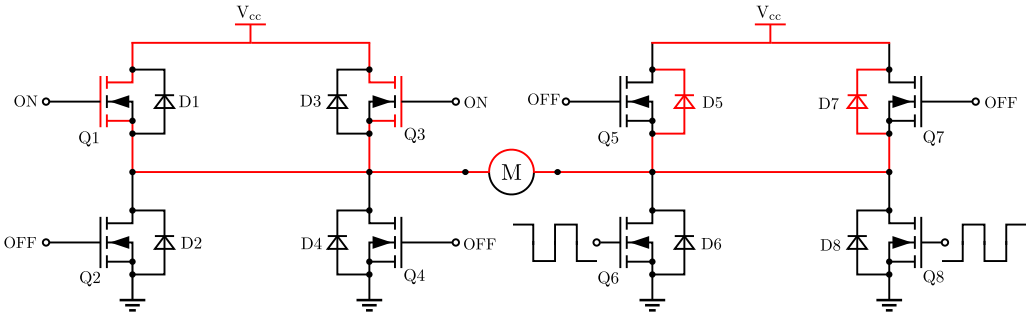


Figure 4.20: Double H-bridge brake operation in off-state.
Edited from image by Biezl.[32]

In brake operation the motor breaks in the OFF-period, as opposed to coast operation, where the motor runs freely in the OFF-period of the PWM signal. Though the coast mode is more energy efficient, it is harder to control the vehicle when trying to compensate for too high speeds. An H-bridge in coast operation can add more inertia to the system, however it is not very efficient when the system gains too much inertia, e.g. when going downhill. Break operation on the

other hand is less energy efficient, as it dissipates to break in the OFF-period of the PWM. This however allows the H-bridge to both increase and decrease speed, which allows for better control when for instance the vehicle is going downhill.

4.4 Software Choice

For the hardware components to interact, software parts are needed to handle them. The software choice is explained in this section, which describes the use of the chosen RTOS (Real Time Operating System), and the principle of scheduling.

Real Time Operating System

The chosen RTOS is a stable open source project named KRNL, written by the Associate Professor Jens Dalsgaard Nielsen from Aalborg University.
Specially written for the Arduino platform, it allows to control the tasks, and therefore the behaviour of the vehicle, through a timer, to keep a precise and constant time schedule. This property makes it possible to export it to another kind of processor and still have the same render, as long as it have a frequency high enough to process all the data needed.
One of KRNL's advantage is also the definition of semaphores, priorities, and critical regions, that the system will use to choose which task to run, and how often to run it. Futhermore, this RTOS allows to program in Arduino language, which is a C++ extension, and can also run C code, using different files.

Scheduling

To run the code on the Arduino microcontroller, and be able to manage all the sensors in parallel and at the time needed, a scheduling scheme is needed. Indeed, the microcontroller must be able to receive and process the data from the GoT computer received through the XBee module, the Hall sensors and the magnetometer. From all this data, it also has to make a decision regarding the planned route to follow. This decision must affect at the same time the speed of the motor, and the steering through the servo. A description of the scheduling principle and its function will now be described.

Semaphores

In programming, a semaphore is a variable or abstract data that is used to prioritize the different tasks to use. It allows a multiprogramming, with functions that use different resources and time, and does not interact with each others directly. Running them independently ease the development of the code.

Tasks

The different functions of the vehicle have been separated into multiple tasks, that can switch regarding to their priority. Five tasks are created to control the vehicle:

```
1
2   task1 = k_crt_task(tSpeed, 11, stack, 300);              // Hall Sensors
3   task2 = k_crt_task(SpeedControl, 12, stack2, 300);       // Speed control
4   task3 = k_crt_task(GoT,14,stack5,1000);                  // GoT and protocol handling
5   task4 = k_crt_task(LeadCompensator, 13, stack4, 300);    // Distance Control
6   task5 = k_crt_task(SteeringControl, 10, stack3, 1000);   // Angular control
```

Listing 4.1: Implementation of the tasks

Those declarations of the tasks are made thanks to the function k_crt_task needing the name of the function to run, it's priority, the stack buffer to use and it's length. The implementation of this function can be seen below.

```
1  struct k_t *k_crt_task (void (*pTask) (void), char prio, char *pStk, int stkSize);
```

Listing 4.2: Declaration of the tasks

Hall Sensors: The two hall sensors of the two belts are read at a certain frequency, and knowing the distance the vehicle moves during a full turn of the drive wheel, the real speed of each belt can be calculated independently.
When the belts are not running the maximum time to run the task has been measured to 5,63 μs, and when the belts are running it takes 8,39 μs. The maximum speed of vehicle is 3 $m \cdot s^{-1}$, and a turn of the gear wheel is 166mm. To get a pulse every full turn of the gear wheel at the highest speed, the sampling period has to be $\frac{0{,}166\,\mathrm{m} \cdot \mathrm{s}^{-1}}{3\,\mathrm{m}} = 55{,}3\,\mathrm{ms}$, and for a quarter of turn it should be 4 time less so every 14ms. According to the Nyquist-Shannon Sampling Theoem, the sampling frequency of reading the hall sensors has to be at least 2 times higher than the highest frequency of the signal, but to be safe, the sampling frequency is chosen 3,5 times higher, resulting in a sampling period of 4ms.

Speed Control: A wanted speed value is compared to the actual speed, and the resulting error is the input of the Velocity PI-Controller, that will set a new duty cycle for the motor, according to the reference.
The running time of this task is 386µs. As the speed can only be regulated when new measurements has arrived from the hall sensors, there will be no real advantages to run this task faster than at a 14ms period, in a perfect world. But as this is a real system, a little margin is added, and the period is chosen as 10ms.

GoT System and Communication Protocol: This task is the communication protocol handling, that receives positions of the vehicle from the GoT system. It will receive data from the computer, and convert it into a distance from the perfect route, so that the distance control can calculate the new heading to follow.
The running time of the GoT System and Communication Protocol task is 395µs, and the sampling period of the GoT system is 100ms.

Angular Control: The steering task gathers readings from the magnetometer, and transform them into the coordinate system that fits the model. Those values are then converted into a heading angle, that will be used in the steering P-Controller to calculate the new angle to send to the servo. This is the inner loop from the Steering Model, seen in *Section 7.2*.
The running time of the Angluar Control is the largest one with 1,7 ms needed to process a new angle. However, the sampling time of the servo is 30ms, so it makes no sense to run the task more often than this. The period of this task is therefore chosen to be 30ms.

Distance control: While the angular control is in charge of the direction of the vehicle, it can not control the parallel distance from the line wanted to be on. The task Distance Control calculate the distance of the vehicle from the line it should follow, and calculate a new heading for the angular control, to get back on the line. This is the outer loop from the Steering Model, seen in *Section 7.2*.
The running time of the Distance Control is 153,6 μs. It only runs when the GoT System and Communication Protocol delivers new data, so this task will also run once every 100ms.

A recapitulation of the parameters can be seen on *Table 4.2*.

n°	Task	Priority	Max. Time to Run	Min. Period	CPU Used
1	Hall Sensors	2	$8,39\mu$s	4ms	0,2%
2	Speed Control	3	386 μs	10ms	19,3 %
3	GoT and Protocol	4	395μs	100ms	0,395 %
4	Distance Control	4	$153,6\mu$s	100ms	0,154 %
5	Angular Control	1	1,7ms	30ms	5,7 %

Table 4.2: Calculations of the steering parameters.

In the *Table 4.2*, the Maximum Time to Run is the time the task needs to complete a full turn of the loop. The Minimum Period is the sampling time for the task, to run every time period. The CPU used is the percentage used by the task of the CPU, which is the ratio between the Maximum Time to Run and the Minimum Period.

Queue

The tasks are meant to be executed at the time they are scheduled to, ensuring the good control of the vehicle. But some functionalities are not critical, and can wait until the processor have time to do them.
When a task is running and another one with a higher priority makes a request to run, the actual task will be paused and wait until the higher priority ones are done.

Request and Priorities

The tasks makes a request of to run every period of time according to their respective setup, the global view of these request can be seen in *Figure 4.21*.

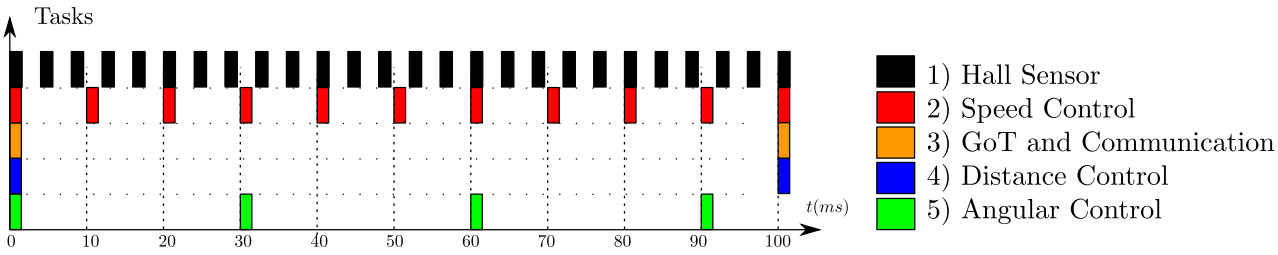


Figure 4.21: An overview of the tasks requests during 100ms.

When two task requests to run at the same time, the order is chosen according to the priorities, that are set at the initialisation. If the processor is free. the task are run when they ask for it, but interrupted when a higher priority task needs to run, and will finish when all the higher priority tasks are done. An example of this case can be seen in *Figure 4.22*.
The priorities of the task have been made to give the system the best stability. In this case, The angular sensor has the highest priority because it has a large period and running time, but is critical to the steering of the vehicle which is the main objective of this project. Then comes the hall sensors that is very fast to execute, and then the speed control which is so fast that is not critical to have it at a high priority. The last one is the GoT system and communication protocol along with the distance control, because of two things: first because it may enter in an infinite loop and block the system if no data is received, and second and most of all because the main point of this project is to be able to run without the GoT system if something happens to block the communication.

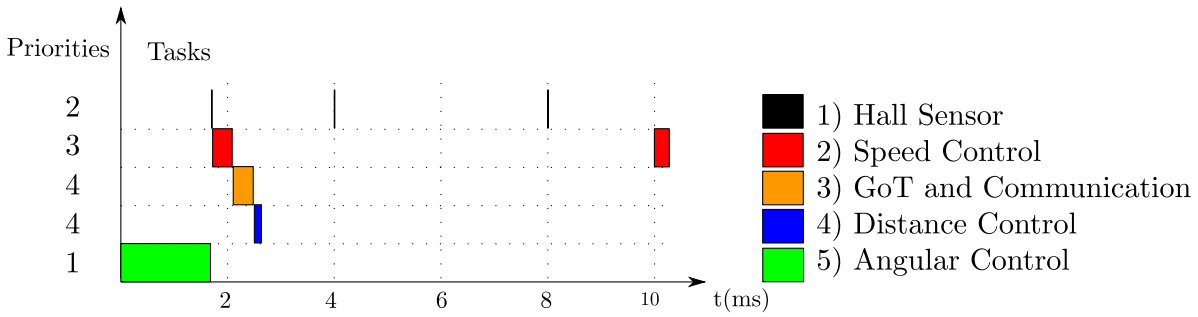


Figure 4.22: Schedule of the tasks at the starting time, regarding the priorities.

As seen on the *Figure 4.22*, the angular control has the highest priority so it will run first, and the others will be registrered in a queue in order of priority. Then the hall sensors are second, and speed control after that. The GoT and comunication will run after them, and in the end the Distance Control.
After a certain time, the task running periods will stabilize, until the tasks with a large period are executed again.

Now that the hardware and software choices have been made, described and explained, the tasks for the sensors can be implemented, independently from each other.

5 | Sensor Implementation

The different sensors used in this project needs software to work as expected in a consistent way and provide useful data.
The following sections describe how this is done, for the Hall sensors and the magnetometer.

5.1 Hall Sensors

The velocity calculation is made using data from the Hall sensors. There are four magnets on the gear with approximately a quarter of a turn of distance between each. When a magnet passes the hall sensor, a pulse is generated. One way of getting the speed is by calculating the time between two consecutive pulses and relating it to the distance travelled during that time, given by a quarter turn of the drive wheel. A plot of the speed calculated using this method can be seen on *Figure 5.1*.

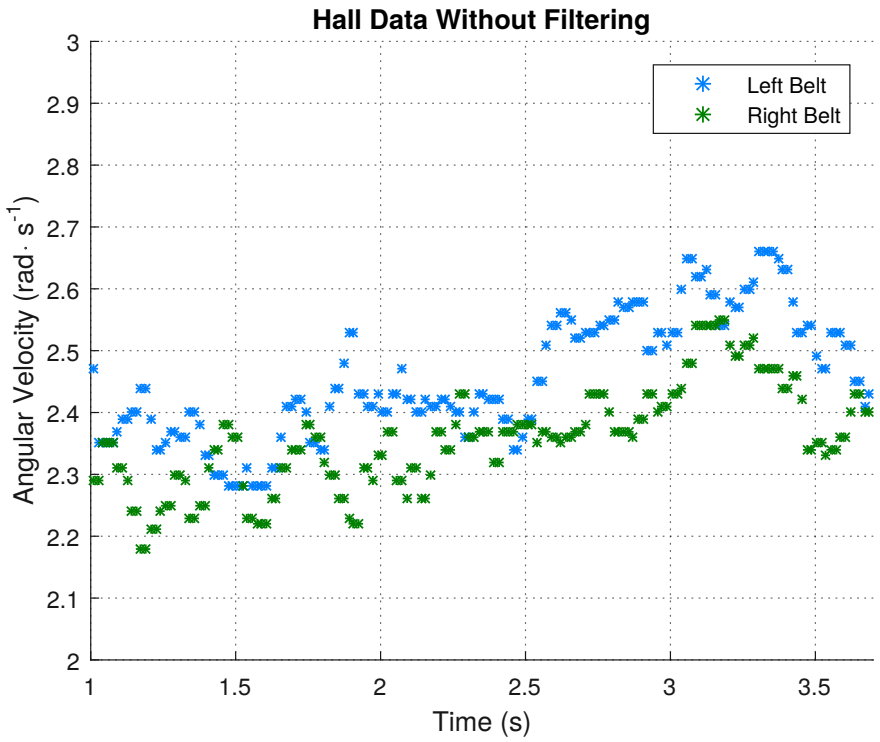


Figure 5.1: Plot of unfiltered measurements by the Hall sensors

As seen from the graph, this method yields inaccurate measurements due to uneven placement of the magnets. While it might be possible to improve the measurements using filtering, another approach will be tested first.

The new approach is to measure the time the wheel takes to make a full turn, to have the exact time and distance of a rotation. The speed will be calculated from a full turn of the drive wheel. Each magnet pulse will then be compared to its own last pass. This means that the first 4 pulses does not yield a speed, since the time of the first rotation happens between the 1^{st} and the 5^{th} pulse. However, the remaining measurements will have the same sampling frequency as the previous. A plot of the measurements using this method can be seen on *Figure 5.2*.

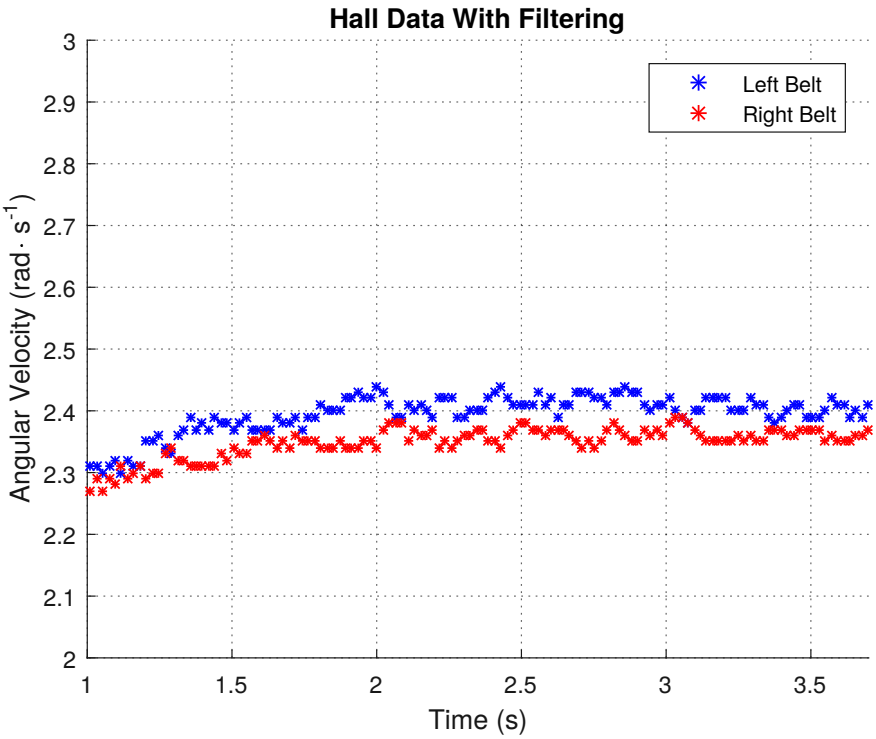


Figure 5.2: Plot of filtered measurement by the Hall sensors.

Comparing *Figure 5.1* and *Figure 5.2*, the noise has been significantly reduced. Because of this, filtering will not be necessary, and the new approach using a full turn of the wheel for each measurements is chosen.

Minimum Speed

The Hall sensors measurements can only capture speeds down to a certain value because of the size of the buffer which holds the timer value. The timers can count up to 65 535, and each clock tick lasts 64 µs.
A full rotation of the wheel yields a linear movement of 166 mm, so the minimum measurable speed is given by

$$\frac{166 \cdot 10^{-3}}{65535 \cdot 64 \cdot 10^{-6}} = 0,0396 \qquad \left[\mathrm{m \cdot s^{-1}}\right] \quad (5.1)$$

The Hall Sensors provides data with an acceptable noise level. A filter is therefore not needed for this sensor. The sensors related to the speed have now been explained, and the sensor related to the angle can be described.

5.2 Magnetometer

The "Sparkfun 9 degrees of freedom" sensor board [21] used in this project includes three sensors, of which one is the magnetometer chip HMC5883L [34]. Since the Earth's magnetic field only moves slightly, it is possibe to neglect this movement from the sensor's point of view. The magnetometer uses this property to calculate the three space components (x,y,z) of the Earth's magnetic field magnitude, therefore indicating the North pole's position relative to the sensor's own orientation.
The coordinates are sent through I^2C each time asked by the software onboard the Arduino. This means that it is possible to set the sampling time according to the needs of the steering controller part, within the sensor's sampling limits. By maintaining the sensor in a horizontal position and using trigonometry it is possible to get an angle equivalent to a compass heading, with the North as a reference of 0° :

$$\theta_{\text{heading}} = \frac{180}{\pi} \cdot \arctan\left(-\frac{\text{y}}{\text{x}}\right) \quad [^\circ] \quad (5.2)$$

Where:

θ_{heading}	is the sensor's heading	[°]
arctan	is the reciprocal of the tan function	[rad]
y	is the y-component of the measured magnitude	[G]
x	is the x-component of the measured magnitude	[G]

In *Equation (5.2)*, the angle calculated from the arctangent function is converted from radians to degrees to facilitate the work with angle orientations later in the code. However, to be able to use the heading values received from the sensor and this formula, it is necessary to first calibrate the sensor [**JJankowski**]. This allows the sensor to account for magnetic disturbances that could happen in its environment, see *Appendix I*. It might be possible though, that unpredictable disturbances occur during the mowing of the lawn. This is why a digital filtering approach is considered in *Chapter 9*.

The sensors placed on the vehicle is up and running, now the coordinate from the GoT system should be set up, so the vehicle can utilize them.

6 | Communication

A communication system is desired for transporting the GoT data, calculated to coordinates on the computer, to the microcontroller, located on the vehicle. Furthermore a protocol handling packet transitions is necessary to implement on top of the communication system to be able to fulfil requirements set in *Section 3*.

6.1 OSI model

The Open Systems Interconnection, OSI, model is a standard used to describe how information circulates in a network. Each layer describes a standardization of what should occur in the specific layer [35]. The seven layers of the OSI model are illustrated in *Table 6.1*.

Layer	**OSI**
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

Table 6.1: OSI model

Starting from the buttom, the Physical layer is the electrical/mechanical hardware layer. The layer handles bits and determines the communication medium, e.g. a copper cable or radio waves, and what is defined as a binary 0 and 1.
The prototype utilizes two XBee modules, see *Section 4.1*, one on the computer and one on the vehicle, for the lowest layers of communication. The XBee already implements this layer through the IEEE 802.15.4 protocol [20, 36]. It uses radio frequencies in the 2,4 GHz spectrum band.

The Data Link layer's functionality ensures physical addressing, error control when transferring data between nodes over the physical layer, and access control [37]. The Data Link layer is something the Xbee module also handles through IEEE 802.15.4, to check that it transmits bytes correctly and from and to which physical device it is transmitted [36].

The Network layer's main functionality allows for the routing across a network with many nodes. This is not utilized in the prototype's protocol stack, since the communication is made only between two Xbees, i.e. two nodes.

The Transport layers handles packets and has the ability to assemble or disassemble them. An important part of the functionality of this layer is to ensure reliable transmission of data packages through error control, retransmission, or possibly through connection-oriented communication. For the prototype, a connectionless protocol is used, see *Section 6.2*.

The Session layer's role is to establish connections between two nodes. It is not needed in the prototype since a connectionless communication protocol is used.

The Presentation layer should define how payload data is arranged for the Application layer to use it. These two layers are already implemented through another prototype's functionality, i.e. the Route Control, see *Section 2.4*, with the presentation being a simple set of two coordinates.

6.2 Protocol

The main object for the communication system is to transmit the coordinates from the computer to the microcontroller. A transport layer protocol is created to ensure packets can be transmitted, received, i.e. the needed information can be extracted from the packet, and be able to identify erroneous packets.

A connectionless system is implemented, since it is one-way communication, from the computer to the microcontroller and retransmission not desired, since the vehicle is moving and a retransmission of a coordinate will cause a delay. Additionally, the sampling rate of the GoT system is 10 Hz, so transmitting the recently received coordinates would be sufficient. Therefore, erroneous packages are thrown away and acknowledgements are not utilized.

Protocol Package Structure

A package structure is desired for containing necessary information for addressing, decrypting and error handling. The protocol has a header which contains a start byte, a destination and the length of the package. The body of the package is the data and the tail is the checksum and the end byte. The package structure is illustrated in *Table 6.3*.

Start Byte	Destination ID	length	Data	Checksum	End byte

Table 6.2: The structure of a package

Destination ID

The Destination ID is to ensure packages, which is transmitted, is handled by the correct receiver. This way the transmitter can transmit to numerous receivers if multiple vehicles is utilized. The destination ID ensures the receivers can filter out packages not intended for them. The microcontroller has the destination ID 0000 0001. The length of the destination byte is set to one byte, so the destination can be read immediately when received.

UDP, a connectionless protocol, utilizes a source, but in this case it is not necessary, since only one computer is transmitting, the receiver does not need to know where the package has been transmitted from. Even if more computers were utilized it would not be necessary as long as the package is marked with the destination ID, the receivers is able to read it. If retransmission was utilized, in case of erroneous packages, a source is necessary to have the ability to ask for a retransmission from the sender. Since retransmission is not utilized a source is not included in the package structure.

Length

The Length is utilized by the receiver. This will make it possible for the receiver to know the bit length of the package, and thereby know when the package should end. Each package created has the same length. The length is set to contain 7 bits, this can count up to $2^7 - 1 = 127$, which is sufficient.

Data

The data received from the GoT system consist of three coordinates (X,Y,Z), locating the position of the vehicle. Each of these coordinates can have a value from -9000 to 9000 (see *Section 2.2*). To contain a number of this size, a 15 bit signed integer is utilized. Of the 15 bit 14 bit is utilized for the magnitude of the coordinate, which can have a value of 16385 ($2^{14} - 1$). The last bit of the 15 bit is utilized to indicate if a coordinate is positive or negative, i.e. sign bit. The protocol is designed so the sign bit will be placed at the end of the 15 bit integer. As both the microcontroller and the computer utilizes little endians, there will not be a problem with the transmission of numbers. With little endians, the bit with the highest value will be located next to the sign bit and the bit with the lowest value is located at the beginning of the bit array. This will yield the bit array illustrated on *Table 6.3*.

15 bits

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	+/−

Table 6.3: Setup for the bit array, that contains one of the coordinate values.

The representation of the integer is called signed magnitude. Another representation that could be used is ones' complement. Here the number is inverted if the signed bit is true. But for an easier implementation on the computer, the signed magnitude is utilized. The code firsts checks if the number is positive or negative. If negative, the sign bit is set and the number is multiplied by minus one, this will yield the magnitude. The magnitude is then converted into a bit array. When converting back, it is only necessary to multiply the magnitude by minus one, if the sign bit is true.

Checksum

To ensure package data is received correctly, error handling is added to the protocol. A error handling which is utilized is a checksum. The way the checksum is calculated, is by splitting

the header and the data up into pieces of equal size and then summing the pieces together. The summed bit array is then inverted and the remaining bit array is the checksum.

there is 60 bit in the header and the data combined, i.e. 15 per coordinate, 8 for the destination and 7 for the length. these will be split up into three pieces of 20 bit. The related checksum consist of 20 bit. Combined, the header, data and checksum is 80 bit, which is equal to 10 bytes. In *Table 6.4* an example of calculating a checksum is given.

	Bit array		Decimal		Bit 0-3	Bit 4-7	Bit 8-11	Bit 12-15	Bit 16-19	Bit 20
a)	Part 1	=	123008	=	0000	0001	0000	0111	1000	
b)	Part 2	=	351365	=	1010	0001	0011	1010	1010	
c)	Part 3	=	729671	=	1110	0010	0100	0100	1101	
d)	Part 1+2+3	=	1204044	=	0011	0010	1111	1010	0100	1
e)	Add carry	=	155469	=	1011	0010	1111	1010	0100	
f)	Checksum	=	893106	=	0100	1101	0000	0101	1011	
g)	Check	=	1048575	=	1111	1111	1111	1111	1111	

Table 6.4: The pieces consisting of the destination 1, length with the value of 96, and the data which is 4259, 7511 and -6418.

The three pieces is added together, the destination, length and the data (see on line d). This produces a carry, since the checksum has the size of 20 bit. The carry is added to the beginning of the sum (line e). The generated sum is then inverted to yield a checksum (sees on line f).

When the receiver utilizes the checksum, it needs to check if the destination, length and data corresponds with the related checksum. The checksum is first added together with the three 20 bit pieces and, if there is any, the carry. If the outcome of the bit summation results in 20 bits which are true, then the package is received correctly. If not, the receiver will throw away the package. An error in the received package could be a bit inversion or some data not received by the receiver.

A flaw when utilizing the created checksum, is that an error can cancelled out another error, and thereby make some errors undetectable. This can happen when the system adds the three pieces and the checksum together. An example, is if the first bit in piece number one is true and the first bit in piece number two is false. If both of these bits are shifted independently, so the bit from part one become false and the bit in part two true, the checksum can not detect the error. The likelihood for this to happen is very slim, as the errors first have to occur and then they have to be located a place where they cancel each other out. Having only 4 pieces of 20 bits, the likelihood of this happening is lesser than if the checksum was set to 10 bit. Since the addition occurs with 7 pieces of 10 bit. This would yield more bits with the same rank, so there would be a higher possibility for a bit cancelling each other out. This is a good argument for utilizing a larger checksum, even if it uses more space.

Start and end byte

As the computer transmits a package each time it receives coordinates from the GoT system, a queue of packages can appear in the microcontroller, if package handling is too slow. To not make the system able to differentiate packages from each other a start byte is added at the beginning of the package and an end byte at the end of the package.

To find the start of a package the system will search for a start byte. When finding the start byte, it will read the destination, length, data and checksum and then look for a end byte. If the end byte is not found at that point, then the package is not received correctly and is thrown away. If the end byte is found, then system utilize the package and apply error handling on it.

The start byte is set to 0000 1111. This is chosen to ensure the start byte does not have an identical value with another byte in the header. This is done to ensure no misunderstandings occurs. The end byte will come just before the next start byte. The value of the end byte is inverted compared to the start byte, this yields an end byte of 1111 0000.

The destination, length, data, checksum, start and end byte has been described, and a final package structure is illustrated in *Table 6.5*

Offsets		Byte 1								Byte 2							
Byte	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	Start byte								Destination							
2	16	Length								X coordinate							
4	32	X coordinate								Y coordinate							
6	48	Y coordinate								Z coordinate							
8	64	Z coordinate								Checksum							
10	80	Checksum								End byte							

Table 6.5: Illustration of a package, that will be send from the transmitter to the receiver.

This yields a package with a length of 96 bit (12 byte). As the GoT system have a sampling frequency of 10 Hertz and each sample is converted to 12 bytes, the transfer speed for the Xbee have to be greater than 120 bytes per second, which it is *Section 4.1*.

Further Error Handling

A package can be damaged when it is transmitted from the computer to the microcontroller or a byte in the coordinate could have a identical value with the start or end byte. for example, if a packet has a byte of the coordinate with the same value as the start byte, the start of a package could be registered in the middle of a package. To detect this and further damages in which the checksum is not able to register, extra error handling is utilized. The receiving process is illustrated in *Figure 6.1*.

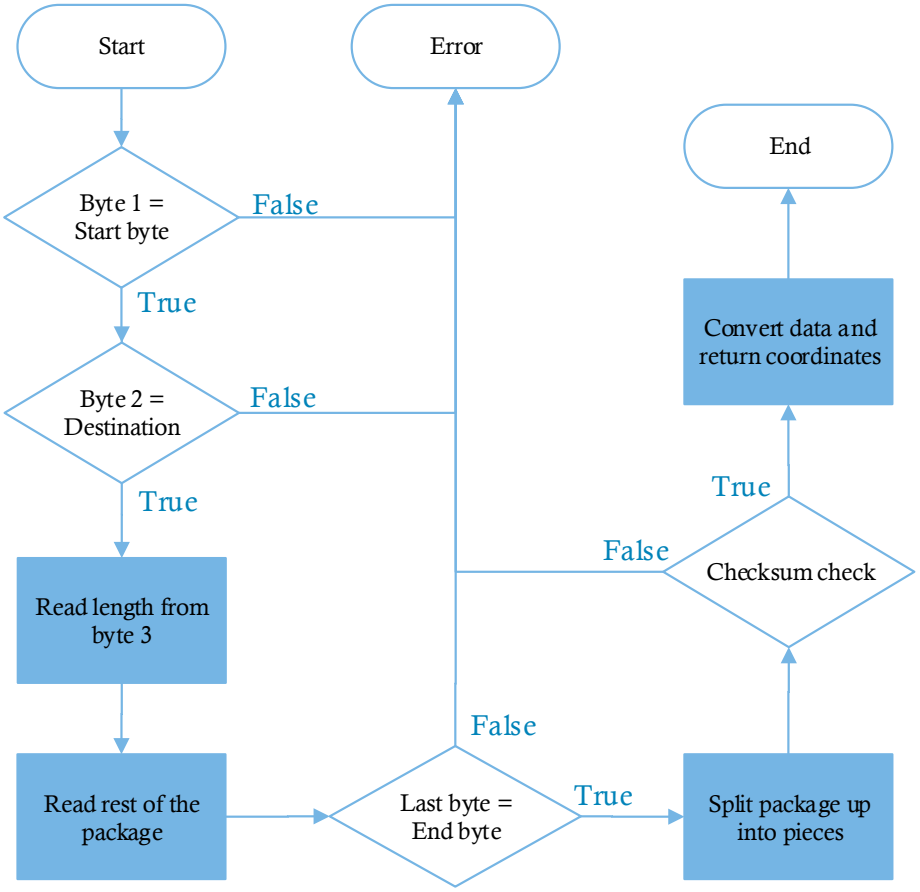


Figure 6.1: Flow chart over the error handling in the receiver part.

If the package is transmitted correctly, the 1st byte received is the start byte and the 2nd byte is the destination ID of the receiver. The 3rd byte will contain 7 bit for the length and 1 bit for the first coordinate. If the length is without errors its value would be 96, which is 12 bytes. The last byte, the 12th one, is the end byte. If these condition are all correct, the receiver will check for errors utilizing the checksum. If that checksum is also correct, the system will convert the data and return the coordinates to the rest of the system.

If an error occurs, then all the bytes that have been read will be thrown away. Thus throwing the whole package away even if the start byte and the destination are correct, but the end byte is not. If the start byte is not correct, then only that byte will be thrown away until finding a correct start byte. This method is applied to ensure the system only utilizes complete packages.

This prevents the receiver to start in the middle of a package, as the system will throw away incomplete or damaged packages, until it finds a start byte. One example when utilizing this feature can cause a problem. If a start byte has a identical value with a byte in data or checksum.

The problem will occur if the second byte is equal to the receivers destination ID, and the first seven bits of the next byte are equal to 96, which is the value of the length. An example on a situation where this problem occurs is illustrated in *Table 6.6*.

	Normal reading									Displaced reading								
1st byte	0	0	0	0	1	1	1	1	5th byte	0	0	0	0	1	1	1	1	← Start byte
2nd byte	0	0	0	0	0	0	0	1	6th byte	0	0	0	0	0	0	0	1	← Destination
3rd byte	0	0	0	0	0	1	1	0	7th byte	0	0	0	0	0	1	1	0	← Length (First seven bit)
4th byte	1	1	1	1	0	0	0	0	8th byte	0	1	1	1	0	0	1	1	
5th byte	0	0	0	0	1	1	1	1	9th byte	1	0	0	1	1	0	1	1	
6th byte	0	0	0	0	0	0	0	1	10th byte	1	0	0	0	0	1	0	0	
7th byte	0	0	0	0	0	1	1	0	11th byte	0	0	1	1	0	1	1	1	
8th byte	0	1	1	1	0	0	1	1	12th byte	1	1	1	1	0	0	0	0	
9th byte	1	0	0	1	1	0	1	1	1st byte	0	0	0	0	1	1	1	1	
10th byte	1	0	0	0	0	1	0	0	2nd byte	0	0	0	0	0	0	0	1	
11th byte	0	0	1	1	0	1	1	1	3rd byte	0	0	0	0	0	1	1	0	
12th byte	1	1	1	1	0	0	0	0	4th byte	1	1	1	1	0	0	0	0	← End byte

Table 6.6: Example of a normal reading of a package and a displaced reading of a package. The package contains the coordinates (-8222, 515, -3699).

In the example illustrated in *Table 6.6*, the 5th to 7th byte are equal to the start byte and the header. If the system is searching for the start byte and finds the 5th byte, it will begin to search for the header. In this case it will find it and as the 12th byte, which is in the next package, is equal to the end byte, the system will confirm it as a complete package. But since a checksum is implemented, it will be unlikely if the package is confirmed. The checksum, in this case, will not be the original checksum for the package, but another part of the package. As this checksum is not calculated to fit, the likelihood for the check going through is so poor, that error handling detecting this is not implemented.

If a erroneous package is being stopped by the checksum, the 12 bytes would still have been read and is therefore thrown away. In the 12 bytes thrown away a correct start byte can be located. A problem can occur if the next three bytes, after the 12 bytes which is thrown away, is equal to the start byte and header again. Then the same scenario will repeat it self and will continue, until the three byte no longer equals the start byte and header. The scenario is unlikely, as three bytes has to be identical to the start byte and the header.

The transport layer protocol implemented on top of the Xbee modules has now been described.

6.3 Communication Filtering

The GoT system yields the vehicle's position throughout time. However, since this system utilizes ultrasound waves to register positions in space, it can be distorted by interferences and objects on the wave path. It is therefore required to eliminate the big jumps in positions, that could occur in the GoT system, before sending them to the vehicle, see requirements in *Chapter 3*.

Since the vehicle cannot run faster than $3,0\ \mathrm{m \cdot s^{-1}}$, it is possible to define a certain range of positions for a defined time interval. The range is defined as a circle, with the a radius as large as the assumed maximum velocity, set to $3,0\ \mathrm{m\,s^{-1}}$, multiplied with the measured time interval. An example with a 0,1 s time interval is given on *Figure 6.2*.

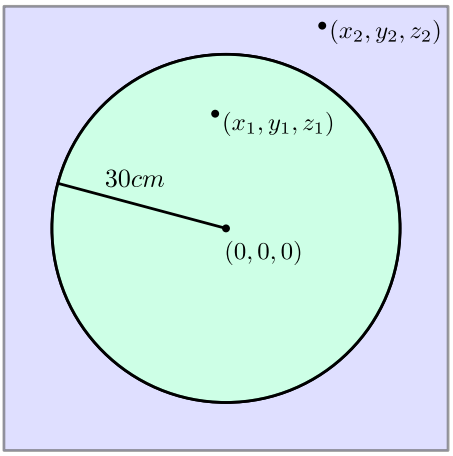


Figure 6.2: Range of possible positions for a maximum speed of $3,0\ \mathrm{m \cdot s^{-1}}$ during a 0,1 second interval

If the position at a given time, t_0 is $(0, 0, 0)$, the coordinates at the next measurement time, $t_1 = t_0 + 0{,}1\,\mathrm{s}$, should stay within a 30 cm radius of the first position.

To ensure the fact that no position jumps is sent to the vehicle, its velocity is measured from the two last sets of coordinates and the sampling time of the GoT system (100 ms, see *Section 2.2*), see *Equation (6.1)*.

$$v = \frac{\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}}{\Delta T} \qquad \left[\mathrm{m \cdot s^{-1}}\right] \quad (6.1)$$

Where:

v	is the velocity of the vehicle	$[\mathrm{m \cdot s^{-1}}]$
(X_2, Y_2, Z_2)	is the the last set of measured coordinates	[m]
(X_1, Y_1, Z_1)	is the the second last set of measured coordinates	[m]
ΔT	is the sampling time	[s]

Whenever a calculated velocity is greater than the limit of $3,0\ \mathrm{m \cdot s^{-1}}$, the newest set of coordinates is completely discarded. To calculate the newest velocity afterwards, the previous position,

that was not discarded, and the newly measured position are taken, and the sampling time is extended by 100 ms, until a velocity under the limit is measured. When the velocity is under the limit again, the time interval is set to the default value again.

However, if the GoT is correctly calibrated and the vehicle's environment is clear of interfering obstacles, the distributions will occur rarely.

The communication protocol has been designed to be able to disregard incorrect packages transmitted from the computer. Furthermore a filter has been implemented to ensure large distortion occurring in the GoT system, can be handled and filtered. This should fulfill the requirements set to the communication in *Section 3*.

7 | Modeling of the Vehicle

The prototype model contains software, hardware and mechanical components. The vehicle, which is a mechanical plant, is provided for this project and therefore not changeable. This means that requirements about how the system should react, described in *Section 2.2*, apply only for elements of the system that are external to the plant. In this chapter, a model of the vehicle is made to describe the power transfer from both the motor's and servomotor's rotational energy to the movement of the vehicle. Once, this model is established, it is possible to measure its output and apply a controller, see *Chapter 8*.

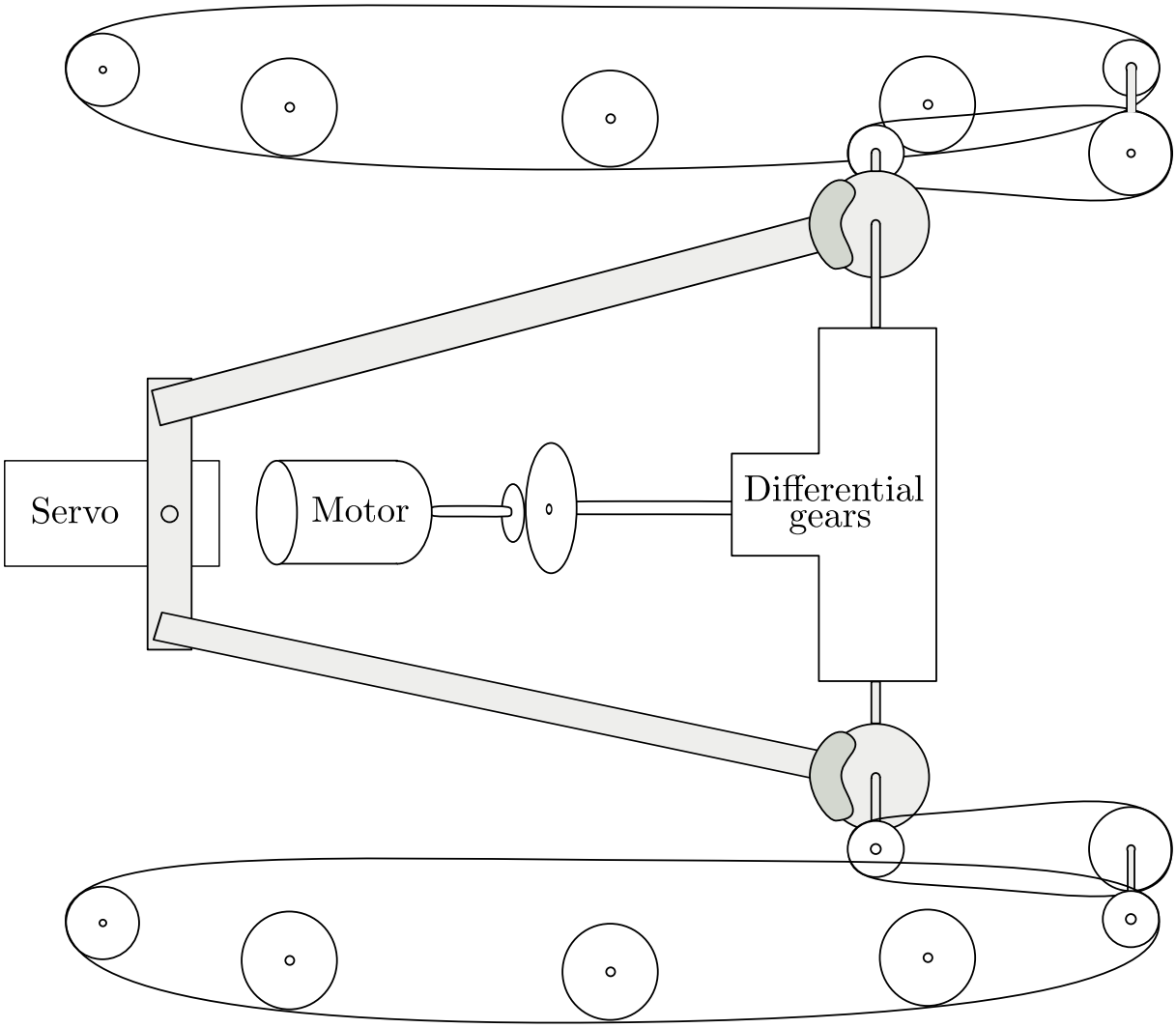


Figure 7.1: A mechanical diagram of the vehicle

The *Figure 7.1* illustrates both the drivetrain and steering mechanisms on the given vehicle. It is possible to identify and separate these two sub-systems.
The parts that allow the vehicle to run includes the motor, the gears (that make the belts turn) and the belts themselves. The steering part includes the servomotor, which acts on two shafts

and thereby capable of breaking one of the other belts, as stated in *Section 2.2*. In the following sections, the motor and the drivetrain, which allow the vehicle to have a certain velocity, and the steering, which allows the vehicle to turn, are modelled independently.

7.1 Velocity Model of the Vehicle

In this section the driving parts of the vehicle in *Figure 7.1* is described and modelled. The following subsections will therefore focus on getting the relationship between the input voltage applied to the motor and the system's output velocity. The schematic in *Chapter 7*, *Figure 7.1* is reduced to the following *Figure 7.2*.

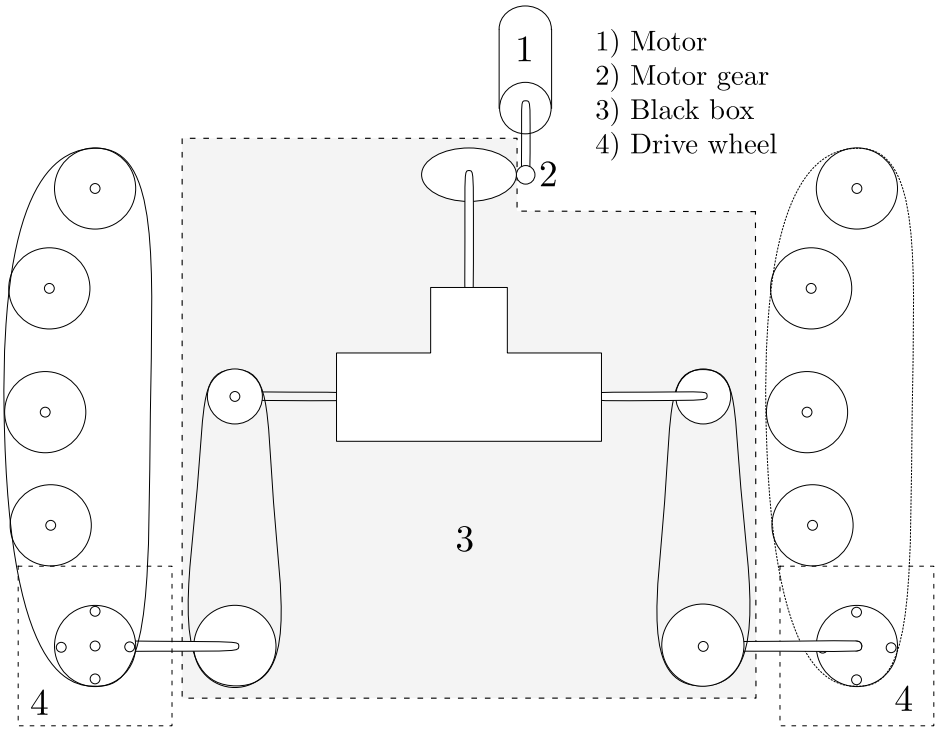


Figure 7.2: A mechanical diagram of the driving part of the system

The system on *Figure 7.2* is expanded into four different parts to facilitate the modeling process. The first part is the motor itself which is the actuator and is controlled by an input voltage. The mechanical work, produced in the form of a torque, is translated onto the motor shaft and to the motor gear, which is the second part. The third part, the black box seen *Figure 7.2*, consists of the largest part of the drivetrain, from and excluding the motor gear to the gear that sits on the same shaft as the drive wheel. The black box, will be named drivetrain gear hereafter. The fourth part of *Figure 7.2* is describing the conversion from the rotational movement of the drive wheel to the vehicle's belts, and with the end result of making the vehicle move.

The velocity model of the vehicle is built in two main steps in the following subsections. The first step describes the motor's electrical behaviour in response to an input voltage. In a second step, a model of the vehicle's velocity is made from the parts 2 to 4, see *Figure 7.2*, in response the motor's torque. These two subsections are derived by only considering the parameters affecting

the vehicle when it is driving in a straight line. Thus, the the two belts are supposed to have the exact same velocity, in this first model.

In the following subsection the motor behaviour is modelled.

Motor Modelling

In this section a model of a motor is depicted only as an electrical system, which delivers a rotational force called torque, τ_{m}. The electrical model provides the motor's produced torque to the drivetrain which is further discussed in *Section 7.1*.

Electrical Model

The output needed from the motor's electrical model is the torque, τ_{m}. To obtain the torque, the formula for translating the electrical current, $\mathrm{i_a}$, to torque is utilized:

$$\tau_{\mathrm{m}}(\mathrm{t}) = \mathrm{K_t} \cdot \mathrm{i_a}(\mathrm{t}) \qquad [\mathrm{N \cdot m}] \quad (7.1)$$

Where:

$\tau_{\mathrm{m}}(\mathrm{t})$	is the rotational force torque	$[\mathrm{N \cdot m}]$
$\mathrm{i_a}(\mathrm{t})$	is the electrical closed loop current	[A]
$\mathrm{K_t}$	is the motor constant	$[\mathrm{N \cdot m \cdot A^{-1}}]$

An expression for the current, $\mathrm{i_a}(\mathrm{t})$, is required to derive a model for the electrical system. In *Figure 7.3* an electrical diagram of the motor is displayed.

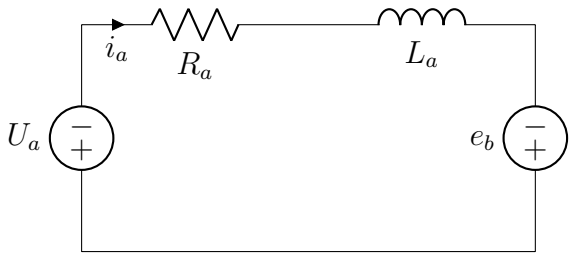


Figure 7.3: A electrical diagram of the motor

By using Kirchoff voltage law on the closed loop, seen in *Figure 7.3*, an expression including i_a can be derived:

$$\mathrm{U_a}(\mathrm{t}) = \mathrm{R_a} \cdot \mathrm{i_a}(\mathrm{t}) + \mathrm{L_a} \cdot \frac{\mathrm{di_a}(\mathrm{t})}{\mathrm{dt}} + \mathrm{e_b}(\mathrm{t}) \qquad [\mathrm{V}] \quad (7.2)$$

Where:

$\mathrm{U_a}(\mathrm{t})$	is the supply voltage	[V]
$\mathrm{R_a}$	is the internal resistance in the motor	[Ω]
$\mathrm{L_a}$	is the inductance in the motor	[H]
$\mathrm{e_b}$	is the electromotive force, also called EMF	[V]

The electromotive force, $e_b(t)$, is equivalent to:

$$e_b(t) = K_e \cdot \dot{\theta}_m(t) \quad [V] \tag{7.3}$$

Where:

K_e	is the electromotive constant	[Wb]
$\dot{\theta}_m(t)$	is the angular velocity in the motor	$[rad \cdot s^{-1}]$

The equivalent for the electromotive force is substituted into *Equation (7.2)*.

$$U_a(t) = R_a \cdot i_a(t) + L_a \cdot \frac{di_a}{dt} + K_e \cdot \dot{\theta}_m(t) \tag{7.4}$$

The Laplace transform is applied to the derived equation:

$$U_a(s) = R_a \cdot i_a(s) + L_a \cdot s \cdot i_a(s) + K_e \cdot \omega_m(s) \tag{7.5}$$

The equation is solved for i_a:

$$i_a(s) = \frac{U_a(s) - K_e \cdot \omega_m(s)}{L_a \cdot s + R_a} \tag{7.6}$$

By substituting the derived equation for i_a into *Equation (7.1)*, a new expression for the motor's torque is derived.

$$\begin{aligned} \tau_m &= K_t \cdot i_a(s) \\ \tau_m &= K_t \cdot \frac{U_a(s) - K_e \cdot \omega_m(s)}{L_a \cdot s + R_a} \end{aligned} \tag{7.7}$$

By dividing with the voltage applied to the motor, U_a, a relation can be established:

$$\frac{\tau_m}{U_a(s) - K_e \cdot \omega_m(s)} = \frac{K_t}{L_a \cdot s + R_a} \tag{7.8}$$

A relation for the electrical model relative to the motor's torque has been derived. Thus enabling setting up calculations for the vehicle's drivetrain.

Drivetrain

The drivetrain translates the torque $\tau_m(t)$, given by the motor, into the actual movement of the vehicle. A mechanical depiction of the parts considered in this modeling section can be seen on *Figure 7.4*. Here the input torque to the system, given by the motor shaft, translates through the system ending up as a linear velocity of the vehicles mass on the belts, i.e. the velocity of the vehicle. The velocity model of the drivetrain is created by only considering when the vehicle's trajectory follows a straight line.

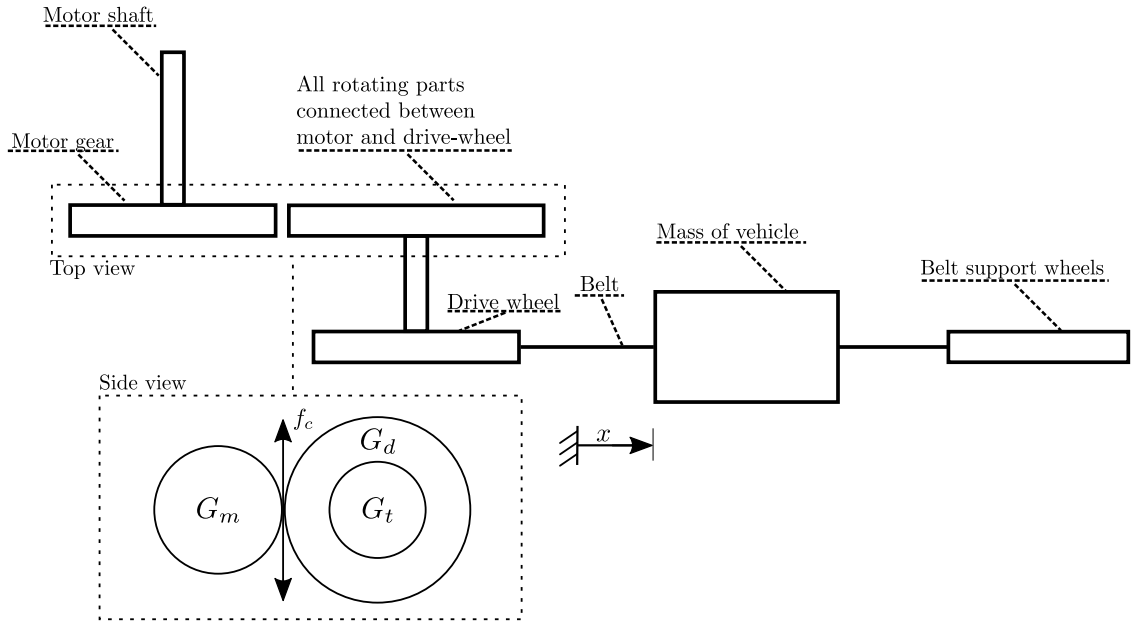


Figure 7.4: A diagram showing the outlines of the mechanical model of the drivetrain

To make it more manageable the modeling of the drivetrain is limited to only modeling parts of the total drivetrain system.

Model of the Internal Gears

To get a rough approximation of the effect of the drivetrain, a black boxed model is used. The black box in the model is placed between box 1 and including box 3 seen in *Figure 2.4*. The gear, G_m, connected to the motor shaft, is connected to the gear, G_d, which represents the gears and shafts that has been black boxed. The output of black box is given on the shaft connected to the drive-wheel, G_t, see *Figure 7.4*. The number of teeth on the gear, G_d, represents the gear ratios throughout the drivetrain, such that the gear ratio between G_m and the output at the drive gear is given by: $\frac{N_m}{N_d}$, where N_m is the number of teeth on the motor gear and N_d is the calculated number of teeth on the black box gear.

The torque of the motor along with the contributions from the load, i.e. the drivetrain, affecting the motor shaft, is depicted as a free body diagram in *Figure 7.5*.

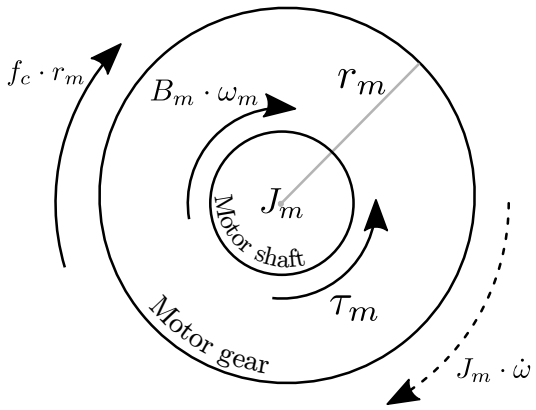


Figure 7.5: A free body diagram of the motor gear, G_m

The radius of the motor gear times the contact force, $r_m \cdot f_c$, is the torque translated from the motor gear, G_m, to the gear representing the drivetrain, G_d, in *Figure 7.6*. Thus it appears as an opposing torque to the applied motor torque, τ_m. From *Figure 7.5*, the following equation can be derived:

$$\mathrm{J_m} \cdot \dot{\omega}_\mathrm{m}(\mathrm{t}) = \tau_\mathrm{m}(\mathrm{t}) - \mathrm{B_m} \cdot \omega_\mathrm{m}(\mathrm{t}) - \mathrm{r_m} \cdot \mathrm{f_c}(\mathrm{t}) \qquad [\mathrm{N} \cdot \mathrm{m}]$$

Where:

J_m	is the motor's inertia	$[\mathrm{kg} \cdot \mathrm{m}^2]$
$\omega_m(t)$	is the angular velocity of the motor	$[\mathrm{rad} \cdot \mathrm{s}^{-1}]$
$\dot{\omega}_m(t)$	is the angular acceleration of the motor	$[\mathrm{rad} \cdot \mathrm{s}^{-2}]$
$\tau_m(t)$	is the torque delivered by the motor	$[\mathrm{N} \cdot \mathrm{m}]$
B_m	is the motor's friction coefficient	$[\mathrm{N} \cdot \mathrm{m} \cdot \mathrm{s} \cdot \mathrm{rad}^{-1}]$
r_m	is the radius of the gear, G_m, connected to the motor shaft	[m]
$f_c(t)$	is the contact force between the two gears	[N]

To find the output of the drivetrain, a free body diagram is is depicted and illustrated in *Figure 7.6*. The figure represents the system's effecting torques on the blackbox gear, G_d, and the translation of these to the drive wheel.

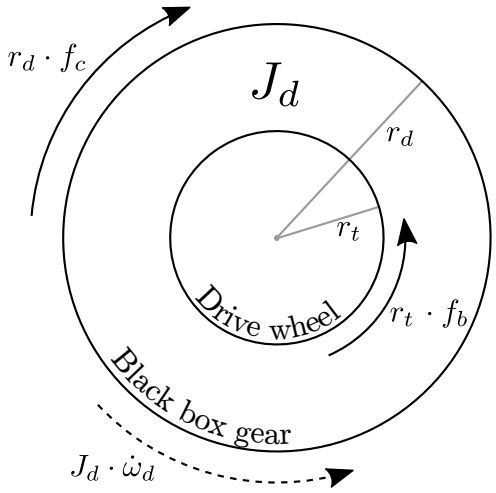


Figure 7.6: A free body diagram of the 'black box' gear, G_d

An equation is extracted from *Figure 7.6*:

$$\mathrm{J_d} \cdot \dot{\omega}_\mathrm{d}(\mathrm{t}) = \mathrm{r_d} \cdot \mathrm{f_c}(\mathrm{t}) - \mathrm{r_t} \cdot \mathrm{f_b}(\mathrm{t}) \quad [\mathrm{N \cdot m}] \quad (7.9)$$

Where:

J_d	is the 'black box' gear inertia	$[\mathrm{kg \cdot m^2}]$
$\dot{\omega}_d(t)$	is the angular acceleration of the 'black box' gear, G_d	$[\mathrm{rad \cdot s^{-2}}]$
r_d	is the radius of the 'black box' gear, G_d	[m]
r_t	is the radius of the drive wheel, G_t (translational)	[m]
$f_b(t)$	is the coefficient of the contact force between G_d and the belt	[N]
$f_c(t)$	is the contact force between the two gears	[N]

Two equations for the internal gears have been found, *Equation (7.1)* and *Equation (7.9)*. In the following paragraph a model of the translation between the drive-gear and the belt is created.

Translational Model

It is necessary to find the translation between the drive-gear and the belt to be able to describe the movement of the vehicle. Furthermore, it is needed to be able to make a full model of the system, going from the gear connected to the motor shaft to the vehicle's belts. In *Figure 7.7* a mechanical diagram illustrates how the force, given through the drive wheel, is translated by the belts. The translation yields a linear force acting on the mass, which makes the vehicle move with a corresponding velocity.

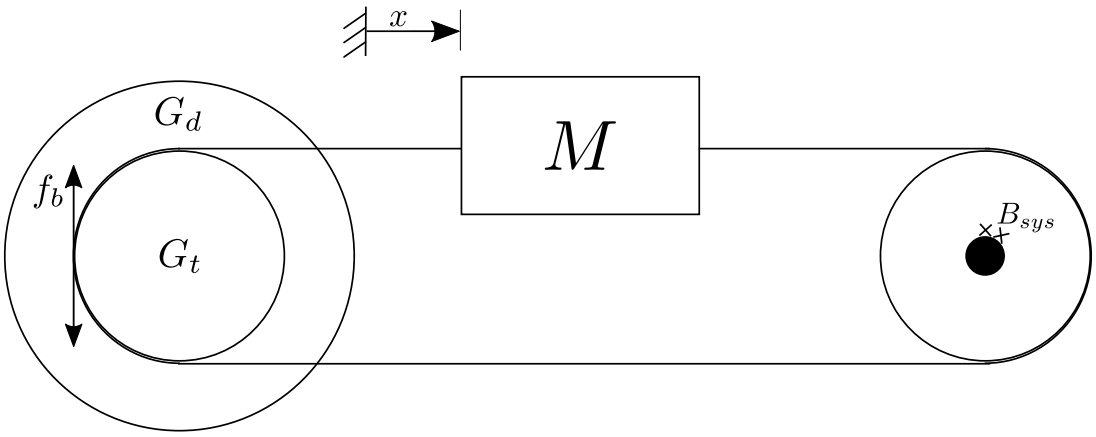


Figure 7.7: A mechanical diagram of the drive-gear rotating the belt. The mass, M, pressing down on the vehicle's wheels.

By using *Figure 7.7* a free body diagram of the mass, M, and the forces applied to the mass is derived.

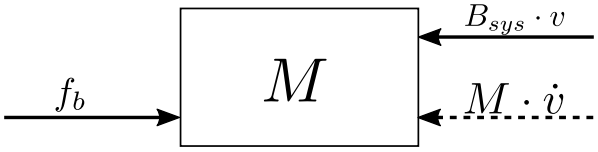


Figure 7.8: A free body diagram illustrating the vehicle's mass affected by the surrounding forces

From *Figure 7.8*, a mechanical equation of the vehicle's resulting force is found to be:

$$\mathrm{M} \cdot \dot{\mathrm{v}}(\mathrm{t}) = \mathrm{f_b}(\mathrm{t}) - \mathrm{B_{sys}} \cdot \mathrm{v}(\mathrm{t}) \quad [\mathrm{N}] \quad (7.10)$$

Where:

M	is the vehicle's total weight	[kg]
$v(t)$	is the linear velocity of the vehicle	$[\mathrm{m} \cdot \mathrm{s}^{-1}]$
$\dot{v}(t)$	is the linear acceleration of the vehicle	$[\mathrm{m} \cdot \mathrm{s}^{-2}]$
B_{sys}	is the coefficient of the friction throughout the rotational parts	$[\mathrm{N} \cdot \mathrm{s} \cdot \mathrm{m}^{-1}]$

In the following section the three mechanical equations, *Equation (7.1)*, *Equation (7.9)* and *Equation (7.10)* is combined.

Model of the Combined Drivetrain

In this section a model will be created of the combined drivetrain, by assembling the three derived mechanical equations, *Equation (7.1)*, *Equation (7.9)* and *Equation (7.10)*. This allows to get a linear velocity of the vehicle, V(s), from the torque, τ_m, delivered by the motor. The three equations are linked through the contact forces $\mathrm{f_c}$ and $\mathrm{f_b}$.

The Laplace-transform is applied on the three equations in order to deduce a transfer function of the input torque, $\tau_m(s)$, to the output velocity, V(s):

$$J_m \cdot \omega_m(s) \cdot s = \tau_m(s) - B_m \cdot \omega_m(s) - r_m \cdot F_c(s) \tag{7.11}$$

$$J_d \cdot \omega_d(s) \cdot s = r_d \cdot F_c(s) - r_t \cdot F_b(s) \tag{7.12}$$

$$M \cdot V(s) \cdot s = F_b(s) - B_{sys} \cdot V(s) \tag{7.13}$$

$F_c(s)$ is isolated from *Equation (7.11)*:

$$F_c(s) = \frac{\tau_m(s) - B_m \cdot \omega_m(s) - J_m \cdot \omega_m(s) \cdot s}{r_m} \tag{7.14}$$

$F_b(s)$ is isolated from *Equation (7.12)*:

$$F_b(s) = \frac{r_d \cdot F_c(s) - J_d \cdot \omega_d(s) \cdot s}{r_t} \tag{7.15}$$

$F_b(s)$ is isolated from *Equation (7.13)*.

$$F_b(s) = V(s) \cdot (M \cdot s - B_{sys}) \tag{7.16}$$

The $F_c(s)$ in *Equation (7.15)* is substituted for the expression given by *Equation (7.14)*:

$$F_b(s) = \frac{r_d \cdot (\frac{\tau_m(s) - B_m \cdot \omega_m(s) - J_m \cdot \omega_m(s) \cdot s}{r_m}) - J_d \cdot \omega_d(s) \cdot s}{r_t}$$

$$F_b(s) = \frac{\tau_m(s) \cdot r_d}{r_m \cdot r_t} - \frac{\omega_m(s) \cdot r_d}{r_m \cdot r_t} \cdot (B_m + J_m \cdot s) - \frac{\omega_d(s)}{r_t} \cdot J_d \cdot s \tag{7.17}$$

The two expressions for $F_b(s)$, *Equation (7.17)* and *Equation (7.16)*, are coupled:

$$V(s) \cdot (M \cdot s - B_{sys}) = \frac{\tau_m(s) \cdot r_d}{r_m \cdot r_t} - \frac{\omega_m(s) \cdot r_d}{r_m \cdot r_t} \cdot (B_m + J_m \cdot s) - \frac{\omega_d(s)}{r_t} \cdot J_d \cdot s \tag{7.18}$$

By using the relationship of ratios, the two rotational velocities are related. This is done as a step towards converting all rotational velocities to linear velocities.

$$\frac{r_m}{r_d} = \frac{\omega_d(t)}{\omega_m(t)} \qquad [\cdot]$$

$$\omega_d(t) = \frac{r_m}{r_d} \cdot \omega_m(t) \qquad \left[\text{rad} \cdot \text{s}^{-1}\right] \tag{7.19}$$

$\omega_d(t)$ is converted to a linear velocity and transformed to the Laplace domain:

$$v(t) = \omega_d(t) \cdot r_t \qquad \left[\text{m} \cdot \text{s}^{-1}\right]$$

$$\omega_d(t) = \frac{v(t)}{r_t} \overset{\mathcal{L}}{\Longrightarrow} \omega_d(s) = \frac{V(s)}{r_t} \tag{7.20}$$

$\omega_m(t)$ is converted to a linear velocity using *Equation (7.19)* and *Equation (7.20)* and transformed to the Laplace domain

$$\frac{\mathrm{r_m}}{\mathrm{r_d}} \cdot \omega_\mathrm{m}(\mathrm{t}) = \frac{\mathrm{v(t)}}{\mathrm{r_t}} \qquad \left[\mathrm{rad \cdot s^{-1}}\right]$$
$$\omega_\mathrm{m}(\mathrm{t}) = \frac{\mathrm{v(t) \cdot r_d}}{\mathrm{r_t \cdot r_m}} \overset{\mathcal{L}}{\Longrightarrow} \omega_m(s) = \frac{V(s) \cdot r_d}{r_t \cdot r_m} \tag{7.21}$$

Using the previously derived equations, *Equation (7.18)*, *Equation (7.20)* and *Equation (7.21)*, the final transfer function, describing the drivetrain, including the mechanical aspect from the motor, is constructed:

$$\mathrm{V(s) \cdot (M \cdot s - B_{sys})} = \frac{\tau_\mathrm{m}(\mathrm{s}) \cdot \mathrm{r_d}}{\mathrm{r_m \cdot r_t}} - \frac{\frac{\mathrm{V(s) \cdot r_d}}{\mathrm{r_t \cdot r_m}} \cdot \mathrm{r_d}}{\mathrm{r_m \cdot r_t}} \cdot (\mathrm{B_m + J_m \cdot s}) - \frac{\frac{\mathrm{V(s)}}{\mathrm{r_t}}}{\mathrm{r_t}} \cdot \mathrm{J_d \cdot s}$$

After rearranging to achieve standard form:

$$\frac{\mathrm{V(s)}}{\tau_\mathrm{m}(\mathrm{s})} = \frac{\frac{1}{\frac{\mathrm{r_m \cdot r_t}}{\mathrm{r_d}} \cdot \mathrm{B_{sys}} + \frac{\mathrm{r_d}}{\mathrm{r_m \cdot r_t}} \cdot \mathrm{B_m}}}{\frac{\frac{\mathrm{r_m \cdot r_t}}{\mathrm{r_d}} \cdot \mathrm{M} + \frac{\mathrm{r_d}}{\mathrm{r_m \cdot r_t}} \cdot \mathrm{J_m} + \frac{\mathrm{r_m}}{\mathrm{r_t \cdot r_d}} \cdot \mathrm{J_d}}{\frac{\mathrm{r_m \cdot r_t}}{\mathrm{r_d}} \cdot \mathrm{B_{sys}} + \frac{\mathrm{r_d}}{\mathrm{r_m \cdot r_t}} \cdot \mathrm{B_m}} \cdot \mathrm{s} + 1} \tag{7.22}$$

The mechanical part of the system is described by the above transfer function, *Equation (7.22)*, which in the following section is used in collaboration with the electrical model of the motor to describe the full velocity model of the vehicle.

Final Velocity Model

The expression for the output torque, τ_m, of the motor model, *Equation (7.7)*, along with the transfer function for the mechanical part, *Equation (7.22)*, delivers the information needed to make a visual representation of the velocity model. The input is the supply voltage, $U_a(s)$ delivered to the motor and the output is the vehicle's velocity, V(s), see *Figure 7.9*.

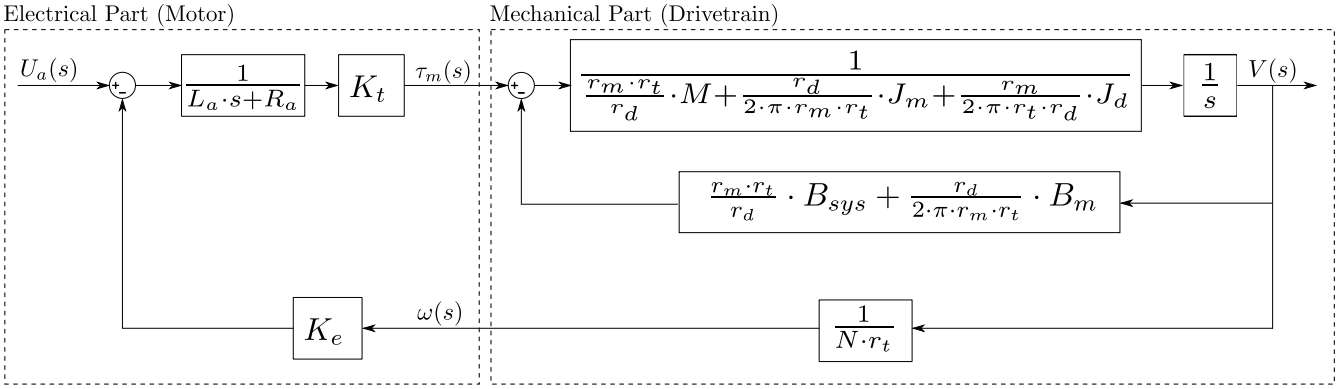


Figure 7.9: A block diagram of the combined drivetrain

Where:

- L_a is the armature inductance found in *Appendix B*
- R_a is the armature resistance found in *Appendix A*
- K_e is the generator constant found in *Appendix D*
- K_t is the motor constant found in *Appendix E*
- M is the mass of the vehicle found in *Section 2.2*
- J_m is the motor's moment of inertia found in *Appendix G*
- B_m is the friction of the motor found in *Appendix F*
- J_d is the drivetrain's moment of inertia which is unknown
- B_{sys} is the friction in the system, excluding the motor, which is unknown

Inspecting the above model of the drivetrain, the following two terms are extracted for closer analysis:

$$\frac{r_m \cdot r_t}{r_d} \cdot B_{sys} + \frac{r_d}{2 \cdot \pi \cdot r_m \cdot r_t} \cdot B_m \tag{7.23}$$

$$\frac{r_m \cdot r_t}{r_d} \cdot M + \frac{r_d}{2 \cdot \pi \cdot r_m \cdot r_t} \cdot J_m + \frac{r_m}{2 \cdot \pi \cdot r_t \cdot r_d} \cdot J_d \tag{7.24}$$

Expression 7.23, contains ratios and all frictions, whereas *Expression 7.24*, contains ratios and all inertias. The ratios brings the terms in each expression on the same base; in this

case linear velocity. What is left is the total friction, B_{tot}, of the system and the total inertia of the system, J_{tot}. Instead of measuring each subsequent part of the friction and inertia, measuring the total friction has the advantage of requiring less testing, which keeps sources of errors to a minimum.

Remembering the conversion from rotational to linear velocity in the drivetrain modeling, *Section 7.1*, it is know that the ratios convert to linear friction and inertia, hence the $\frac{1}{N \cdot r_t}$ in the feedback when going back to the motor. Since these ratios ends up being a constant on the signal going toward the output, the B_{tot} and J_{tot} terms can be measured either as rotational or linear factors. Here the only difference being whether $\frac{1}{N \cdot r_t}$ is placed on the feedback or $N \cdot r_t$ is placed on the direct therm. In this case it is chosen to consider B_{tot} and J_{tot} as factors on rotational velocity. These considerations yields the following model, *Figure 7.10*.

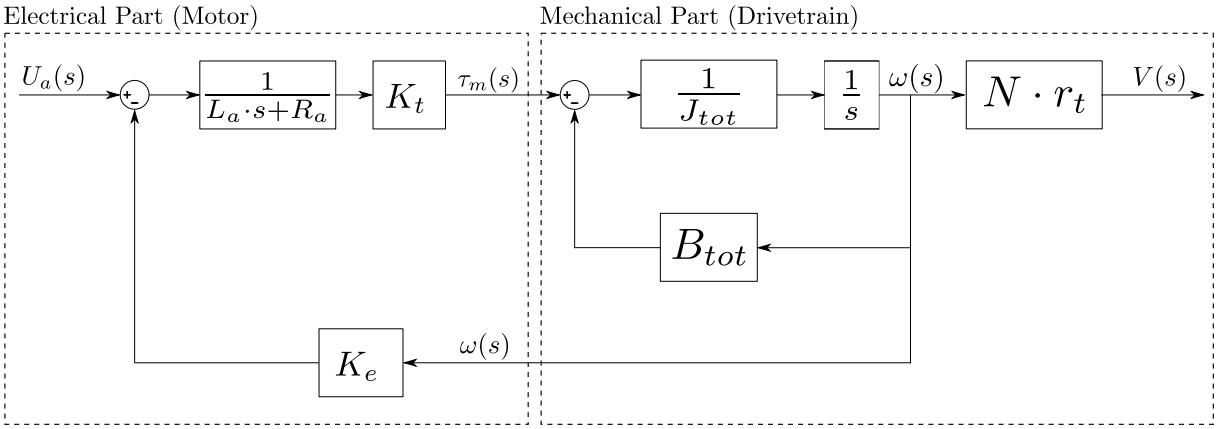


Figure 7.10: A block diagram of the combined drivetrain considering total inertia and friction acting on rotational velocity

In order to use this model in simulation, all terms must have a known value. To accomplish this, several tests has been carried out. Tests are carried out to determine the generator constant, K_e, the motor constant, K_t, the armature inductance, L_a, and the armature resistance, R_a, in *Appendix D*, *Appendix E*, *Appendix B* and *Appendix A*. The value of the total friction of the system, B_{tot}, is found in *Appendix J* and the total inertia, J_{tot}, in *Appendix K*. The gear ratio, N, and the radius of the drive wheel, r_t, are measured directly on the vehicle.

Simulation and Simplified Model

The previous sections show a velocity model of the system which is modelled in this section. As mentioned in the inertia test, *Appendix K*, the armature inductance is so small that it is not necessary to include for a good simulation. If the armature inductance is neglectable, and removed, the system will appear as a first order system instead of a second order. Therefore L_a is set to zero and hence removing the second order term. This effect is illustrated in the transfer function:

$$\begin{aligned}
\frac{V(s)}{U_a(s)} &= \frac{K_t}{L_a \cdot J_{tot} \cdot s^2 + (R_a \cdot J_{tot} + L_a \cdot B_{tot}) \cdot s + R_a \cdot B_{tot} + K_t \cdot K_e} \\
&\Downarrow \\
\frac{V(s)}{U_a(s)} &= \frac{K_t}{R_a \cdot J_{tot} \cdot s + R_a \cdot B_{tot} + K_t \cdot K_e} \\
\frac{V(s)}{U_a(s)} &= \frac{\frac{K_t}{R_a \cdot B_{tot} + K_t \cdot K_e}}{\frac{R_a \cdot J_{tot}}{R_a \cdot B_{tot} + K_t \cdot K_e} \cdot s + 1}
\end{aligned} \tag{7.25}$$

By inspection of the first order expression, in *Equation (7.25)*, the terms for the time constant and the gain are extracted:

$$\begin{aligned}
K &= \frac{K_t}{R_a \cdot B_{tot} + K_t \cdot K_e} \\
\tau &= \frac{R_a \cdot J}{R_a \cdot B_{tot} + K_t \cdot K_e}
\end{aligned}$$

To verify that this is a viable approximation, the following simplified model, in *Figure 7.11*, is created:

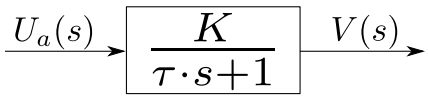


Figure 7.11: A block diagram showing a simplified model

Where K is the gain of the system and τ is the time constant. This simplified first order model of the system is simulated alongside the second order model, to verify that the vehicle can indeed be handled as a first order system. The plot of the two simulations are illustrated in *Figure 7.12*. From this figure it is impossible to see any difference between the two models, however, zooming in on the base of the step, see *Figure 7.13*, the difference becomes clear.

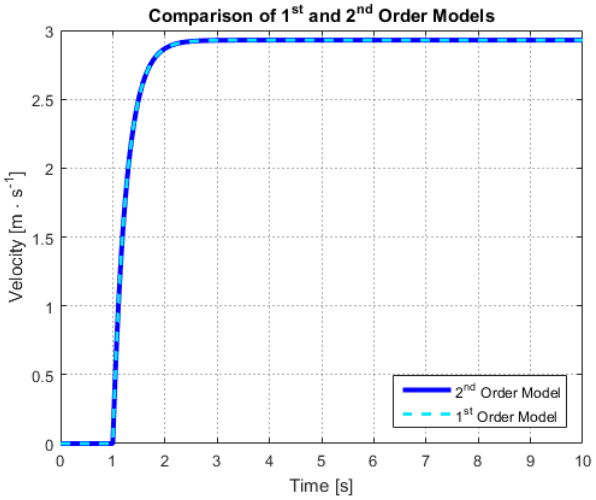


Figure 7.12: Comparison of the 1st and 2nd order models

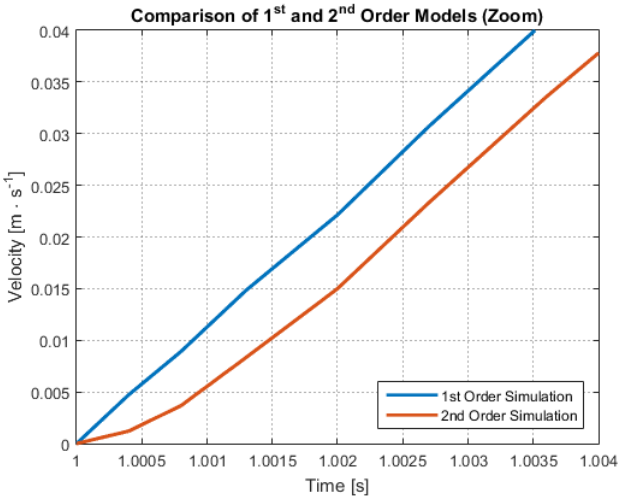


Figure 7.13: Comparison of the 1st and 2nd order models

However, a quick glance at the axes shows how minimal the impact of this difference must be. An other and possibly clearer way to show the difference of the two models, is through use of Bode plots. A Bode plot of the two models imposed on one another is seen on *Figure 7.14*.

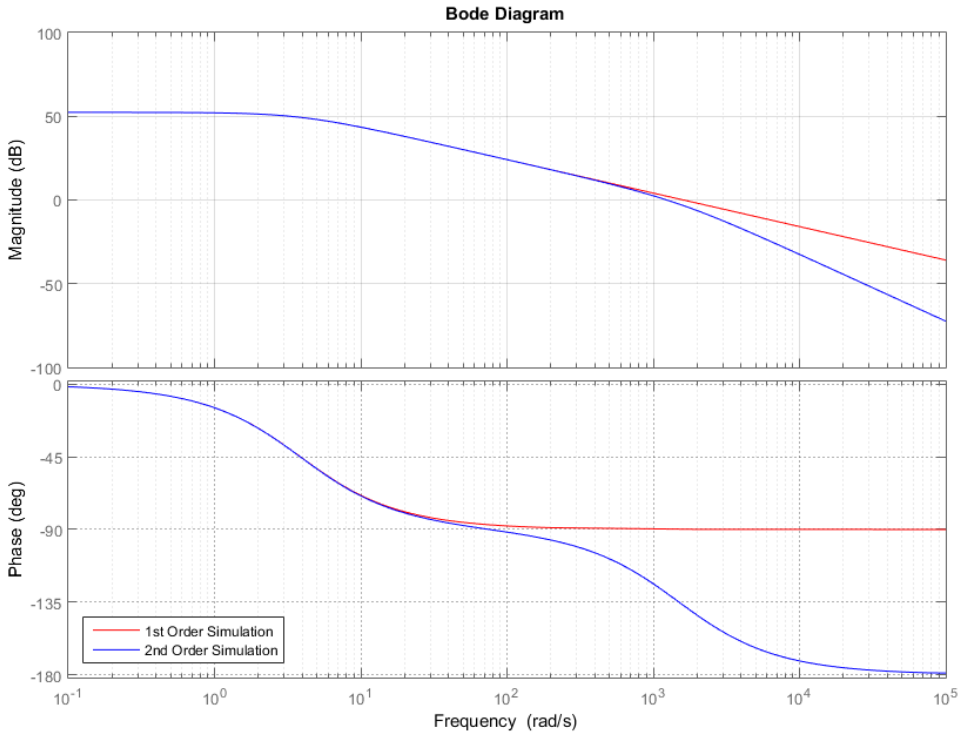


Figure 7.14: Bode plot showing the differences between the 1st and 2nd order models

For frequencies below 1000 $\text{rad} \cdot \text{s}^{-1}$ the two models are very similar because each of their first poles are very close; in 3,8759 $\text{rad} \cdot \text{s}^{-1}$ for the 1^{st} order model and 3,9 $\text{rad} \cdot \text{s}^{-1}$ for the 2^{nd} order model. The 2^{nd} order model however has an other pole which causes the phase shift and shift in magnitude at higher frequencies. However since this pole is so high, 1489,5 $\text{rad} \cdot \text{s}^{-1}$, it does not affect the lower pole considerably and the differences are way beyond significant frequency range.

Now the model is approximated with a first order model using a certain gain and time constant, deduced from the second order model. The parameters for this has been determined through a series of smaller tests, however determining the gain and time constant can also be done through testing of the system in its entity. Such tests are carried out and can be found in *Appendix L* and *Appendix K*, where the gain and time constant is found. The two approaches gives slightly different results when simulating, as seen on *Figure 7.15*. For both simulations a disturbance, sticktion, is added to the output. For the model parameters determined through the 2nd order model, sticktion is not accounted for. It is however accounted for when measuring the gain and time constant through testing of the entire vehicle. This is the reason of the offset between the two simulations. It also causes the negative offset for both simulations.

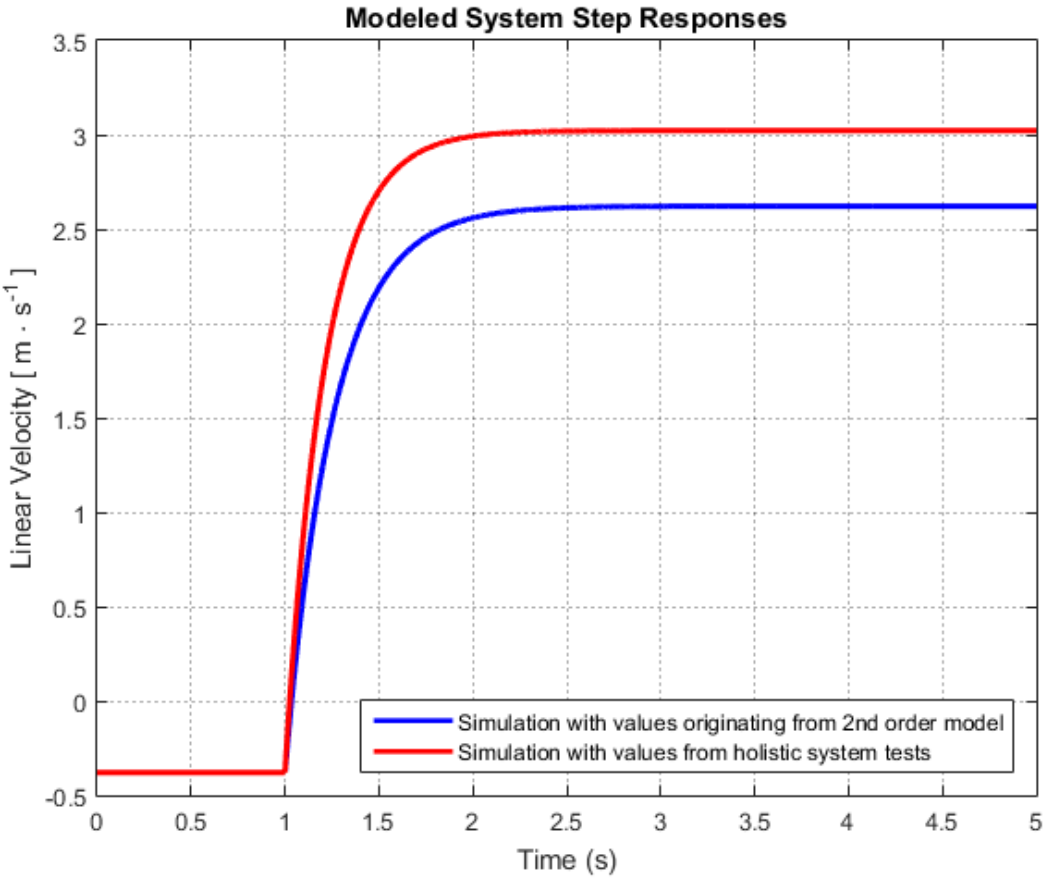


Figure 7.15: Plot showing differences between the first order model parameters, derived from the 2nd order model and for the other found through testing of the combined system

In the rest of the report the holistically measured gain and time constant is used in simulation of the plant as a first order system. In the following section this simplified model is verified directly by comparison with recorded data of the vehicle's step response.

Verification of the Velocity Model

In the former section a simplified first order model of the vehicle's velocity model was established. To ensure the approximation is a reliable model of the system, it is compared to a measured step-response of the vehicle. However, the simulation uses a first order model as described in the previous section, the gain, found in *Appendix L* and the time constant, found in *Appendix K*, is used as the first orders parameters. In *Figure 7.16* the simulated model's step-response and the measured step-response of the vehicle is illustrated.

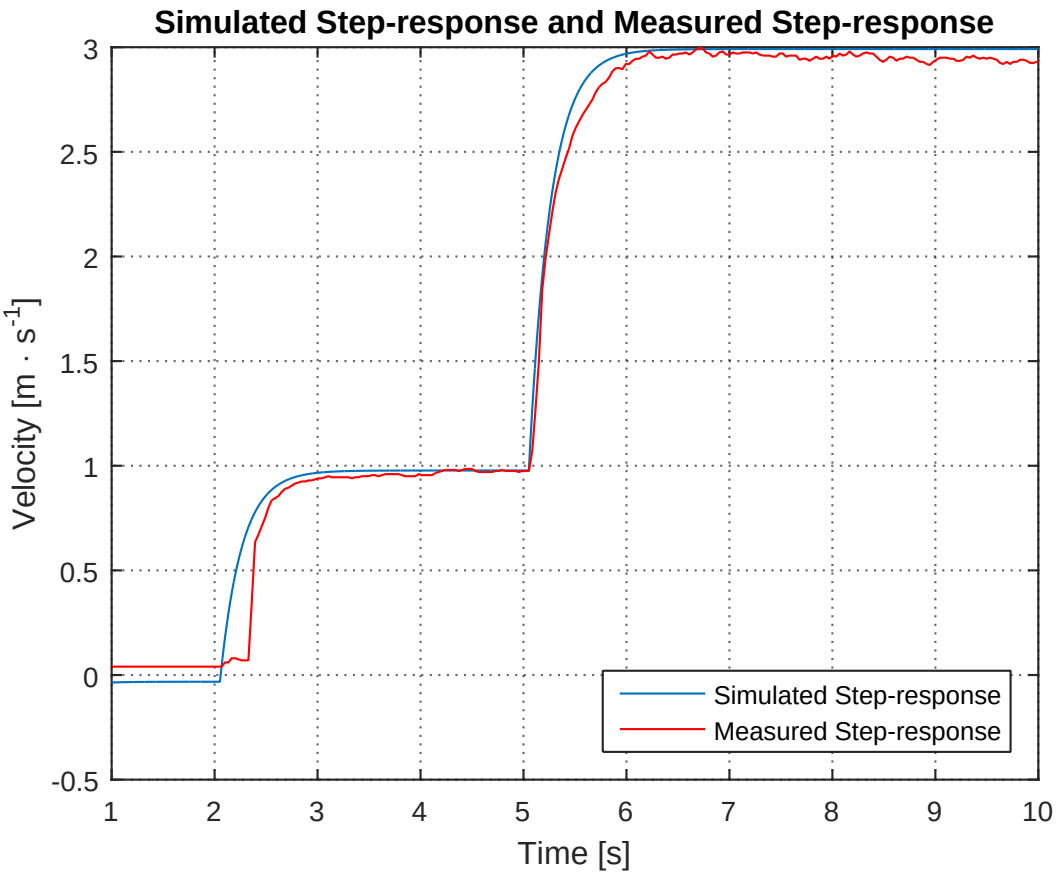


Figure 7.16: A plot illustrating a simulated step-response of the approximated velocity model (the blue line) and a measured step-response of the vehicle (the red line).

In *Figure 7.16* the red line is the measured data from the vehicle's step-response. To ensure enough data points is registered by the Hall sensor, when performing the step-response of the vehicle, the vehicle is set to drive at 1 $\mathrm{m} \cdot \mathrm{s}^{-1}$ at the start of the test. The vehicle preserves this velocity continuously for three seconds, before the vehicle is set to drive at its maximum velocity. The simulation (the blue line) is set to have the same milestones as the measured step-response of the vehicle. Furthermore it can be seen in *Figure 7.16* that the simulation starts before zero, this is due to rounding errors in simulated transfer-function, which is disregarded.

The occurrence of stiction in the start of the simulation and measured step-response, has been eliminated to a certain extent, which makes it insignificant. Furthermore, even though the vehicle is moving at a velocity of 0 $\mathrm{m} \cdot \mathrm{s}^{-1}$, the velocity is registered to be $0,04$ $\mathrm{m} \cdot \mathrm{s}^{-1}$, see why in *Section 5.1*.

The little bump in the beginning of the measured step-response, the red line, is caused by the lag of the Hall sensors. This is, however, disregarded. Additionally, the ripples seen on the measured step-response, when the vehicle is at its maximum velocity, could be caused by different kinds of noise factors, e.g.:

- Uneven belts
- Uneven floor
- Belts slipping on the floor
- Wear and tear on the internal gears

The data from the simulated and measured step-response illustrated in *Figure 7.16*, when compared is very similar, except for the bump and ripple on the measured data. Besides these two elements the approximated velocity model is sufficient.

By knowing how the velocity of the vehicle reacts to an applied voltage, it is possible to apply a control to it, see *Section 8.1*, and study the steering behavior of the vehicle. But first the steering needs to be modelled.

7.2 Steering Model of the Vehicle

After describing a model for the driving straight with the vehicle seen on *Figure 7.1*, the present section draws a model for the steering part, which is isolated in *Figure 7.17*. Thus, the focus is made on the relationship between the PWM command signal to the servo, the angle of the servo, and the resulting orientation of the vehicle. To facilitate the steering modelling process, the vehicle's velocity is considered only around an operating point.

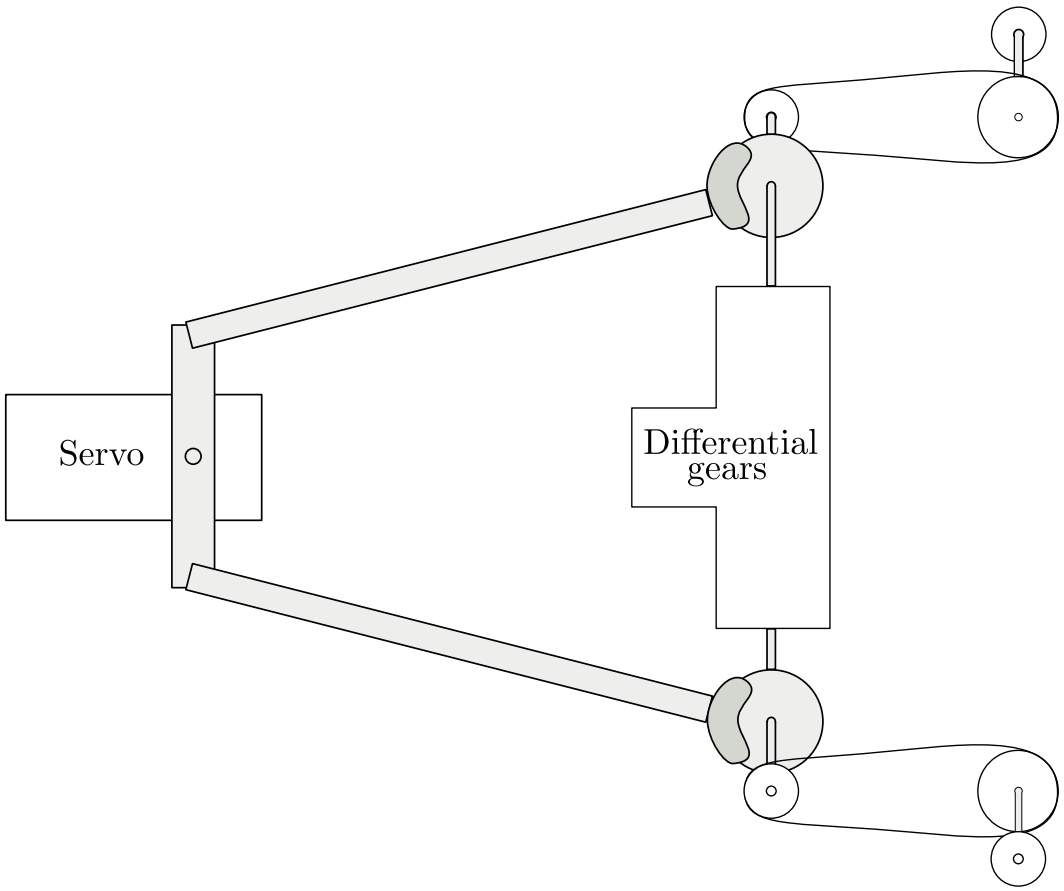


Figure 7.17: Mechanical drawing of the steering

As described in *Section 2.2*, when the servo turns one way, it pulls one of the arms, which in turn moves a brake pad towards the brake disc. This adds friction and will thereby slow down the rotating shaft connected the differential gears and drive gear. The differential gears will then transfer the power from one belt to the other, making the vehicle turn.

Directional model

As the steering system contains many moving parts, it is convenient to start with a simple model, to verify it, and iterate until it is satisfactory.
The first model considered can be seen on *Figure 7.18*.

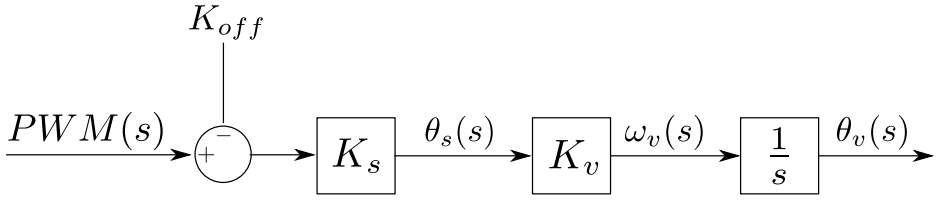


Figure 7.18: A basic steering model

As described in *Section 2.2*, the angle of the servo, θ_s, is controlled by a pulse width modulated signal on its reference input. The servo is in a middle position (0°) for a non-zero pulse-width. This means that an offset, K_{off}, is substracted to the PWM so that the servomotor's gain, K_s, translates linearly its input to an angle. As seen in *Section 2.2*, this offset is different than the one used to get the middle position of the servo, so that it compensates for the mechanical inaccuracies and makes the vehicle go straight. Moreover, depending on the operating point of the vehicle, i.e. if it goes to the right or to the left, this offset shall also include a supplementary term. The latter is used to get the servo at the edge of the steering's linear area, be it to the left or to the right, see *Section 2.2*.

In the following modelling process, the vehicle's steering is considered as being in either of the two operating points. Since the velocity of the vehicle is also assumed constant, the rate of change of the direction, $\omega_v(s)$, must be a function of the servo angle, and a constant, K_v, representing the action speed of the vehicle and the braking system. The rate of change in the vehicle's angle is finally integrated over time, resulting in a angle heading, $\theta_v(s)$.
As stated in *Appendix N*, the servomotor's gain, K_s, and velocity and braking gain, K_v are combined into a single gain value $K_{steering}$ to ease the gain calculations, see *Figure 7.19*.

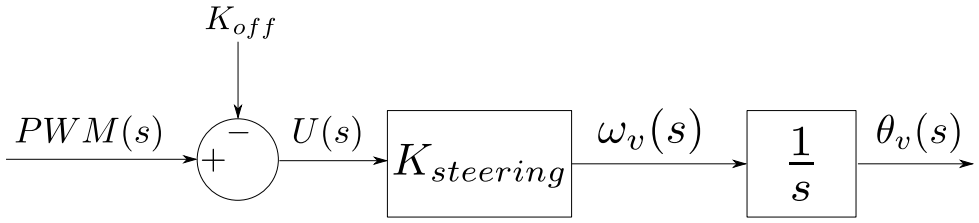


Figure 7.19: A basic steering model with combined gains

Extension of the Directional Model

The first model describes in a simple manner how the mechanical part of the steering system functions. However, it can be extended to include the time delay caused by the servo. According to *Section 2.2*, the servo is controlled by a PWM signal with a period of 30 ms. This means, that it will not be possible to update the servo angle continuously, but only in discrete time steps of 30 ms. These steps will be implemented in the model as a sampling delay.
The delay of a signal, u(t), is usually described in frequency domain by an exponential factor [38]:

$$\mathcal{L}\left[\mathrm{u}(\mathrm{t}+\lambda)\right] = \mathrm{U}(\mathrm{s}) \cdot \mathrm{e}^{-\lambda \cdot \mathrm{s}}$$

However, for small values of λ, it is possible to use the approximation:

$$\exp(-\lambda \cdot \mathrm{s}) \simeq \frac{1}{\lambda \cdot \mathrm{s}+1}$$

With the delay from the servo, λ, being 30 ms, this approximation is used and inserted into the model after the application of intrinsic offset and before the action of the steering mechanism itself, as shown on *Figure 7.20*:

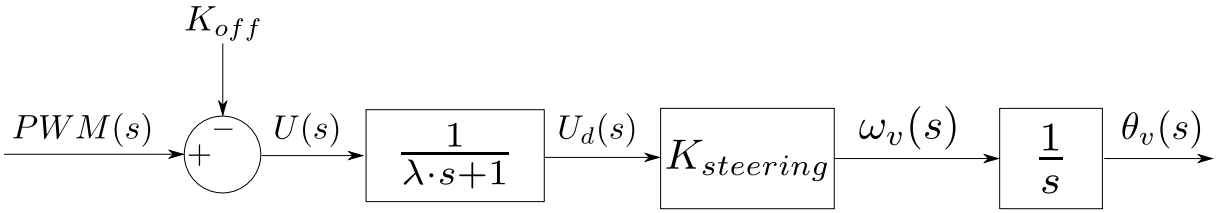


Figure 7.20: A basic steering model

Where:

$\mathrm{K}_{\mathrm{steering}}$	is the steering gain found in *Appendix N*
λ	is the delay from the servomotor found in *Section 2.2*

To be sure that this model can be used for the design of controllers, it is necessary to verify it and check the fact that it matches the reality, by comparing a simulation of the model with measured data of a step-response.

Verification of the Directional Model

A simple model of the steering mechanisms has been built in the previous section. To verify the accuracy of this model compared to reality, a test and a simulation of the plant are plotted on the same graph, see *Figure 7.21*.

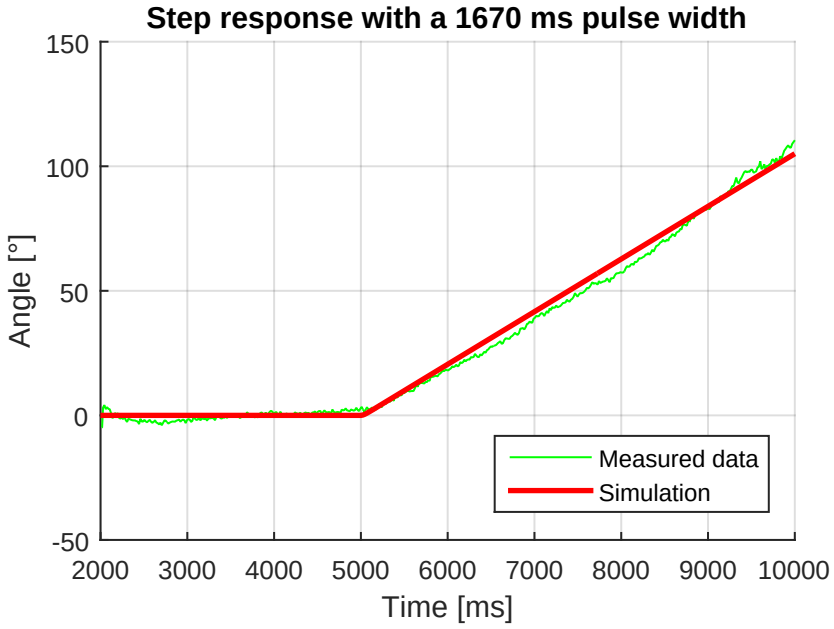


Figure 7.21: Plot of the plant model, showing the simulation and the real life test.

The vehicle is started with the servo receiving a 1450 ms PWM signal, so that the vehicle goes straight during 5 s and reaches a steady speed of $1,4$ ms. After this time, a step of 1670 ms PWM is sent to the servo,that makes the vehicle turn at a constant rate. This makes the angle vary linearly as seen on *Figure 7.21*.

The simulated plant appears to react in a very similar way as the real one, except for two small differences.
At the beginning, the vehicle's angle seems to increase slightly while the simulated vehicle's heading stays at 0°. This is probably due to the approximation made when tweaking the pulse width for the vehicle to go straight.
Similarly, after the step is applied to the system, the angle of the vehicle varies barely faster in the test than in the simulation. The reasons for this slight difference might be because of the test environment and conditions (ground humidity and texture) and thus, because of variation in the offset needed to steer linearly, as stated in *Section 2.2*.

Despite these neglectable discrepancies, the previously established model fits the reality quite closely and is therefore used hereafter in the project. This simple model describes the action of the steering on the vehicle, by translating PWM signals sent to the servo into headings of the vehicle. It allows to apply a control on the direction of the vehicle, but not for it to follow a predefined set of points.

Distance Model

Since the vehicle has to follow a predetermined route, a direction control alone is not enough. As seen on *Figure 7.22*, any change of direction caused by a disturbance, will cause a deviation from the planned line between two points A and B on the wanted route. This is why a model of this deviation is created in this section.

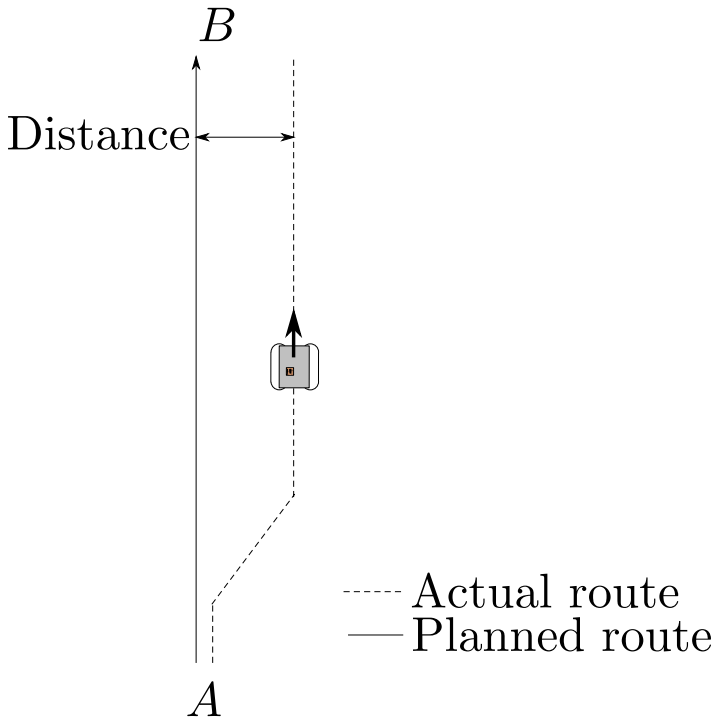


Figure 7.22: Consequence of using directional control alone

How large the deviation is, should depend on the vehicle's erroneous angle from the wanted line, its velocity and the time it takes for the control system to account for the error. The instantaneous distance, d_1, can be expressed with simple trigonometry as:

$$d_1 = v_1 \cdot t_1 \cdot \sin(\Delta\theta_1) \quad [\text{m}] \quad (7.26)$$

Where:

d_1	is the distance of the vehicle from the wanted line at instant t_1	[m]
t_1	is the time instant of measurement	[s]
v_1	is the vehicle's velocity at instant t_1	$[\text{m} \cdot \text{s}^{-1}]$
$\Delta\theta_1$	is the vehicle's angle compared to the wanted line at instant t_1	[rad]

The speed is assumed to be constant. From this assumption, the distance error over time, d(t), is then described as an integration over time of the sine of the error angle multiplied with the velocity, see *Equation (7.27)*.

$$d(t) = v \cdot \int^{t} \sin\left(\Delta\theta(t)\right) dt \quad [m] \quad (7.27)$$

Where:

$d(t)$	is the distance of the vehicle from the wanted line	[m]
t	is the time variable used for the integral	[s]
v	is the constant vehicle's velocity	$[m \cdot s^{-1}]$
$\Delta\theta(t)$	is the vehicle's angle compared to the wanted line	[rad]

The integration over time actually multiplies the integrated sine function with a time interval. Moreover, the sine function output being dimensionless, the resulting function, $d(t)$, has the dimension of a length and the unit of meters: it is the time varying distance between the vehicle and the wanted line.
To facilitate the process of Laplace transform, it is necessary to linearize the sin function. By assuming that the vehicle is in its operating point, i.e. within a short distance and small angle from the wanted line, the angle difference, $\Delta\theta(t)$, is then very small. For small angle values, the sin function can be approximated to :

$$\sin\left(\Delta\theta\right) \simeq \Delta\theta$$

Thus, the *Equation (7.27)* is simplified into:

$$d(t) = v \cdot \int^{t} \Delta\theta(t) dt \quad [m] \quad (7.28)$$

The transformation of *Equation (7.27)* into the Laplace domain eventually yields:

$$D(s) = v \cdot \frac{1}{s} \cdot \Delta\theta(s) \quad [m] \quad (7.29)$$

By definition, $\Delta\theta$ is the difference between the real heading of the vehicle, θ_v, and the reference angle, θ_{ref}. Since both angles are given or measured in degrees, it is also needed to convert them to radians to match the units, as shown in *Equation (7.30)*.

$$D(s) = \frac{\pi}{180} \cdot v \cdot \frac{1}{s} \cdot \left(\theta_v(s) - \theta_{ref}(s)\right) \quad [m] \quad (7.30)$$

Where:

$\theta_v(s)$	is the absolute real heading of the vehicle	[°]
$\theta_{ref}(s)$	is the absolute wanted heading of the vehicle	[°]

From *Equation (7.30)*, the block diagram of the steering model on *Figure 7.19* can be extended with the boxed blocks seen on *Figure 7.23*.

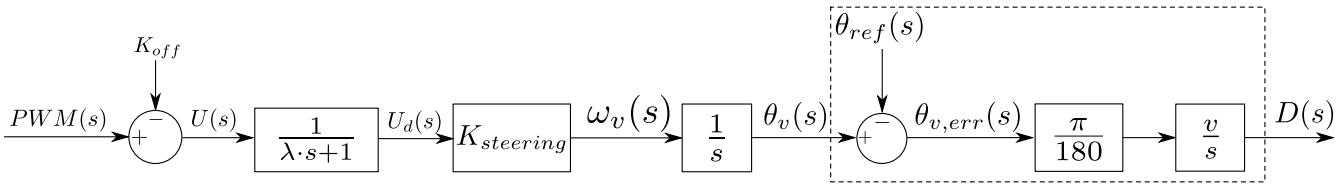


Figure 7.23: Block diagram of the combined directional and Distance models

This model describes how the steering reacts and therefore, how the vehicle can be moved sideways. It allows for the controlling of both the heading of the vehicle and its position relative to the line it has to follow, see *Section 8.2*.
However, it also uses assumptions that need to be true when running the lawn mower. Firstly, the distance calculation model only works if the vehicle is deviating and not if it starts on a line parallel to the wanted one. Indeed, were that the case, then the angle difference would approach zero and the calculated distance too, even though it wouldn't actually be. To avoid the problem, the vehicle shall start on the right line path. The issue of returning to the right path in case of disturbance (lawn mower slipping or hurting an object) is considered in *Section 8.2*.
Secondly, this model is only used to simulate the system's behavior since the distance calculations are made with the use of the Games on Track positioning system.
Finally, the velocity can only be considered constant if properly controlled, as done in *Section 8.1*.

A velocity and steering model is established. It is thereby possible to design controllers for regulating the velocity and angle.

8 | Control of the Vehicle

With both the velocity and steering model established, controllers can be designed for the vehicle to become autonomous.
The first controller to be made in this chapter is on the velocity system to allow the vehicle to run at steady speeds. It will then be possible to control the steering, first to get the vehicle going in the wanted direction and finally to have it following a predefined route.

8.1 Velocity Controller

The purpose of the velocity controller is to preserve the vehicle at a steady velocity, when going uphill, downhill and when turning, this is in conjunction with the velocity requirement from *Section 3*. The three most widely utilized controllers are the proportional, integral and differential controllers [39]. Usually all three controllers combined are not required when controlling a system. Different controller approaches are explored in the following segment, starting with the controller which is the most frequently utilized controller, alone or in combination with others, the proportional controller.

P-Controller

As illustrated on *Figure 8.1* the P-controller, K_p, is a proportional gain which is multiplied with the direct term.

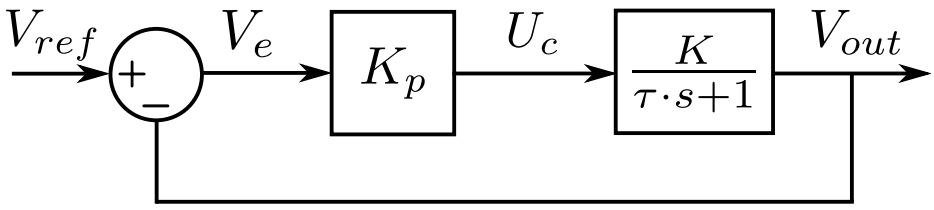


Figure 8.1: Diagram of the proportional controller

The illustrated diagram yields the following closed loop transfer function:

$$\frac{V_{out}}{V_{ref}} = \frac{\frac{K_p \cdot K}{\tau \cdot s+1}}{1+\frac{K_p \cdot K}{\tau \cdot s+1}} = \frac{1}{\frac{\tau \cdot s+1}{K_p \cdot K+1}+1} = \frac{\frac{K_p \cdot K}{K_p \cdot K+1}}{\frac{\tau}{K_p \cdot K+1} \cdot s+1} \tag{8.1}$$

From *Equation (8.1)* it is evident that the time constant and the gain for the system is dependent on the chosen K_p. The time constant of the closed loop transfer function is the coefficient of s in standard form:

$$\tau_{closed} = \frac{\tau}{K_p \cdot K+1}$$

As an example, the K_p is chosen such that it cancels out the gain, K, of the plant. This results in a time constant and a gain which is reduced to half:

$$\frac{V_{out}}{V_{ref}} = \frac{\frac{\frac{1}{K}\cdot K}{\frac{1}{K}\cdot K+1}}{\frac{\tau}{\frac{1}{K}\cdot K+1}\cdot s+1}$$
$$\frac{V_{out}}{V_{ref}} = \frac{\frac{1}{2}}{\frac{\tau}{2}\cdot s+1} \tag{8.2}$$

This results in the P-controller giving an output of half the input, but rise to its set-point twice as fast as the system step without control. This is tested utilizing the vehicle and the response is as expected. The test can be seen in *Appendix M*.

If a system where an input velocity is equal to an output velocity is desired, a P-controller is insufficient. The reason lies in the numerator in *Equation (8.1)*. No matter the value of K_p the numerator will always be less than 1. This results in an offset. A larger K_p resulting in a smaller offset, will make the coefficient of s in *Equation (8.1)* (time constant) smaller, which can result in larger overshoot.

P-Controller with Feed Forward

The P-controllers difference between the reference and set-point, steady state error, can be diminished without compromising the time constant, through use of feed forward, see *Figure 8.2*.[40]

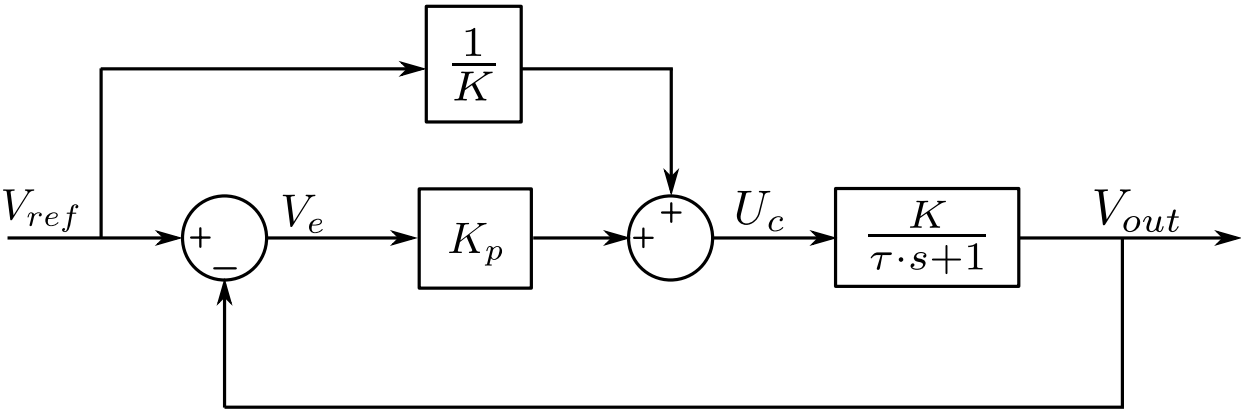


Figure 8.2: Diagram of the proportional controller with feedforward

In this design the set-point is changed by feed forwarding the desired value of the output, V_{out}, to the input of the plant. Thereby summing up the desired value with the error, V_e, fed through the P-controller. The gain located on the feed forward ensures the input, V_{ref}, has the same unit as the signal going into the plant, in the case volts, U_c. Since the

system gain, K, converts volts into linear velocity, the gain in the feed forward is set to $\frac{1}{K}$. The control loop in *Figure 8.2* yields the following closed loop transfer function:

$$\frac{V_{out}}{V_{ref}} = \frac{\frac{K \cdot \frac{1}{K} + K \cdot K_p}{1 + K \cdot K_p}}{\frac{\tau}{1 + K \cdot K_p} \cdot s + 1}$$

If the K_p value is selected to cancel out the system gain, as with the P-controller, allowing for velocity directly on the controller input, the following equation emerges from the closed loop transfer function:

$$\frac{V_{out}}{V_{ref}} = \frac{\frac{K \cdot \frac{1}{K} + K \cdot \frac{1}{K}}{1 + K \cdot \frac{1}{K}}}{\frac{\tau}{1 + K \cdot \frac{1}{K}} \cdot s + 1}$$

$$\frac{V_{out}}{V_{ref}} = \frac{1}{\frac{\tau}{2} \cdot s + 1}$$

If this is compared to the resulting closed loop transfer function for the original P-controller, *Equation (8.2)*, it can be seen that the steady state error is removed. Notice that the time constant of the closed loop remains half of that of the plant.

By utilizing the feed forward to place the set-point of the controller, P-control becomes a viable option for controlling a velocity. This is tested utilizing the vehicle and the response is as expected. The test can be seen in *Appendix M*. A Comparing of a step response utilizing the vehicle and a simulation of a step response is illustrated in figure *Figure 8.3*.

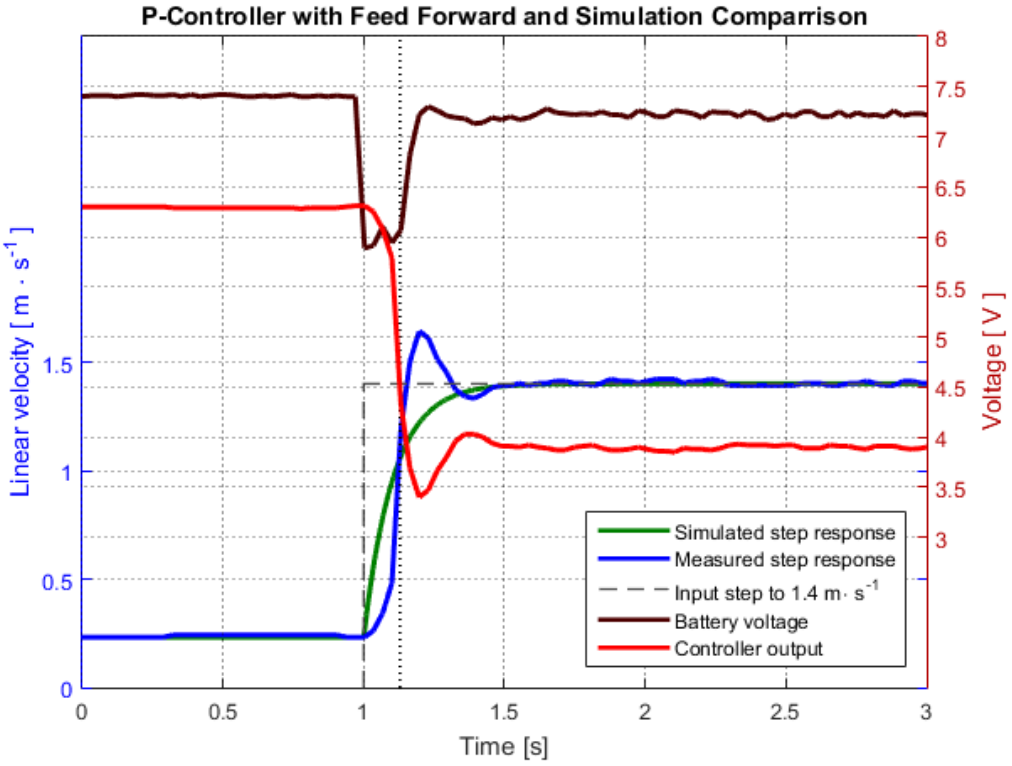


Figure 8.3: Proportional control with feed forward. The simulation is first order while the vehicle is second order due to the armature coils in the motor.

There are a few things to notice. First, the rise time of the two are approximately equal, however, the simulation of the first order approximation rises directly, while the measured data has a delayed initial rise. This delay is presumably because of the armature coils in the motor, through which the current cannot change instantaneously. This effect gives rise to a second order term, as discussed in *Section 7.1*, where the first order approximation was made. After the armature coils have been energized the controller output has a much faster effect on the velocity of the vehicle, causing it to rise so fast that it overshoots due to inertia.

An other interesting thing to notice is the sudden drop in battery voltage. When large currents are drawn from a NiMH battery, the voltage drops[9]. However in the implementation the duty cycle is scaled according to the current battery voltage, so that the controller maintains the same effect regardless of battery voltage. The exception being when the controller output exceeds the battery voltage, in which case the control will follow the battery voltage until the battery recovers. The P-controller with feed forward seems to be a good choice, however if subjected to disturbances some problems emerge. A disturbance imposed on the system is analogue to a sudden change of the gain of the plant. since the set-point compensation origins at the input of the controller, the sudden change in gain of the plant cannot be compensated for with this feed forward P-controller. The effect is demonstrated in *Figure 8.4*, where the feed forward controller is going up a

hill.

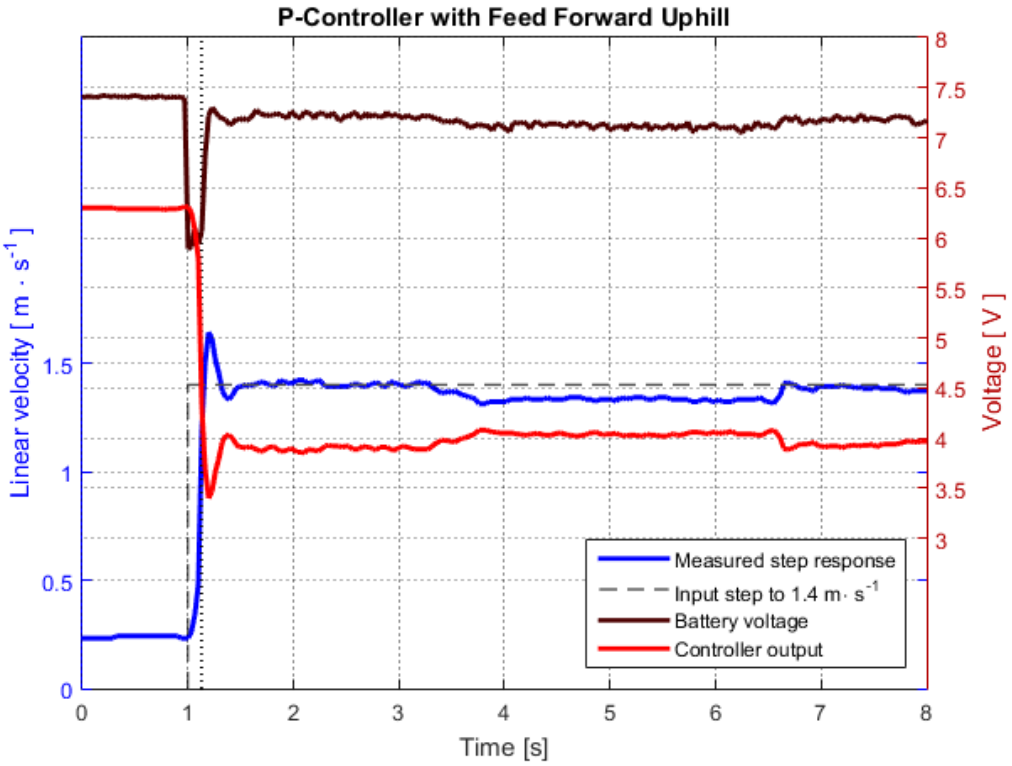


Figure 8.4: When the P-controller with feed forward is subjected to constant disturbances, the gain of the system changes and the proportional gain and set-point is no longer sufficient, to avoid steady state error.

Since the vehicle steers by breaking on one side, this kind of steady state error will not only occur when the vehicle is encountering slopes, but also every time the vehicle turns.

PI-Controller

To attack the new offset-problem caused by changes in the system gain when disturbances affects the system, a proportional integral controller is the next natural choice. The reason for choosing a PI-controller is because the I-component integrates over the error. This component therefore increases or decreases over time, and therefore has an adaptive effect on the system. The design is illustrated in *Figure 8.5*.

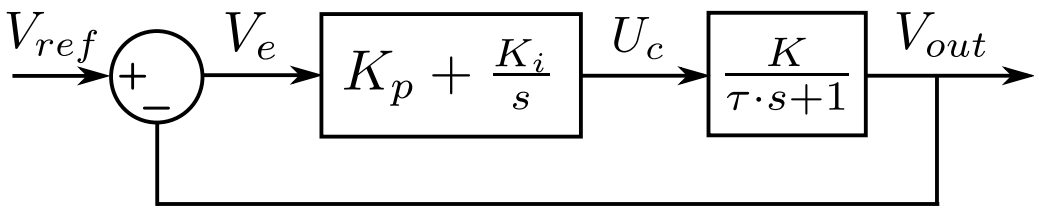


Figure 8.5: Diagram of the proportional integral controller

The standard equation for a PI-controller can be rewritten to:

$$K_p \cdot \left(1 + \frac{1}{T_i \cdot s}\right) = K_p \cdot \frac{T_i \cdot s + 1}{T_i \cdot s}$$

Where T_i is the time constant of the integrator. Now if the time constant of the integrator is matched to the time constant of the plant, that is $T_i = \tau$, the following equation emerges from the closed loop transfer function:

$$\frac{V_{out}}{V_{ref}} = \frac{K_p \cdot \frac{\tau \cdot s+1}{\tau \cdot s} \cdot \frac{K}{\tau \cdot s+1}}{1 + K_p \cdot \frac{\tau \cdot s+1}{\tau \cdot s} \cdot \frac{K}{\tau \cdot s+1}} \quad \Leftrightarrow \quad \frac{V_{out}}{V_{ref}} = \frac{K_p \cdot \frac{K}{\tau \cdot s}}{1 + K_p \cdot \frac{K}{\tau \cdot s}}$$

Inserting $K_p = \frac{1}{K}$, yields the following:

$$\frac{V_{out}}{V_{ref}} = \frac{\frac{1}{K} \cdot \frac{K}{\tau \cdot s}}{1 + \frac{1}{K} \cdot \frac{K}{\tau \cdot s}} \quad \Leftrightarrow \quad \frac{V_{out}}{V_{ref}} = \frac{\frac{1}{\tau \cdot s}}{1 + \frac{1}{\tau \cdot s}} \quad \Leftrightarrow \quad \frac{V_{out}}{V_{ref}} = \frac{1}{\tau \cdot s + 1}$$

This is equivalent to the plant but with a gain of 1 instead of K, which is desirable, and so also the reason for inserting $K_p = \frac{1}{K}$ in *Equation (8.1)*. Now to determine K_i, the original equation for a PI-controller is evaluated:

$$K_p + K_i \cdot \frac{1}{s} = K_p \cdot \left(1 + \frac{1}{T_i \cdot s}\right) \quad \Rightarrow \quad K_i \cdot \frac{1}{s} = \frac{K_p}{T_i \cdot s} \quad \Rightarrow \quad K_i = \frac{K_P}{T_i} \quad \Rightarrow \quad K_i = \frac{K_p}{\tau}$$

This concludes the initial design of the PI-controller, however, in the following segment the controller will be analyzed and compared to the other controllers. Implementation and discussion of results follow immediately after.

Comparison of the Controllers

On *Figure 8.6* the different controller designs are simulated being subjected to a velocity step of 1.4 $m \cdot s^{-1}$. Furthermore, to compared the controllers with a step response of the plant, the plant is subjected to a voltage step calculated from the velocity step and the gain of the plant, corresponding to the value K_p. Since it is a step response of a plant, and not a controller, no feedback is utilized.

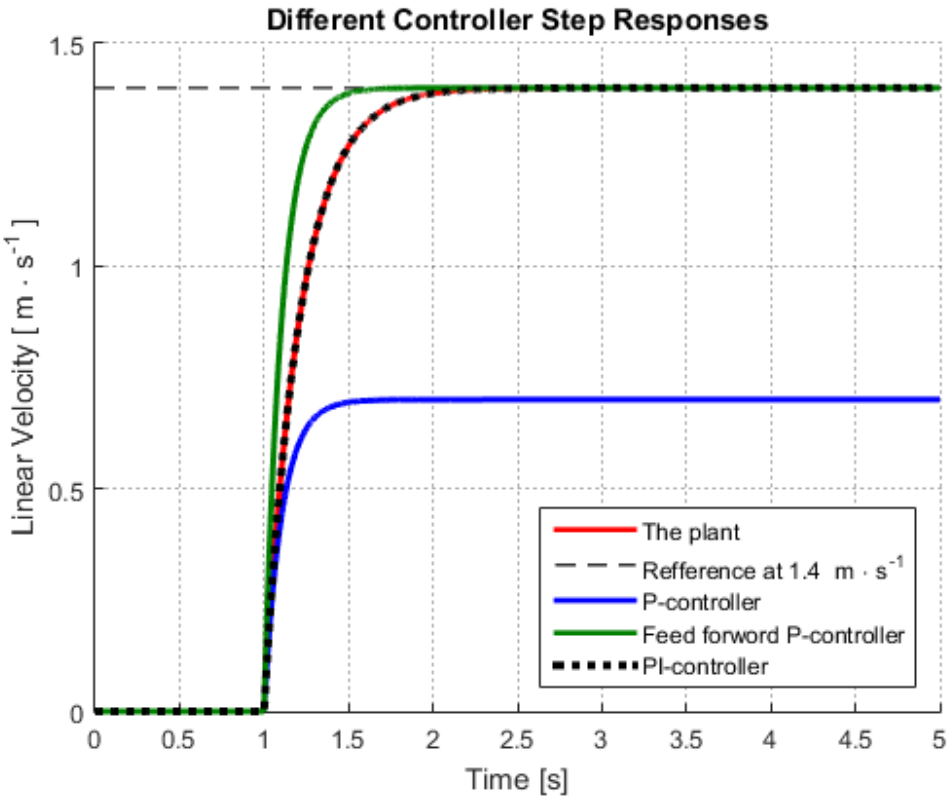


Figure 8.6: Simulations of the different controller designs along with the first order system model, plant, which in this simulation has an added scale-factor to obtain a gain of 1.

A thing to notice on *Figure 8.6*, is the inadequacy of the P-controller, given by its steady state error. The feed forward P-controller however, solving this problem, seems like a very good solution if consulting only its step response. As discussed, a problem arises when introducing a disturbance, as seen in *Figure 8.4*. The last discussed option is the PI-controller, which places itself right on top of the step response of the plant when having a gain of 1, which is by design. The difference between the plant with a scaled input to obtain a gain of 1, compared to the PI-controller lies in the feedback along with the adaptive gain emerging from the I-component. It becomes clear when investigating the open loop rather than the closed loop transfer function:

$$V_{error}(s) \cdot \frac{(K_p \cdot \frac{K_i}{s}) \cdot K}{\tau \cdot s + 1} \Bigg|_{\substack{K_p = \frac{1}{K} \\ K_i = \frac{K_p}{\tau}}} \Rightarrow V_{error}(s) \cdot \frac{1}{\tau \cdot s} \xRightarrow{\mathcal{L}^{-1}} \frac{1}{\tau} \cdot \int_0^t V_{error}(\tau_i)\, d\tau_i$$

Where:

- τ_i is an integration variable
- V_{error} is the error from $V_{ref} - V_{out}$, see *Figure 8.5*

This illustrates how the integral component, of the controller, integrates over the error from the reference to the output over time.

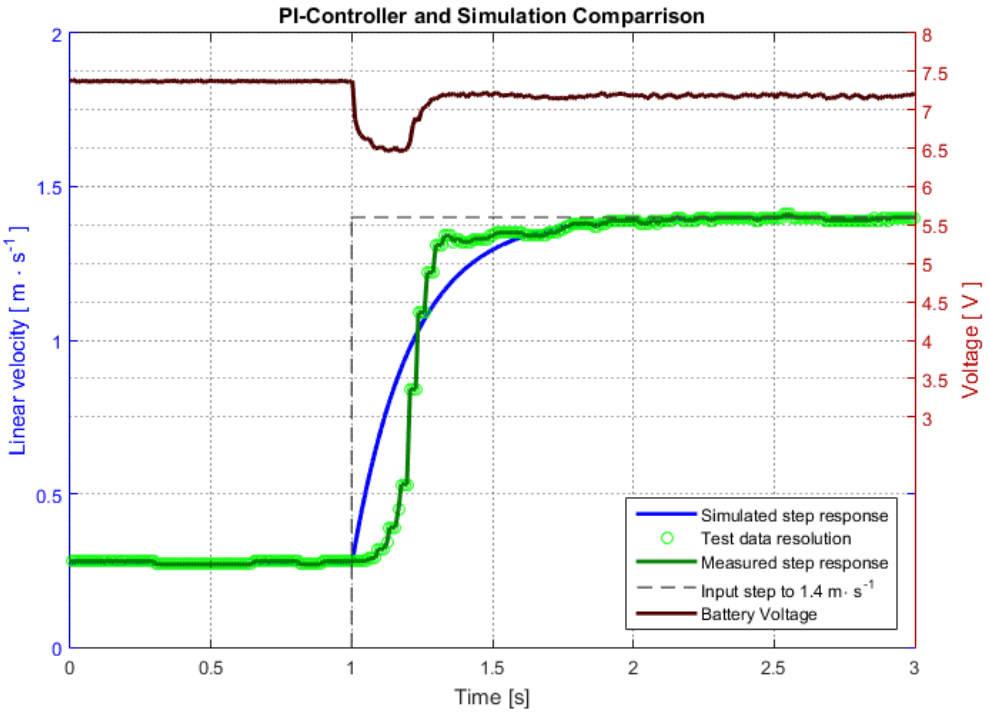


Figure 8.7: Step response of the proportional integral controller compared to simulation

The difference between the two responses at the bottom of the graph might, as mentioned for the feed forward P-controller, be partially due to the fact that the vehicle is a second order system. The approximation to a first order model is however not to be discarded. The design still delivers a response relatively close to the simulation. If this implementation is investigated at different velocity steps, a certain characteristic of the current design can become more apparent, see *Figure 8.8*.

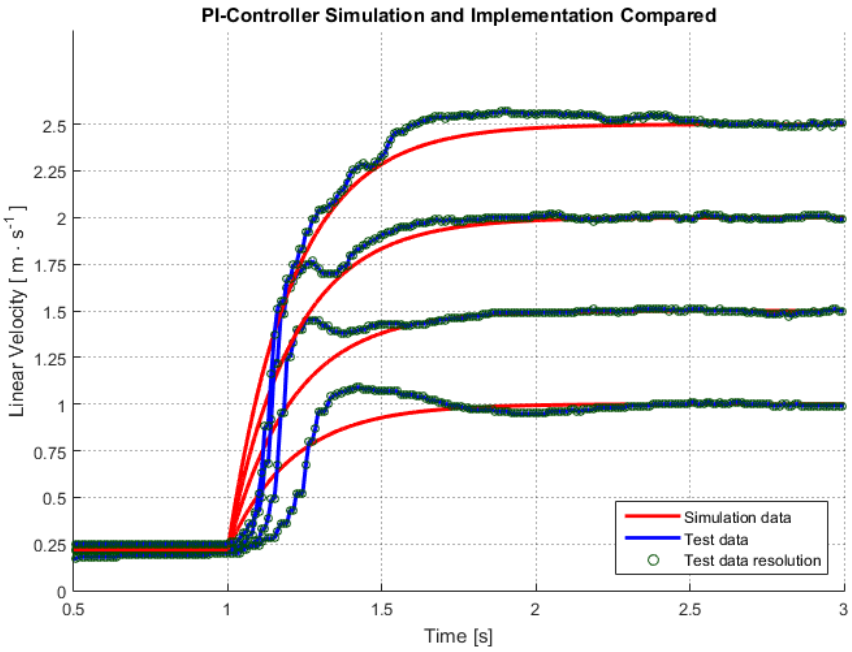


Figure 8.8: Diagram of the proportional integral controller subjected to different velocity steps, both in simulation and test

The system undershoots before reaching the reference velocity. To investigate this behavior further a step is made at 2 $\mathrm{m} \cdot \mathrm{s}^{-1}$, and data from the battery and controller output is recorded, see *Figure 8.9*.

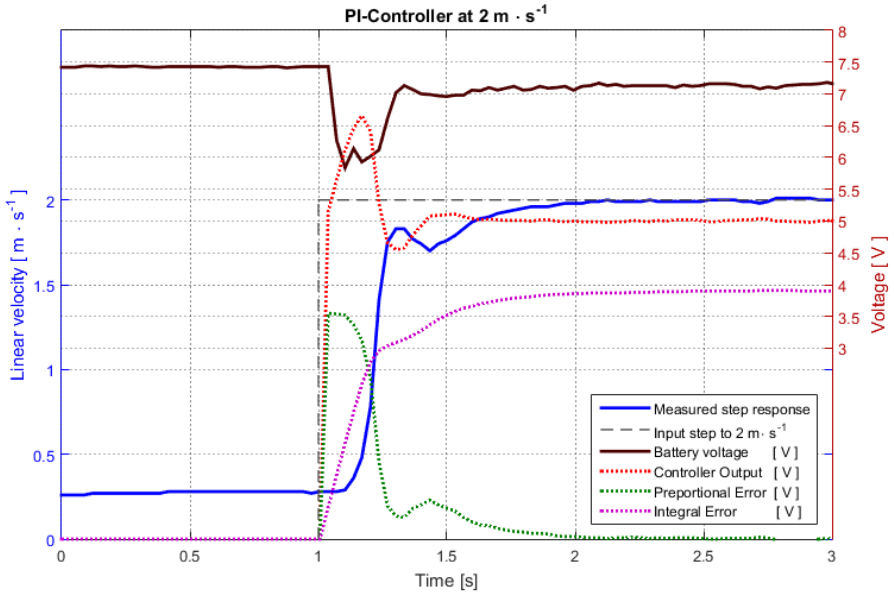


Figure 8.9: Step response of the proportional integral controller, also showing controller output going into battery saturation

As mentioned earlier, in this implementation the controller output is converted to duty cycle taking into account a varying battery voltage. So the battery voltage has an impact on the control only when the controller output exceeds the battery voltage. As a side note, the battery voltage is limited to a maximum of 100 % duty cycle. The overshoot happens after the battery voltage saturation, and so the answer must be found elsewhere.

When the velocity of the vehicle rises, the error naturally falls, and the proportional error follows the error. The controller output follows this fall of the error, while the integral gain keeps building. This means that the controller output will keep falling until the integral gain takes over, which happens too late, compared to the settle value (5 V) of the controller output, causing it to overshoot. A smaller K_p would decrease the controller output overshoot by giving less gain in the rise, also increasing the time constant. The delay from the overshoot of the controller to the velocity is caused by inertia in the system.

Implementation of the PI controller

The functional part of the implementation is seen in Listing 8.1. The *Actualspeed* is set from the average of recorded speeds of the two belts, from the Hall sensors. Then the error is calculated by comparison between the feedback, *Actualspeed*, and the reference, *Wantedspeed*. A quick glance at *Figure 8.9*, reveals a problem, where the controller output exceeds the battery voltage. This is called integral windup, and can be prevented by implementing anti windup, which locks the integral error so that it does not make the controller output voltage exceed the saturation of the battery[41]. The first if-statement in Listing 8.1, is a very simple implementation of an anti-windup functionality, and the effect is clearly demonstrated by repeating the test from *Figure 8.9* with anti windup, as seen on figure *Figure 8.10*.

```
1  Actualspeed = (speed0 + speed1)/2; // Average speed of the vehicle
2
3  Error = Wantedspeed - Actualspeed;                  //  ANTI-WINDUP:
4                                                      //  Only increase integral error if
5  if( ControllerOutput < ((float)batReading/102.4) ) //<-the battery is not in saturation
6  {
7    Integral = Integral + (Error*0.030); //Delta t = 0.03 s = sample time i.e. 30 ms
8  }
9
10 ControllerOutput = ((Kp * Error) + (Ki * Integral) + Stiction);
11
12 DutyPrSpeed = 100.0/((float)batReading/102.4); // Duty cycle pr volt [% pr V]
13                                               // batReading/102.4 = volts
14 Duty = ControllerOutput * DutyPrSpeed;
15
16 if(Duty > 100) Duty = 100;  //maximum duty cycle = 100 %
17 if(Duty < 0) Duty = 0;      //minimum duty cycle =   0 %
```

Listing 8.1: Implementation of the PI-controller

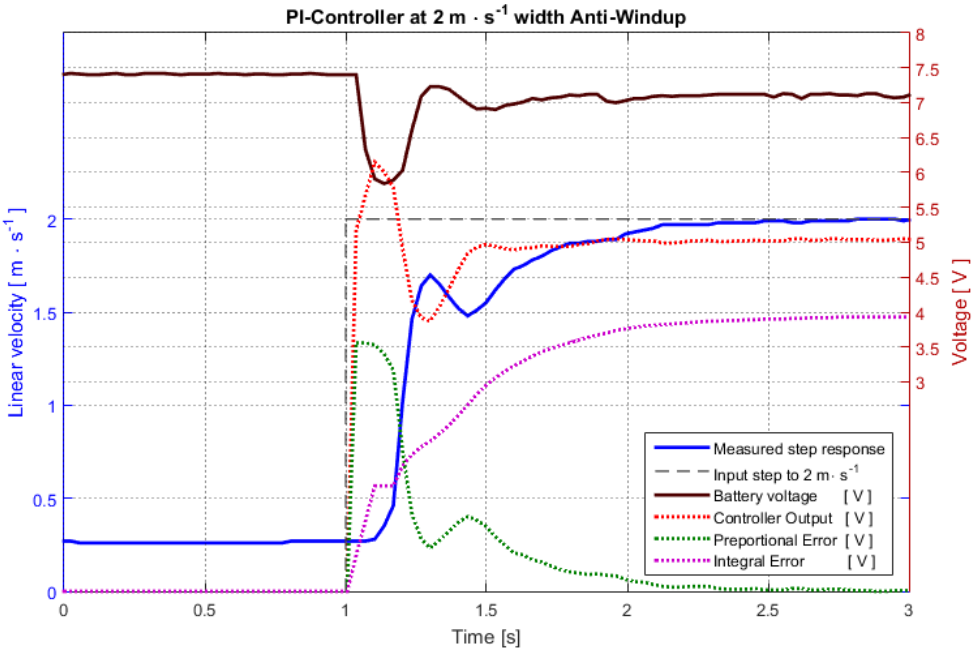


Figure 8.10: Step response of the proportional integral controller controller, showing the effect of the simple anti-windup implementation.

After making sure that the battery is not in saturation the integral error is increased or decreased depending on weather the error is positive or negative, see line 7 in Listing 8.1. Finally in line 10 the *ControllerOutput* is calculated, including the proportional gain, K_p, and the integral gain, K_i, along with the *Stiction*, which is a voltage offset to counter the effect of stiction. The duty cycle is then calculated in line 12 and 14, where it is scaled according to the current battery voltage, so that each controller output applies the calculated voltage, regardless of the battery voltage.

This final implementation with the calculated proportional and integral constants, see *Section 8.1*, is tested at the decided operating speed, $1,4\text{m} \cdot \text{s}^{-1}$, see *Figure 8.11*.

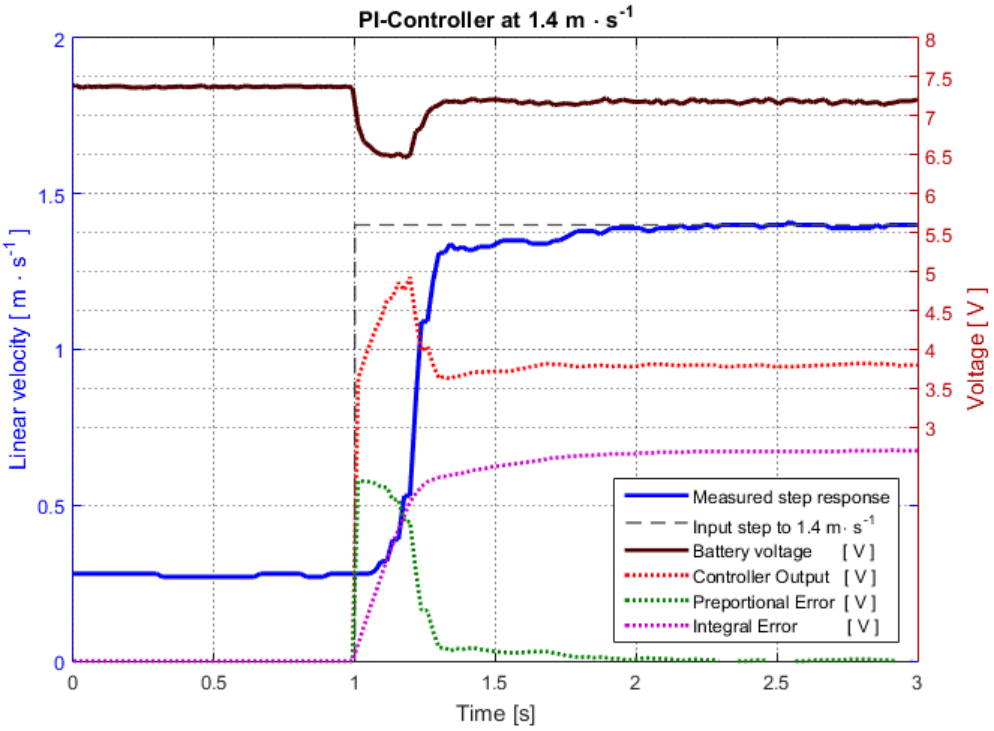


Figure 8.11: Step response of the proportional integral controller compared to simulation

The rise time is approximately 1 second and the effect of controller overshoot is minimal at decided operating velocity, $1,4\text{m} \cdot \text{s}^{-1}$. Finally a test of the PI-controller subjected to constant disturbances is carried out. The result is seen in *Figure 8.12*.

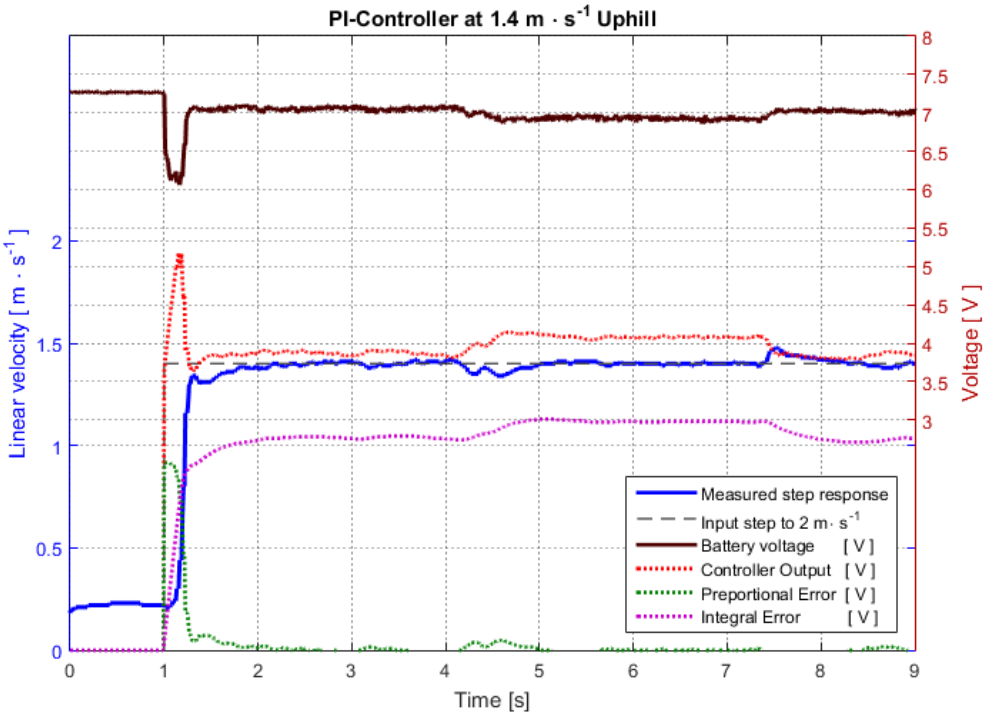


Figure 8.12: Proportional integral controller tested on a ramp

The test is carried out on a ramp, which goes up and then flattens out. The same ramp was used for testing the P-controller with feed forward, see *Figure 8.4*, where a steady state error occurred, while on the hill. Whereas the PI-controller's integral gain closes this steady state error, and so keeps the speed when on the hill. As seen from *Figure 8.4*, the adaptive capabilities of the integral gain takes care of closing in on the reference when the vehicle is subjected to disturbances.

8.2 Steering Controller

The steering controller is being split up in two controllers. One to control the angular movement of the vehicle, with the use of the servo and magnetometer, so the car can follow the right direction and another controller to keep the vehicle on the route, using the GoT system. Both controller will be design out from the steering model from *Section 7.2*.

A problem, that have occur, is that the magnetometer do not work inside, which makes testing with both the GoT system and magnetometer together impossible, as the GoT system is located indoor. This problem effects the testing of some parts of the system, which will only be simulated.

As the angular controller works on the first part of the plant, it will be design first.

Directional controller

The purpose of the directional controller is to have the vehicle turn according to a reference, which the controller receives from the route control, see *Section 2.4*. The design will start out with a proportional controller for the directional controller.

Proportional controller

A proportional controller implemented in the directional model, see *Section 7.2*, can be seen in *Figure 8.13*.

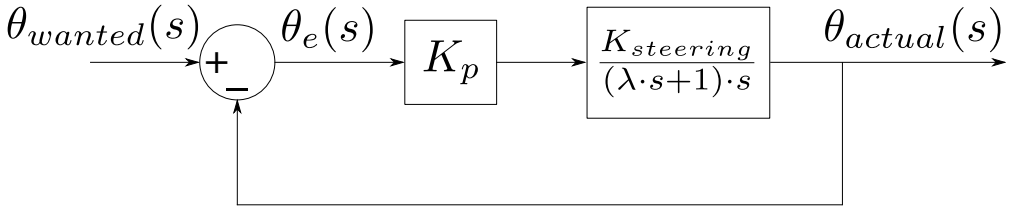


Figure 8.13: Illustration of a proportional controller for the directional steering.

This will yield the following closed loop transfer function:

$$\frac{\theta_{\text{actual}}}{\theta_{\text{wanted}}} = \frac{\frac{\text{K}_\text{p}\cdot\text{K}_\text{v}\cdot\text{K}_\text{s}}{(\lambda\cdot\text{s}+1)\cdot\text{s}}}{\frac{\text{K}_\text{p}\cdot\text{K}_\text{v}\cdot\text{K}_\text{s}}{(\lambda\cdot\text{s}+1)\cdot\text{s}}+1} \tag{8.3}$$

According to *Appendix N*, the values of K_s and K_v are not needed independently. they are therefore combined to the steering gain, $\text{K}_{\text{steering}}$. From *Appendix N*, the steering gain can be calculated, with *Equation (8.4)*.

$$\text{K}_{\text{steering}} = 0,3 \cdot \text{v} - 0,034 \tag{8.4}$$

As v is the wanted velocity, which is 1,4 $m \cdot s^{-1}$, $K_{steering}$ is equal to 0,386. And with lambda equal to the servo duty cycle time constant on 30 ms *Section 2.2*, the transfer function yields:

$$\frac{\theta_{\text{actual}}}{\theta_{\text{wanted}}} = \frac{\frac{\text{K}_\text{p}\cdot 0,386}{(0,03\cdot\text{s}+1)\cdot\text{s}}}{\frac{\text{K}_\text{p}\cdot 0,386}{(0,03\cdot\text{s}+1)\cdot\text{s}}+1} \Rightarrow \frac{1}{\frac{0,03\cdot\text{s}^2+\text{s}}{\text{K}_\text{p}\cdot 0,386}+1} \tag{8.5}$$

From the test done in *Appendix O*, it is know, that the steering gain is not constant for different K_p values. The test illustrates a linear area occurs between a K_p value of 1,5 and 3, where the steering gain is constant if the velocity is kept the same. From *Equation (8.5)*, it can be seen that the time constant will decrease, if the K_p increases. To get the fastest reacting system, with this limitation, K_p is set to be 3. This give a transfer function which yields:

$$\frac{\theta_{\text{actual}}}{\theta_{\text{wanted}}} = \frac{1}{\frac{0,03\cdot\text{s}^2+\text{s}}{3\cdot 0,386}+1} \Rightarrow \frac{1}{0,026\cdot\text{s}^2+0,86\cdot\text{s}+1} \tag{8.6}$$

From this transfer function, it can be seen that the gain is equal to one and the system is a second order system. The reason that the proportional controller yields a gain equal to 1, unlike the proportional controller for the velocity, is that there is a integrator in this system, that removes the steady state error. A proportional controller will therefore be sufficient in handling the control the vehicle's angular movement. The proportional controller is then implemented and tested. For the feedback, the magnetometer from *Section 5.2* is used. The sampling time for the magnetometer, have been chosen to be the same as the length of the duty cycle for servo, on 30 ms. As the magnetometer only needs to update one time, each time the controller sends a new duty cycle to the servo, there is not needed for a higher sampling frequency. The sampling frequency will then be on:

$$f_{sampling} = \frac{1}{30\text{ms}} \Rightarrow 33,33 \quad [\text{Hz}] \quad (8.7)$$

The angular controller is then tested, where the start heading is 5° and the reference heading is set to 45°. The data from the test is illustrated in *Figure 8.14*.

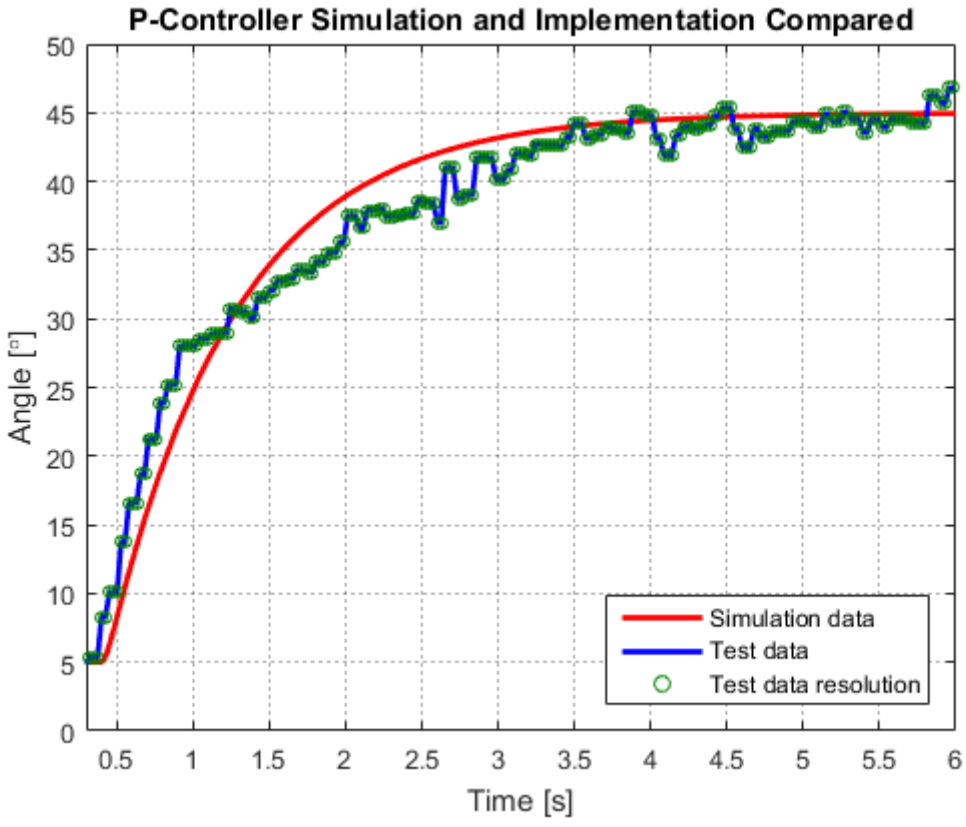


Figure 8.14: Test of the proportional controller, with a start heading on 5° and a reference on 45°.

As illustrated in *Figure 8.14*, the system looks similar as the simulation, with the same rise and settling time. The proportional controller is thereby utilized in the system, as it functions inside the area where the system is linear and is stable.

The directional controller is designed, and it is thereby possible to design the the distance controller which is around the directional controller.

Distance controller

A distance controller will now be designed, to complement the directional controller. As concluded in *Section 7.2*, a deviation from the planned line, will cause the vehicle to drive at a parallel line, beside the planned line. The deviation will be calculated from the position provided by the GoT system. As the deviation is a function of the error angle integrated over time, it is assumed that a P controller will be sufficient to handle this function, just as in the Angular controller. The controller will therefore be based on a P controller, and iterated until results are satisfactory.

As real-world test data is not available, caused by the magnetometer not working in the room with the GoT system installed, the controller will be designed and simulated in Simulink. As a starting point, the proportional gain will be set at 1, and the controller will be designed based on bode plots and step responses.

The proposed controlling scheme can be seen in *Figure 8.15*

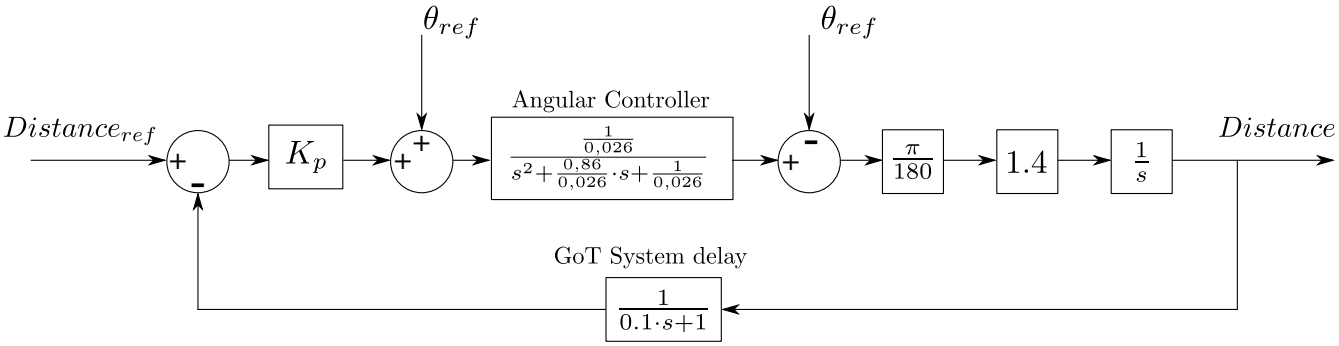


Figure 8.15: Initial Distance Controller

As illustrated in the figure, the directional controller is located in the center of the loop, as this will be responsible for changing the vehicle's angle, to what the distance loop calculates to be necessary. A reference angle, *angleRef* is added to the output of the P controller, representing the angle of the line to follow. To prevent the P controller from over-regulating, and command the system to turn, e.g. 360 degrees, it is limited by a floor and ceiling of ±90°. The following three stages (Deg to radians, Velocity and Deviation integrator) are taken directly from the steering model *Figure 7.23*.

According to *Section 2.9*, the GoT system provides position updates every 100ms. To model this delay, the same approximation as used for the servo delay in *Section 7.2*, will be implemented:

$$\frac{1}{\lambda \cdot s + 1} \Rightarrow \frac{1}{0{,}1 \cdot s + 1}$$

The input to the loop is the wanted deviation from the line. This should ideally be zero, but is set to 1m in the simulation. This is done, to avoid an initial error, when simulation the controller. The error integrator starts at zero when the simulation is started, so a input of zero would imply that the vehicle is not deviating from the route and therefore no error. Since the system is assumed linear, a step from 1m to 0m error, should be the same as the opposite and therefore can the simulation be utilized this way. The resulting step response can be seen in *Figure 8.16*

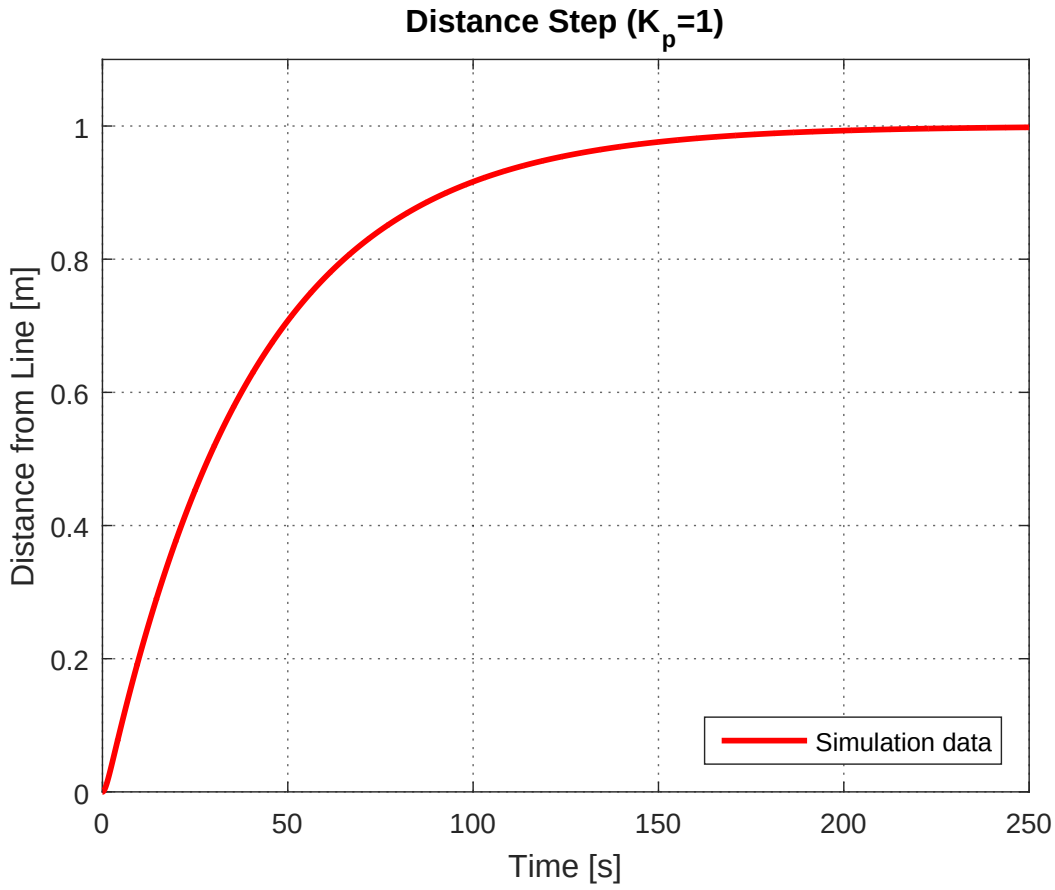


Figure 8.16: A simulated step-response of the system correcting a 1 meter offset

As seen on *Figure 8.16*, the system takes around 240 seconds, i.e. 4 minutes, to correct the error, which leaves room for improvements.

Using the Linear Analysis function of Simulink, a Bode plot is made from the open loop system. The resulting plot can be seen in *Figure 8.17*

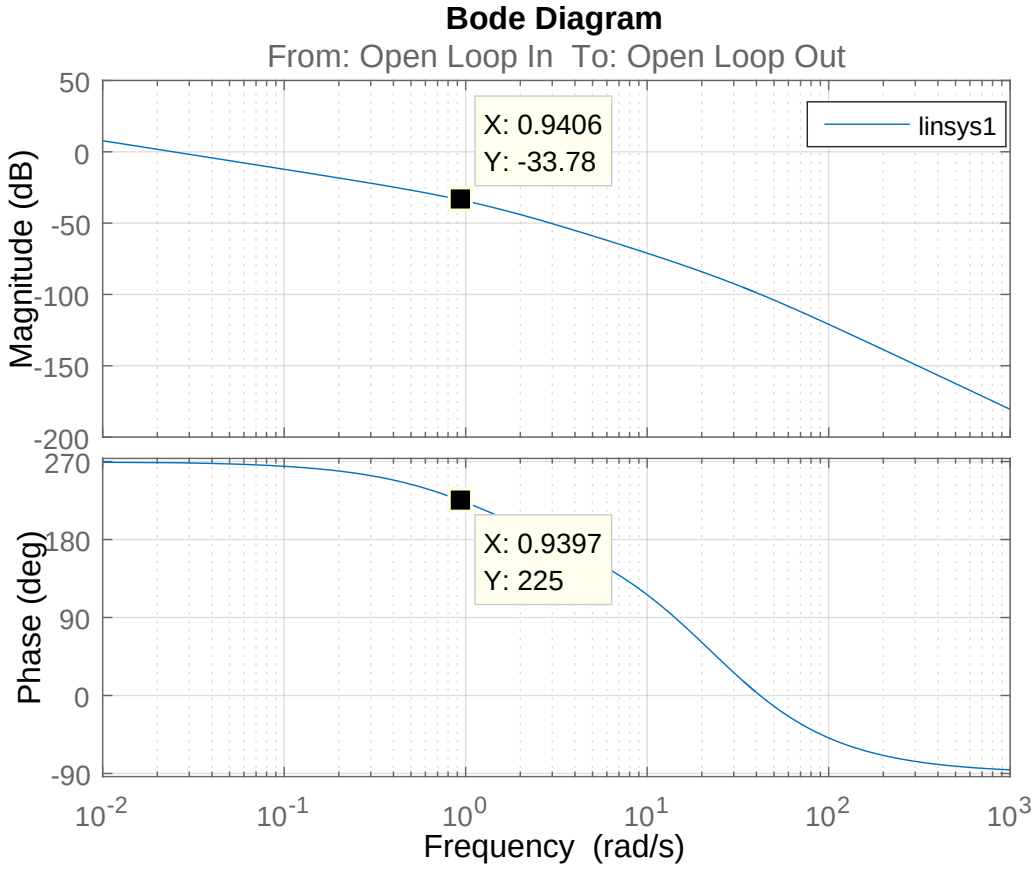


Figure 8.17: An open loop Bode plot of the system with a K_p on 1.

To improve the system response, the K_p value is changed. This is done by looking at the phase margin. As a system with phase margin on 45 degrees will give a stable system, with the highest gain, the K_p is set to give this phase margin. As seen on *Figure 8.17*, the gain is $-33{,}78\,\mathrm{dB}$ at the 45 degree point (225-45=180). To get 0 dB gain at this point, the proportional gain will be increased by $33{,}78\,\mathrm{dB}$:

$$\mathrm{K_p} = 10^{\frac{33{,}78}{20}} = 48{,}87$$

The new value is used and yields the step response illustrated in *Figure 8.18*.

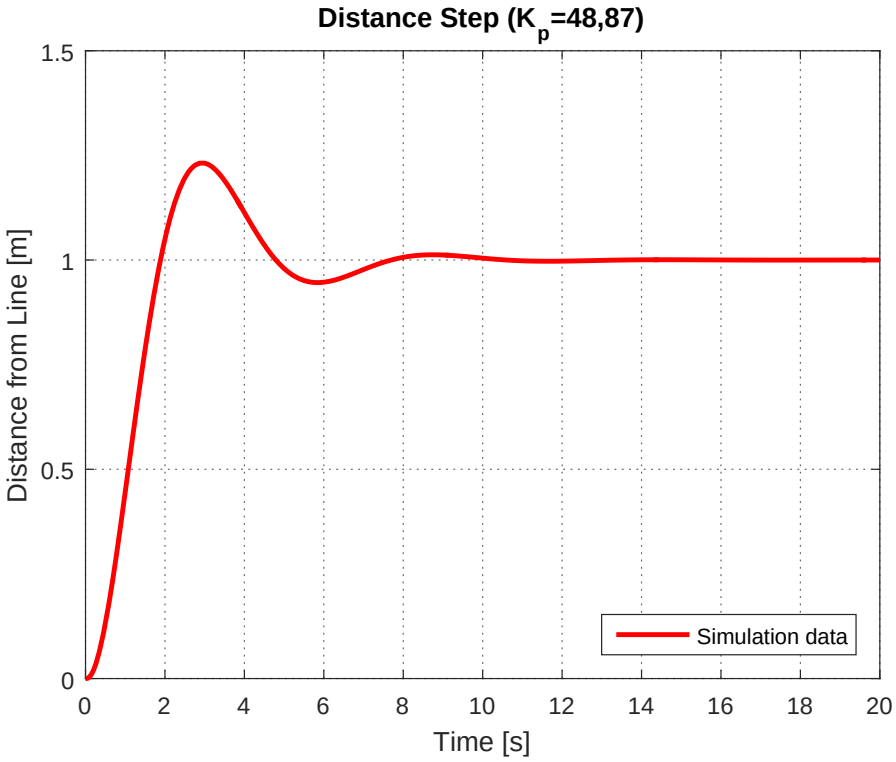


Figure 8.18: Step response with new K_p on 48,87.

As seen on *Figure 8.18*, the response has improved and the system reaches the target in approximately 2 seconds. The higher K_p gives overshoots and rings, that first settles after 10 seconds. It is improved, when compared to the response on *Figure 8.16*, but still needs improvements. If the overshoot is removed, the settling time will be around 2 seconds. A way to remove an overshoot is to increase the high frequency gain [38]. This can be done by adding a differentiator to the controller, which will then become a PD controller. According to [42], a PD controller is rarely used in real life, as it is extremely sensitive to noise, and a lead compensator is recommended instead.

$$\mathrm{K_{lead}} = \frac{1 + \mathrm{a} \cdot \mathrm{s}}{1 + \mathrm{b} \cdot \mathrm{s}} \tag{8.8}$$

A lead compensator, see *Equation (8.8)*, adds a zero to the system, which cancels out one of the poles in the system. This lowers the phase shift, improving the phase margin, and increases the gain at higher frequencies. At the same time, a pole is added at an even higher frequency, in order to ensure stability in the system.

The sampling period of the loop is limited to 100ms by the GoT system. According to the Nyquist-Shannon sampling theorem, a signal needs to be sampled by at least twice

the signal bandwidth. This limits the upper frequency of which the pole in the lead compensator can be placed. To ensure the theorem is fulfilled, the pole is given a time constant of 0,3 s. This in turn limits the frequency of the zero - if the zero is placed too close to the pole, they will cancel each other out. On the other hand, it is desired to place the zero as high as possible, in order to not change the low frequency gain, and at the same time to affect the existing poles in the system as much as possible. A compromise must be made, and the zero is given a time constant of 1 second. This yields a new controller layout, which can be seen on *Figure 8.19*.

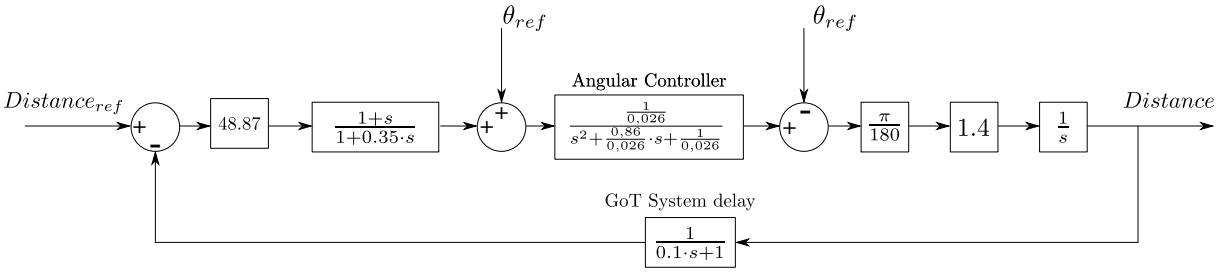


Figure 8.19: Distance controller with lead compensator

A bode plot is made with the lead compensator added to the system, shown on *Figure 8.20*.

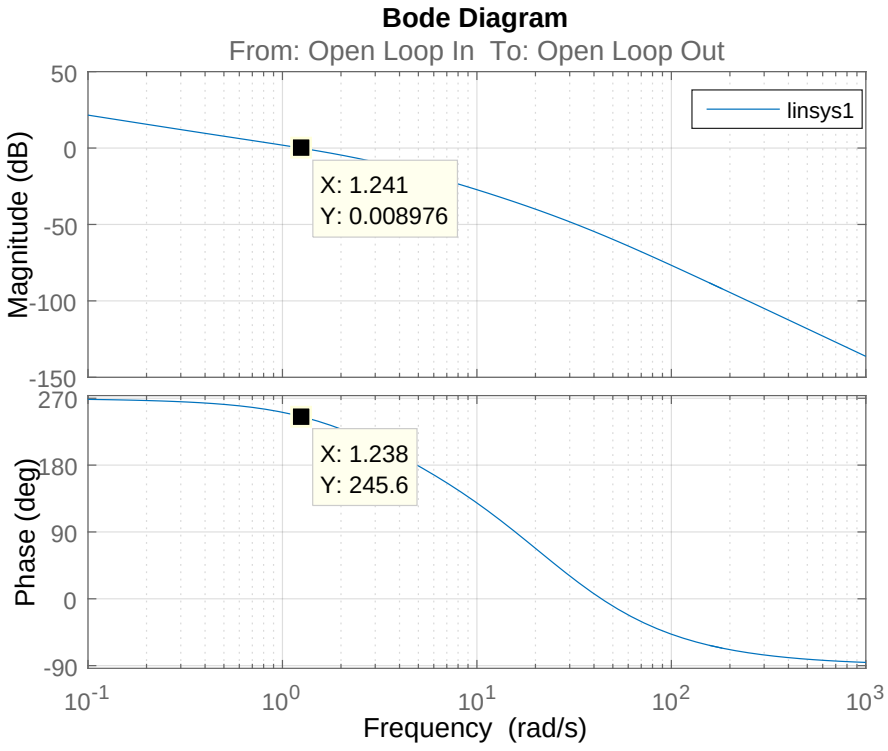


Figure 8.20: Bodeplot with lead compensation

The 0 dB frequency has increased from 0,94 $\frac{\text{rad}}{\text{s}}$ to 1,24 $\frac{\text{rad}}{\text{s}}$, and the phase margin is now:

$$245{,}6 - 180 = 65{,}6\,^\circ$$

The lead compensator has increased the gain at higher frequencies and improved the phase margin. That is made a new step response for the controller with lead compensator, shown on *Figure 8.21*.

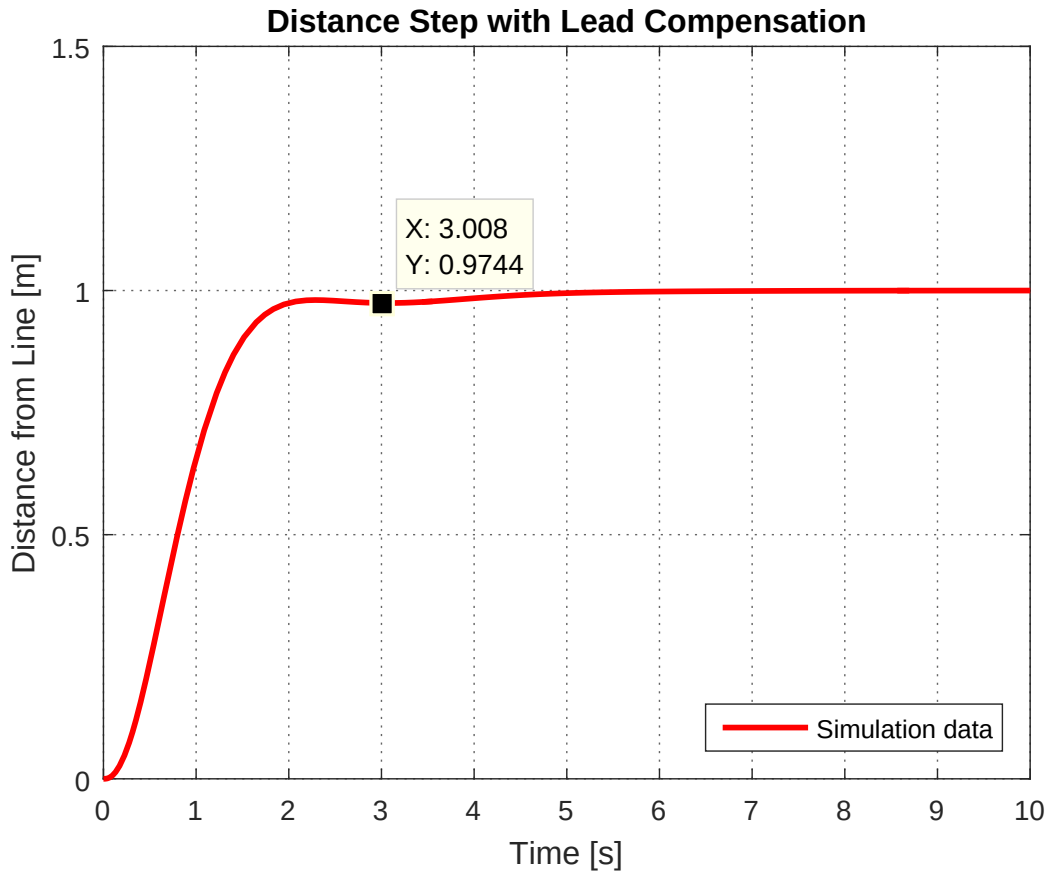


Figure 8.21: Step response with lead compensation

As seen on *Figure 8.21*, the system reaches the target in approximately two seconds, and the ringing has been reduced to less than 3 cm. This is considered acceptable and the gain is not changed, to get a phase margin on 45 degrees. It is desirable to keep the increased phase margin. As the whole loop is based on an approximated model, with no real data to verify it. The extra phase margin will provide a better chance of a stable system in the real world.

Route control

The steering controllers uses information from the route, which it have to follow. The route is made up by a number of points in the coordinate system from the GoT system, with drawn lines between, illustrated in *Figure 8.22*.

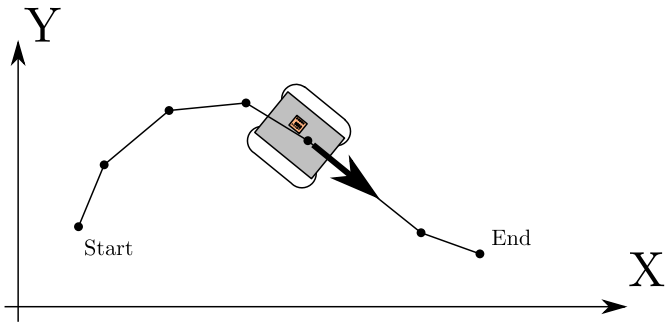


Figure 8.22: An example of a route, containing 7 points, which the vehicle have to follow.

Angle

The directional controller needs the angle of the route line, that is added to the output from the distance controller, to be the reference angle for the controller. The lines is made by connection the points from the route and therefore only the start and end point of each line is known. To calculate the angle of the line, compared to the coordinate system of the GoT system, *Equation (8.9)* is used.

$$\theta_{\text{line}} = \tan^{-1}\left(\frac{\text{Y}_{\text{end}} - \text{Y}_{\text{start}}}{\text{X}_{\text{end}} - \text{X}_{\text{start}}}\right) \cdot \frac{180}{\pi} \quad [^\circ] \quad (8.9)$$

As the output from the inverse tangent is in $rad \cdot s^{-1}$, it is multiple by $\frac{180}{\pi}$ to convert it to degrees. With this equation, zero degrees will be in the direction of the positive x axis and it will go counter clockwise for positive angles and clockwise for negative angles. This gives a range from -180° to 180°, which also is the output from the magnetometer. If compared, the angle from the coordinate system for the GoT system is not equal to the angle, that comes from the magnetometer. The reason for this is that the positive x axis is not going in the same direction as north. Therefore an offset angle is needed, which rotates the coordinate system, so zero degrees for the coordinate system and the magnetometer is the same direction.

There will be a mathematical problem, if the X coordinates is the same for both points, as there will be divided by zero. If the X values is the same, the angle will be either 90 degrees, if going toward positive y axis, or -90 degrees, if going the opposite way. As the difference can be seen on the difference in the Y value, the function will return the right numbers, even if is should not be mathematically possible to calculate it.

As the magnetometer do not work in the GoT room, this offset angle cannot be measured and therefore the right angle of the line cannot be calculated and used for indoor test.

Distance

The distance controller needs the distance from the vehicle to the route line, as it utilizes this value as a error value. The distance can be calculated with *Equation (8.10)*.

$$\text{Distance} = \frac{(a \cdot X_{now}) + (b \cdot Y_{now}) + c)}{\sqrt{a^2 + b^2}} \quad [\text{m}] \quad (8.10)$$

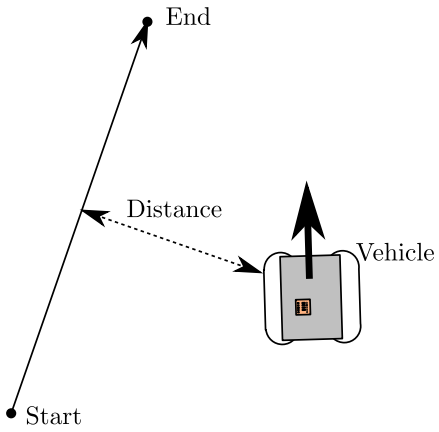


Figure 8.23: Illustration of the distance, that is calculated. The line, that comes from the shortest distance, is orthogonal on the route line.

Equation (8.10) is a standard vector equation to calculate the shortest distance from a point to a line (See *Figure 8.23*). The a, b and c values can be calculated from the start and end point, by using *Equation (8.11)*, *Equation (8.12)* and *Equation (8.13)*.

$$a = Y_{end} - Y_{start} \quad [\text{m}] \quad (8.11)$$
$$b = X_{start} - X_{end} \quad [\text{m}] \quad (8.12)$$
$$c = X_{end} \cdot Y_{start} - Y_{end} \cdot X_{start} \quad [\text{m}] \quad (8.13)$$

From *Equation (8.10)*, the distance is calculated. If the distance is negative, the vehicle is on the left side of the line and on the right side, when the distance i positive. With this equation and the measured coordinate from the GoT system, the distance the vehicle is away from the line is known and can be used as a error in the distance controller.

New line

When the vehicle reach the end point of a line on the route, a new line is used to calculate the angle and distance. The new line's start point will be the last line's end point, so the route lines is connected. Then the next end point is set and the angle can be calculated and the distance measured.

As the GoT system have a precision down to 1 mm, the coordinates too have a precision of 1 mm *Section 2.2*. The vehicle will not hit the end point with this precision, as the GoT system only delivers a measurement each 100 ms, and if the vehicle travels with 1,4 $m \cdot s^{-1}$, it will travel 140 mm for each measurement, which make the chance for hitting the end point exactly, small. Therefore is the area, which indicates the end of the route line, made bigger. The size should be around twice as big as the distance, which the vehicle travels between each measurement. With this size, there will always be two measurement in the circle, so even if one of the measurements is not good, there will be another one. This will give a circle with a radius of 28 cm

Controllers for the two models, velocity and steering, have been designed and tested. It is now possible to design a filter for the directional controller of the steering, to see if improvements on the angular sensors measurements is obtainable.

9 | Digital Filter

When utilizing a sensor, unwanted noise can arise and influence the measurements acquired. By implementing a filter, it is possible to attenuate and/or enhance specific frequency components contained in the measurements. When the magnetometer, described in *Section 4.1*, is active and the vehicle is stationary, the measured angle varies approximately two degrees, see *Figure 9.1*. The noise affecting the measurements can have a undesired effect on the steering controller. Under ideal circumstances, the magnetometer would measure an angle variation of zero degrees. Since this is not the case, implementing a filter to attenuate some of the noise is a potential solution to get more precision.

There are many pros and cons when deciding between implementing an analogue or digital filter. But since the signal received from the magnetometer is already digital, it is chosen to implement a digital filter.

Before establishing requirements for the digital filter, an analysis of the hardware and system is necessary to determine the possibilities.

9.1 Filter Considerations

In this section, the needed sampling frequency is determined, and the measurements extracted from the magnetometer and the different possibilities regarding filter type are analysed.

Sampling Frequency

It is necessary to examine if it is possible to have a high enough sampling frequency which can yield a cut-off frequency which does not affect the stability of the system. To determine this, the inner loop of the steering model, where the Magnetometer is placed, is examined.

The transfer function for the plant in the inner loop of the steering model, see *Figure 7.20*, is given as:

$$\mathrm{H(s)} = \frac{\mathrm{K}}{(0.03\mathrm{s}+1)\mathrm{s}} \tag{9.1}$$

The poles of the system yields 33.3 and 0 $\mathrm{rad} \cdot \mathrm{s}^{-1}$. When designing a filter it is necessary to consider the placement of the pole which is at the highest frequency. In this situation the pole which is at the highest frequency is the pole placed at 33.3 $\mathrm{rad} \cdot \mathrm{s}^{-1}$. A rule of thumb is to place the cut-off frequency of the filter at least one decade after the highest frequency pole, i.e. 333 $\mathrm{rad} \cdot \mathrm{s}^{-1}$. If the cut-off frequency of the filter is beneath 333 $\mathrm{rad} \cdot \mathrm{s}^{-1}$ it will influence the system's phase-margin, which can cause instability.

By being aware of this requirement, it is possible to find a sampling frequency which can grant a cut-off frequency above 333 $\text{rad} \cdot \text{s}^{-1}$. The least required cut-off frequency of 333 $\text{rad} \cdot \text{s}^{-1}$ is calculated to a frequency:

$$\omega_\text{s} = \frac{333}{2\pi} = 53 \quad \text{[Hz]} \quad (9.2)$$

To calculate the least required sampling frequency, the Nyquist-Shannon Sampling Theorem, see *Equation (9.4)*, is utilized:

$$\Omega_s \geq 2 \cdot 53 \rightarrow \Omega_s \geq 106 \quad \text{[Hz]} \quad (9.3)$$

Thus the magnetometer has to be sampled with at least 106 Hz.

In the specification for the Magnetometer [34], the maximum sampling rate is stated to be 75 Hz. A sampling rate of 106 Hz was desired, since at this rate it would not effect the system.

With this taken into consideration, it is decided to sample at 66,66 Hz, i.e. a sample every 15 ms, even though it will affect the phase margin. The 66,66 Hz has been chosen instead of the 75 Hz to have a margin from the maximum sampling rate supported by the sensor.

Frequency Analysis of Measured Data

Before establishing requirements for the filter, it is necessary to analyse the data measured by the magnetometer, to ensure that it is the noise that is attenuated and not the signal. The data analysed is logged when the vehicle and the magnetometer is stationary. It will thereby be the stationary variations on the sensor that are found. The acquired measurements are illustrated in *Figure 9.1*.

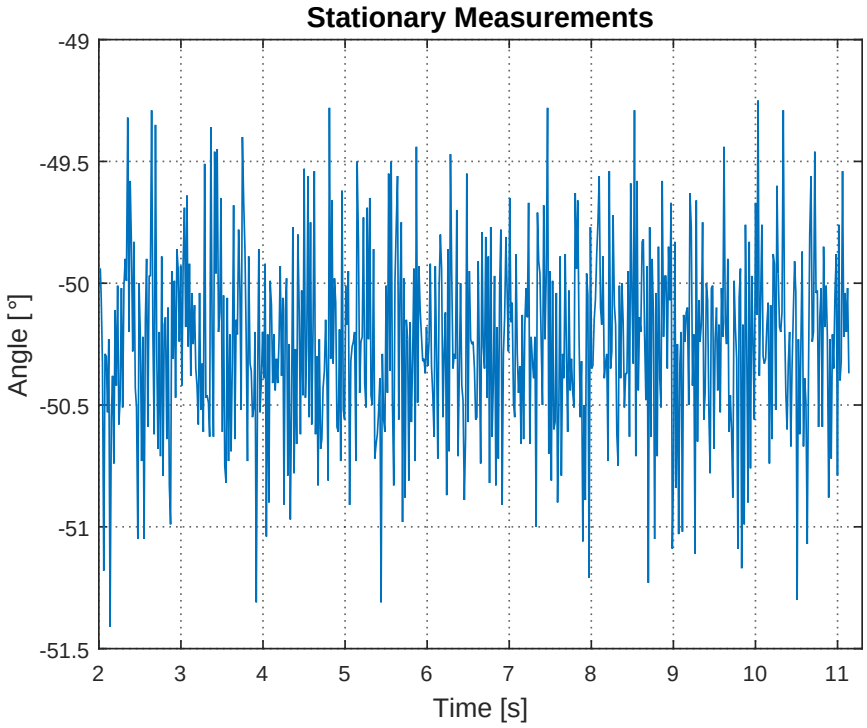


Figure 9.1: A plot of data measured with the magnetometer while the vehicle is stationary. The x-axis indicates time and the y-axis is the angle.

From *Figure 9.1*, it is seen how the angle measured vary with approximately 2 degrees. In the datasheet for the magnetometer it is explained that the sensor has a 1 to 2 degrees accuracy [34], which is consistent with the measured data. To be able to analyze measured data affected by noise, a Fast Fourier Transform, FFT, is utilized. The FFT makes it possible to see the frequency components contained in the signal, thereby making it possible to find the signal and the noise affecting the signal. An FFT of the measured data is performed and illustrated in *Figure 9.1*.

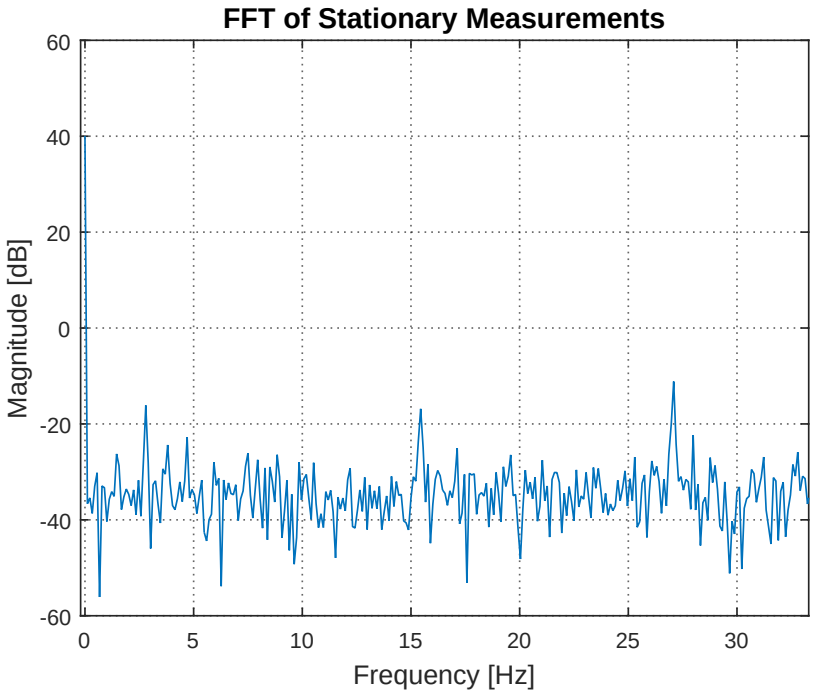


Figure 9.2: A FFT of the measured data illustrated in *Figure 9.1*

The data measured in *Figure 9.1* is acquired with a sampling frequency of 66.6 Hz. The Nyquist-Shannon Sampling Theorem says that to find the frequency components contained in a signal, it is necessary to have at least twice the sampling frequency [43]:

$$\Omega_s \geq 2 \cdot \Omega_N \quad [\text{Hz}] \quad (9.4)$$

Where:

Ω_s	is the sampling frequency	[Hz]
Ω_N	is the Nyquist frequency	[Hz]

This is illustrated in *Figure 9.2*, where the frequency components only goes from 0 to 33.3 Hz on the x-axis, which is half the sampling frequency. The y-axis is the magnitude of the frequency components in the signal.

From the FFT performed, as seen in *Figure 9.2*, a spike is present at 0 Hz. This is the DC value, i.e. the offset seen in *Figure 9.1*. This frequency component is the signal which the filter should not attenuate. The frequency components present after 0 Hz, is the noise from the sensor. The noise is determined to be white noise, as it looks evenly distributed over the frequency range.

Some considerations have to be made for the filter requirements, to ensure that the filter is not influencing the system and the desired frequency needlessly.

Filter Type

Data from the filter has been examined, it is thereby possible to examine which filter would be suitable for fulfilling the predetermined requirements. It should do this without unnecessarily influencing the DC-component and the system.

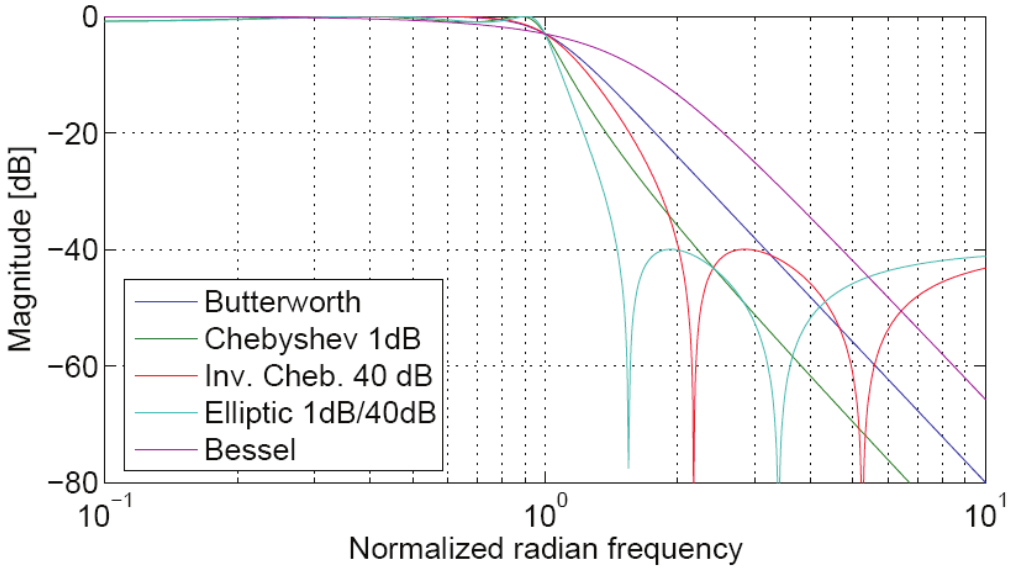


Figure 9.3: Frequency response for various filter types [44]

The disadvantages of a Elliptic filter are that it has both ripples in the pass- and stopband. additionally, it has a high group delay both before and at the cutoff frequency, this can be seen in *Figure 9.4*. The advantage of a Elliptic filter is that is has the sharpest cut-off frequency of the filters illustrated in *Figure 9.3*.

The Chebyshev filter only has ripples in the passband, and a sharp cut-off frequency, but still has a high group delay before and at the cut-off frequency. The inverse Chebyshev only has ripples in the stopband, but does not have as sharp cut-off frequency as the Chebyshev and the Elliptic filter. Compared to the two former filters it has a lot less group delay before and after the cut-off frequency.

The Bessel filter has the lowest group delay before and at the cut-off frequency, see *Figure 9.4*. Additionally, it does not have any ripples, neither in pass- or stopband. The disadvantage of the filter is that it has the least sharp cut-off frequency of the filters illustrated in *Figure 9.3*.

The Butterworth filter has a sharper cut-off frequency compared to the Bessel filter and as the Bessel filter it does not have any ripples neither in the pass- nor the stopband. Furthermore, the Butterworth filter has the second lowest group delay at the cut-off frequency compared to the other filters illustrated in *Figure 9.4*.

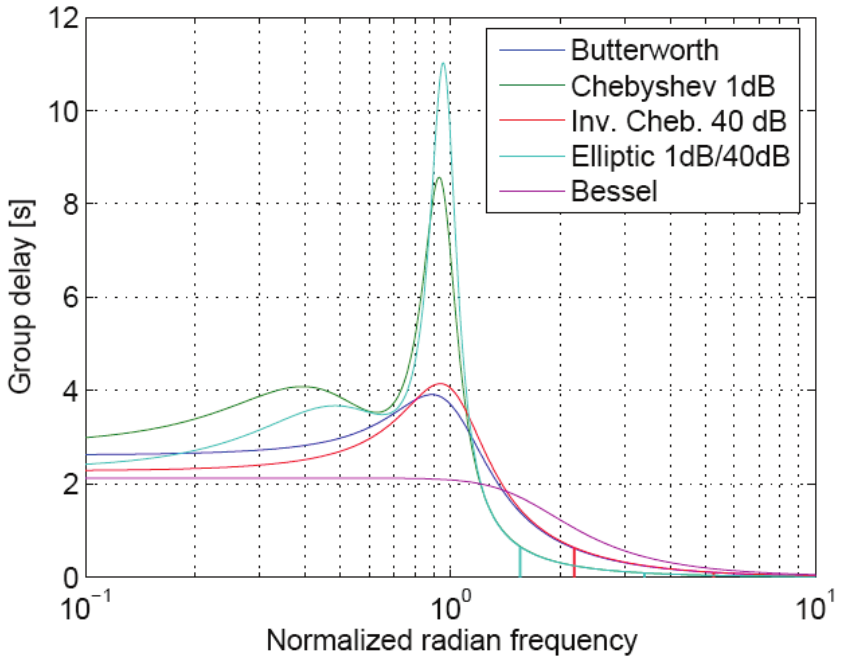


Figure 9.4: Frequency response illustrating group delay for various filter types [44]

Because of the above-mentioned descriptions of the filters, a Butterworth filter is selected for filtering the measured data. The Butterworth does not have any ripples in the pass-band which could influence the DC-component, seen in *Figure 9.2*, and compared to the other described filters, the Butterworth has a small group delay.

9.2 Filter Requirements

From the filter considerations *Section 9.1*, it is possible to establish the specifications for the filter.

It has been decided to design a filter with a cut-off frequency which is less than one decade above the highest frequency pole. This will affect the system and cause a delay. To ensure the delay does not cause instability the filter first needs to be designed and simulated. Furthermore, since it is desired to influence the DC-signal as little as possible, see *Figure 9.2*, a low-pass filter is implemented.

By the use of an iterative process frequency requirements for the pass - and stopband is established. A passband frequency of 19 Hz is chosen. The ideal filter would have a passband attenuation of 0 dB. Since this is not an ideal filter, a maximum attenuation variation is set from 1 dB to 0 dB. The stopband frequency has been chosen to be at 29 Hz with an attenuation of 40 dB.

Since the requirements needs to be adaptable to the z-plane, the Nyquist-Shannon sampling theorem [43], is utilized:

$$\Omega_N = \frac{2\pi \cdot f_s}{2} \qquad \left[\text{sycles} \cdot \text{s}^{-1}\right] \quad (9.5)$$

The passband edge frequency, 19 Hz, and the stopband edge frequency, 29 Hz, are normalized by the Nyquist sampling frequency, in order to get the requirements in discrete time, ie. cycles per. sample:

$$\frac{19 \cdot 2\pi}{66,6} = 0.5705\pi \quad \wedge \quad \frac{29 \cdot 2\pi}{66,6} = 0.8709\pi \qquad \left[\text{cycles} \cdot \text{samples}^{-1}\right] \quad (9.6)$$

Summary of Filter Specifications:

- **Filtertype**

  A Butterworth lowpass filter is designed.

- **Passband specifications**

  $0,08912 \leq |H(e^{j\omega})| \leq 1$
  $0 \leq |\omega| \leq 0,5705\pi$

- **Stopband specifications**

  $|H(e^{j\omega})| \leq 0,01$
  $0,8709\pi \leq |\omega| \leq \pi$

The specifications for the filter is established, and it is thereby possible to design the filter.

9.3 Design

Various methods can be utilized when designing a digital filter. In the following subsection two methods for transferring a continuous-time filter to the z-domain is examined. The specific design process of the Butterworth low-pass filter will be done by following steps retrieved from "Discrete-Time Signal Processing"[43].

Bilinear Transform vs. Impulse Invariance transformation

There are different methods for transferring a continuous-time filter to a discrete-time filter, the two most common transformation methods are examined. The first method of the two is the impulse invariance transformation.

When utilizing the impulse invariance transformation a discrete filter is generated by sampling an impulse response of a continuous analogue prototype filter[43].

This makes a linear mapping, $\omega = \Omega \cdot \mathrm{T_d}$ for each pole in the analogue filter's transfer function on the s-plane to a pole on the z-plane.

The main issue when utilizing the impulse variance method is that the sampling rate needs to be relatively high compared to the filter's bandwidth, to not cause aliasing [45]. In the frequency response, seen in *Figure 9.5*, it can be seen that the response of a 6th-order Butterworth filter transformed by impulse invariance, has a frequency response which is larger than from 0 to pi, hence causing aliasing[43].

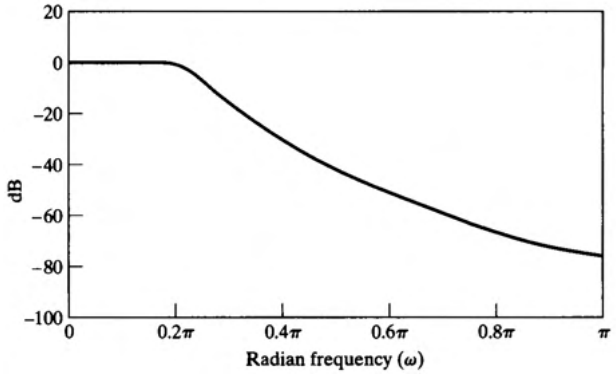


Figure 9.5: A frequency response of a transformed impulse variance 6th order Butterworth filter [43].

The Bilinear transform also utilize an analogue prototype filter to convert from s-domain to z-domain, when designing a digital filter. This transformation type is not a linear transformation from the s-domain to the z-domain, as it is with the impulse variance method. Instead it is a non-linear transformation which is mapping poles from $-\infty < \Omega < \infty$ in the s-domain to $-\pi < \omega < \pi$ in the z-domain [43]. This yields the following relationship between the continuous-time frequencies to the discrete-time frequencies:

$$\omega = 2 \cdot \arctan(\frac{\Omega \cdot T_d}{2}) \wedge \Omega = \frac{2}{T_d} \cdot \tan(\frac{\omega}{2}) \tag{9.7}$$

A frequency response, of the same filter as the frequency response for the impulse invariance, is illustrated in *Figure 9.7*. The bilinear transform avoids aliasing problems because it maps the entire s-planes imaginary axis onto the z-planes unit circle [43]. The s-plane and z-plane can be seen in *Figure 9.6*.

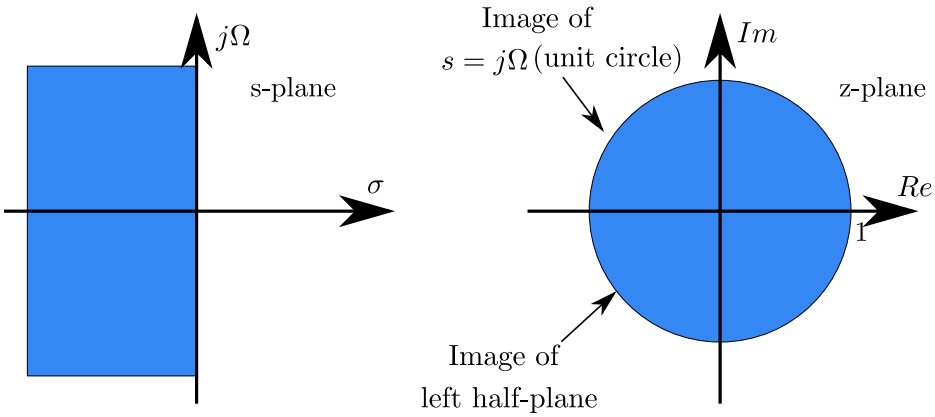


Figure 9.6: s-domain and z-domain [43].

From *Figure 9.7*, it can be seen that the frequency response does not exceed the limit from 0 to π, hence not causing a aliasing problem [43].

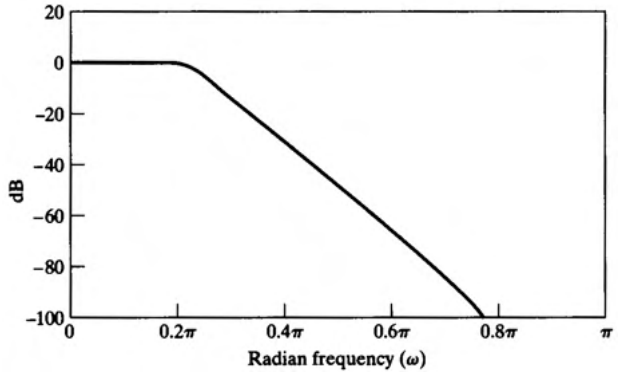


Figure 9.7: A frequency response of a transformed bilinear transform 6th order Butterworth filter [43].

It has been chosen to use the bilinear transformation when designing the filter. This is mainly due to the issue with aliasing that occurs when utilizing the impulse variance transform, which is avoided when utilizing the bilinear transform.

Designing the Filter by the use of Bilinear Transform

When utilizing bilinear transform it is necessary to frequency warp the passband frequency and the stopband frequency, from *Section 9.2*. This is done by utilizing the relationship between the discrete-time frequencies in the z-domain to the continuous-time frequencies in the s-domain illustrated in *Equation (9.7)*.

$$\Omega_p = \frac{2}{0.015} \cdot \tan(\frac{0.5705\pi}{2}) = 166 \quad \wedge \quad \Omega_s = \frac{2}{0.015} \cdot \tan(\frac{0.8709\pi}{2}) = 644 \qquad [\text{Hz}] \quad (9.8)$$

Where:

Ω_p is the passband frequency in continuous time [Hz]

Ω_s is the stopband frequency in continuous time [Hz]

By utilizing the magnitude squared function for a Butterworth filter, it is possible to find the cut-off frequency:

$$|H(e^{j\omega})|^2 = \frac{1}{1 + (\frac{\Omega}{\Omega_c})^{2N}} \tag{9.9}$$

Where:

Ω	is the pre-warped frequencies	[Hz]
Ω_c	is the cut-off frequency	[Hz]
$\lvert H(e^{j\omega}\rvert$	is the attenuation requirements set for the bands from *Section 9.2*	[dB]

Since there are two unknown variables, the cut-off frequency and the order N, it is possible to calculate the values of these by utilizing the two equations with two unknowns:

$$0.89125^2 = \frac{1}{1 + (\frac{166}{\Omega_C})^{2N}} \quad \wedge \quad 0.01^2 = \frac{1}{1 + (\frac{644}{\Omega_C})^{2N}} \tag{9.10}$$

The cut-off frequency is calculated to be **197.9 Hz** and the order to be 3.9, rounded to the nearest integer, i.e. **4**. When the order is known it is possible to utilize the following equation for finding the poles for a Butterworth filter.

$$P_k = \Omega_c \cdot e^{j(\frac{2 \cdot k - 1}{2 \cdot N} \cdot \pi + \frac{\pi}{2})} \tag{9.11}$$

Where k equal 1,2 ... N.

Since it would be desirable to have a stable system, the poles should only be located on the left side of the imaginary-axis.

$$P_1 = \Omega_c \cdot e^{j \cdot \frac{5}{8}\pi} \tag{9.12}$$

$$P_2 = \Omega_c \cdot e^{j \cdot \frac{7}{8}\pi} \tag{9.13}$$

$$P_3 = \Omega_c \cdot e^{j \cdot \frac{9}{8}\pi} \tag{9.14}$$

$$P_4 = \Omega_c \cdot e^{j \cdot \frac{11}{8}\pi} \tag{9.15}$$

The poles' placement indicated that it is two complex pole-pair, since they are symmetric around the real-axis.

The general transfer function for a Butterworth filter is defined as:

$$H(s) = \frac{G_o}{\prod_{k=1}^{N} (s - P_k)} \tag{9.16}$$

Where:

G_o is the gain of the filter [·] The transfer function is rewritten to contain the located poles.

$$H(s) = \frac{G_o}{(s - \Omega_c e^{j \cdot \frac{5}{8} \cdot \pi})(s - \Omega_c e^{j \cdot \frac{7}{8} \cdot \pi})(s - \Omega_c e^{j \cdot \frac{9}{8} \cdot \pi})(s - \Omega_c e^{j \cdot \frac{11}{8} \cdot \pi})} \tag{9.17}$$

Seen from the placement of the poles, and by the use of the rule $s = s*$, it is possible rewrite the transfer function.

$$H(s) = \frac{G_o}{(s - \Omega_c e^{j\cdot\frac{5}{8}\cdot\pi})(s - \Omega_c e^{j\cdot\frac{7}{8}\cdot\pi})(s - \Omega_c e^{-j\cdot\frac{7}{8}\cdot\pi})(s - \Omega_c e^{-j\cdot\frac{5}{8}\cdot\pi})} \tag{9.18}$$

The transfer function is rewritten to contain the complex poles in standard form:

$$H(s) = \frac{G_o}{(s^2 + 2\cdot\cos(\frac{7}{8}\cdot\pi)\cdot\Omega_c\cdot s + \Omega_c^2)(s^2 + 2\cdot\cos(\frac{5}{8}\cdot\pi)\cdot\Omega_c\cdot s + \Omega_c^2)} \tag{9.19}$$

The DC gain, G_o is determined by setting s to 0. furthermore, since you have 0 dB at DC which corresponds to 1 in amplitude, then H(0) is equal to 1. The DC gain will thereby be:

$$G_o = \Omega_c^4 \tag{9.20}$$

All the coefficients have been found for the transfer function, thus making it possible to perform a frequency response illustrated in *Figure 9.8*. The continuous transfer function will thereby be:

$$H(s) = \frac{\Omega_c^4}{(s^2 + 2\cdot\cos(\frac{7}{8}\cdot\pi)\cdot\Omega_c\cdot s + \Omega_c^2)(s^2 + 2\cdot\cos(\frac{5}{8}\cdot\pi)\cdot\Omega_c\cdot s + \Omega_c^2)} \tag{9.21}$$

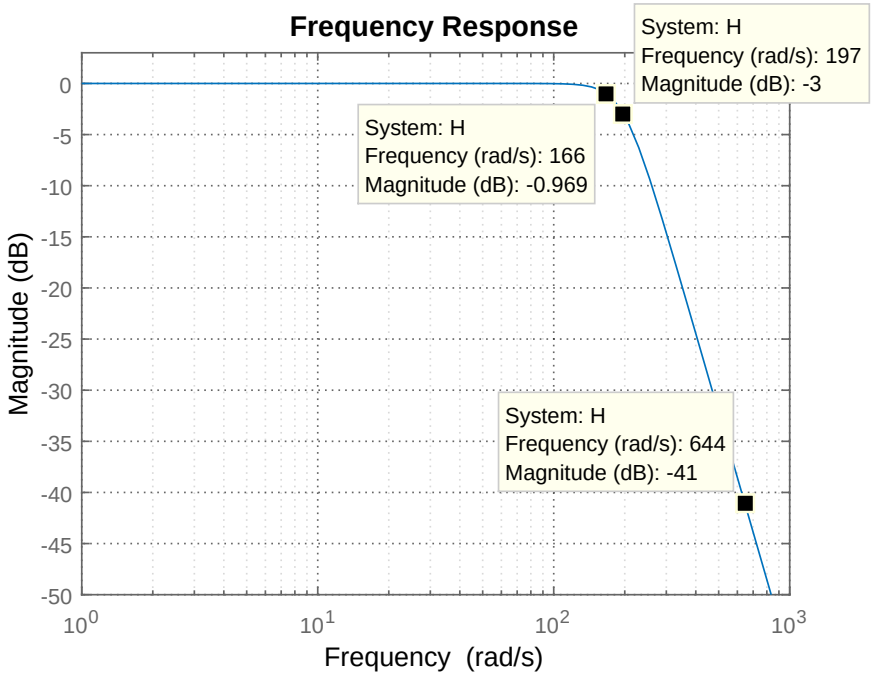


Figure 9.8: Bode plot of the continuous filter

The discrete specifications pre-warped to continuous-time, see *Equation (9.8)*, are determined for the passband frequency to 166 Hz and for the stopband to 644. Furthermore from the requirements set in *Section 9.2*, the attenuation in the passband should be beneath 1 dB and the stopband should be at a attenuation of 40 dB. This requirements is in compliance with what is observed on *Figure 9.8*. Additionally, the cut-off frequency calculated in *Equation (9.10)* corresponds with what is illustrated in the frequency response.

The established continuous-time filter can now be transferred to the z-domain.

Transforming the filter to the discrete-time domain

The bilinear transform is utilized to transform the continuous-time filter to the z-domain. Substituting the expression for s, given in *Equation (9.22)*, in the continuous-time transfer function, in *Equation (9.21)*, a discrete-time transfer function for the filter is determined.

$$s = \frac{2}{T_d}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) \tag{9.22}$$

A frequency response of the discrete-time transfer function is illustrated in *Figure 9.9*.

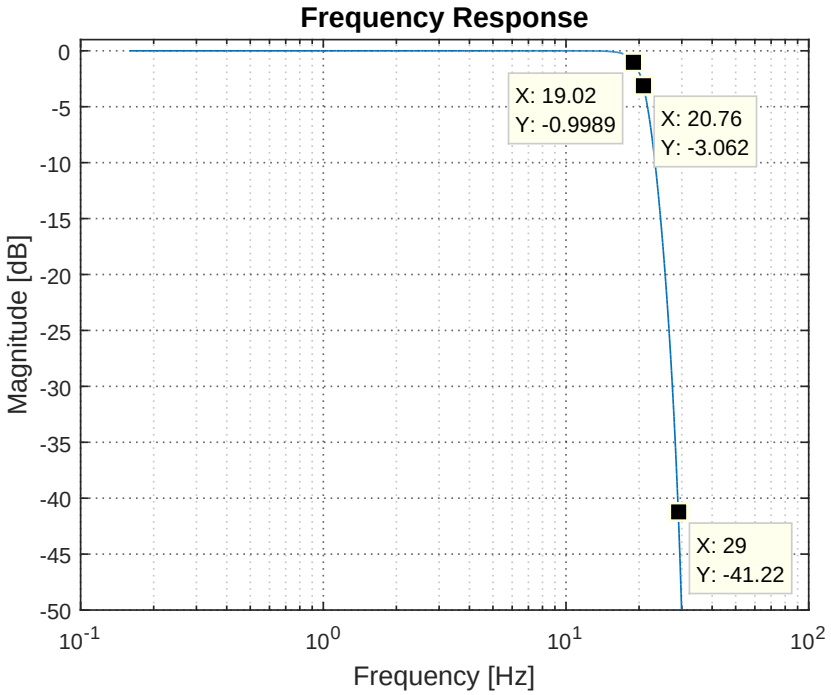


Figure 9.9: Bode plot of the discrete time filter

The requirements set for the filter in *Section 9.2* are in compliance with the frequency response plotted in *Figure 9.9*. The passband frequency should be at 19 Hz and the

stopband at 29 Hz. Additionally, the attenuation in the passband within the range from 0 to 1 dB and the stopband attenuation of 40 dB.

By utilizing normalized expansion it is possible to rewrite the discrete-time transfer function to a standard form, defined as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{n=0}^{M} b_n z^{-n}}{\sum_{n=0}^{N} a_n z^{-n}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}} \tag{9.23}$$

The coefficients, a_N and b_M, can be used directly when implementing the filter. The discrete transfer function is rewritten to:

$$H(z) = \frac{B(z)}{A(z)} = \frac{0.1882 + 0.7527z^{-1} + 1.129z^{-2} + 0.7527z^{-3} + 0.1882z^{-4}}{1 + 0.9595z^{-1} + 0.7785z^{-2} + 0.2358z^{-3} + 0.03691z^{-4}} \tag{9.24}$$

9.4 Implementation

There is different methods to implement an IIR filter. The coefficients from *Equation (9.24)*, can either be implemented directly by utilizing the structure called Direct form I or implemented with the structure Direct form II which has less delay. The fourth order low-pass Butterworth IIR filter in direct form II is illustrated in *Figure 9.10*.

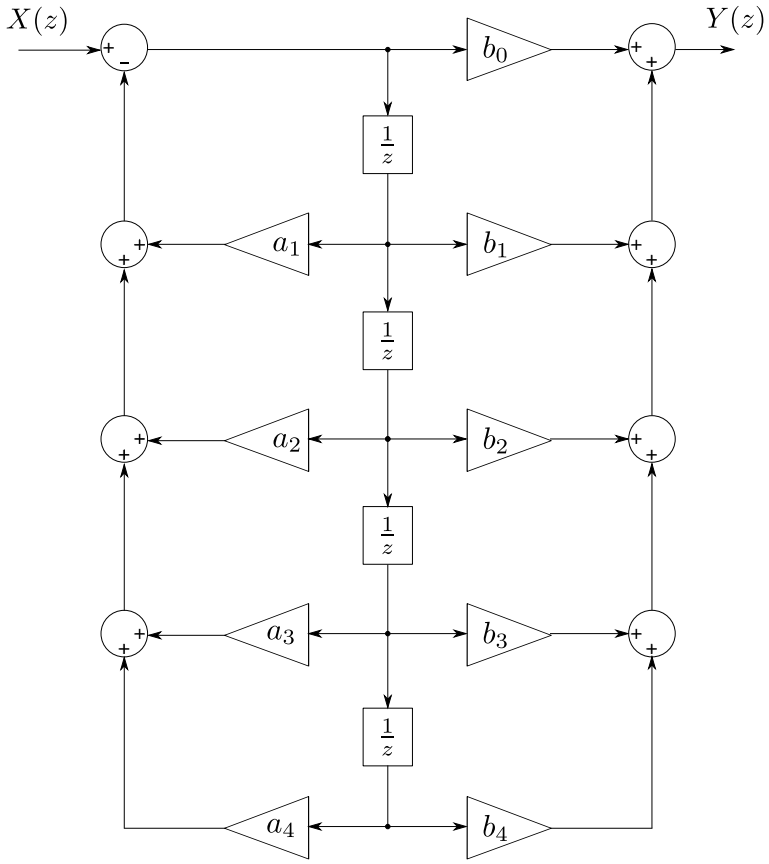


Figure 9.10: fourth order low-pass Butterworth IIR filter implemented with the structure Direct form II

By directly utilizing the structure and patterns of *Figure 9.10*, it is possible to implement the filter almost directly in Matlab. The implementation of the related code can be seen in Listing 9.1.

```
for i = 1:707 %Quantity of samples

  Buffer0 = Input(i) - (a1*Buffer1 + a2*Buffer2 + a3*Buffer3 + a4*Buffer4)
  %Calculated value of the first Buffer in Direct form II

  Output(i) = Buffer0*b0 + b1*Buffer1 + b2*Buffer2 + b3*Buffer3 + b4*Buffer4
  %The generated output of the filter

  Buffer4 = Buffer3 %A delay generated by shifting the buffers
  Buffer3 = Buffer2
  Buffer2 = Buffer1
  Buffer1 = Buffer0

 end
```

Listing 9.1: Butterworth low-pass IIR filter Implementation in Matlab by the utilizing the Direct form II structure

When implementing the code directly on a microcontroller a for-loop will not be utilized, as each sample from the Magnetometer is processed immediately, thus generating a filtered measurement for the control loop to utilize. Buffer0 is located just above the first delay, $\frac{1}{z}$, in *Figure 9.10*, Buffer1 beneath the first delay etc.. Buffer0 contains the Input, the measurement delivered by the magnetometer, substracted with the a_N coefficients from *Equation (9.24)* multiplied with the related buffers.

Buffer0 now contains the needed information of the first part of the structure and can be utilized to transfer this value to be part of the output. The output of the filter, i.e. the filtered measurement, is composed of the b_M coefficients and the related buffers. The delay needed is generated by shifting the value of the buffers.

9.5 Results

The filter is implemented in Matlab, and it is thereby possible to filter the measurements received from the Magnetometer, see *Figure 9.1*, to see if the filter can decrease the sensors variations of approximately 2 degrees. The original measurements and the related filtered measurements are illustrated in *Figure 9.11*.

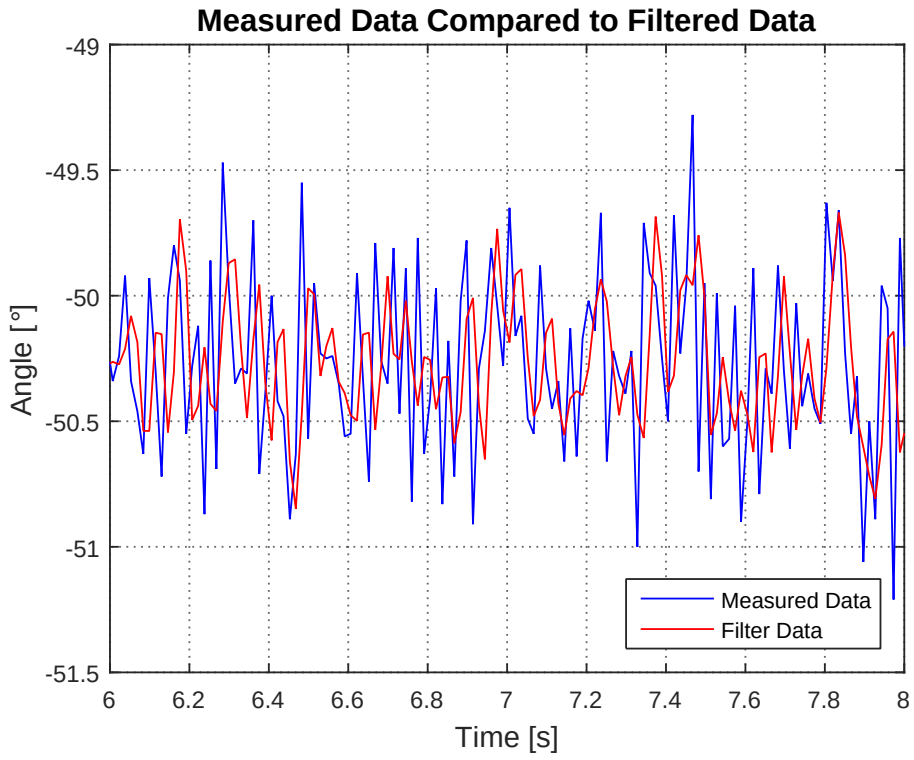


Figure 9.11: The original measurements from *Figure 9.1* compared to the related measurements filtered utilizing the implemented filter in Listing 9.1

In *Figure 9.11* the red line is the filtered data and the blue line is the raw data measured from the Magnetometer. It can be seen from the figure that the signal is still noisy, but the filter reduces the peak variations of the raw measured data, making the overall noise amplitude lower. A delay is observed, caused by the group delay in the filter.

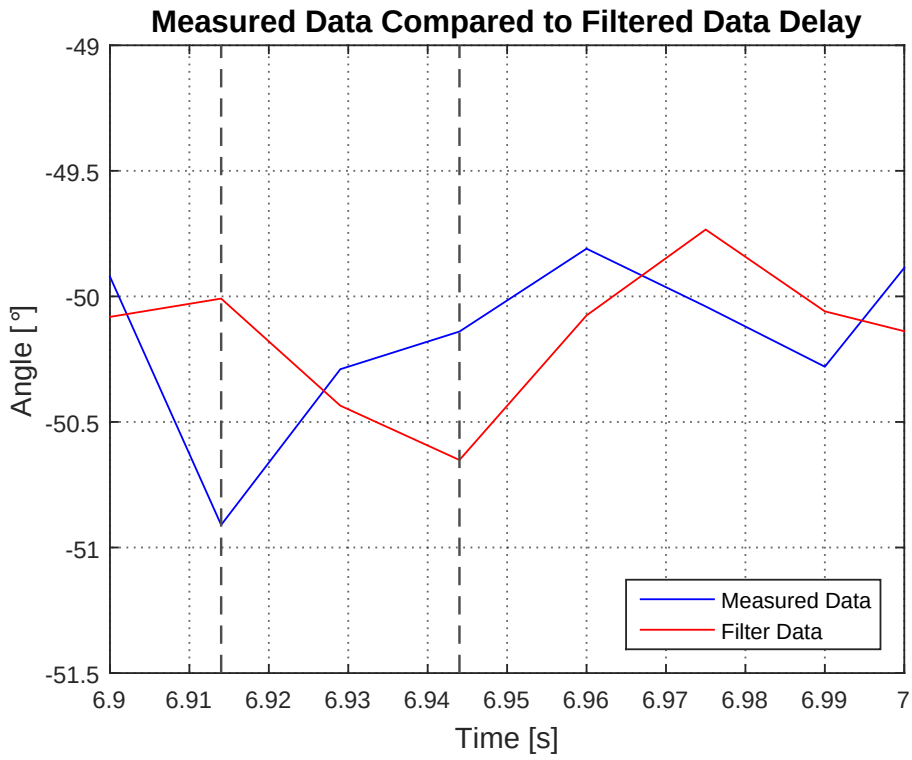


Figure 9.12: A plot illustrating the delay occurring when the filter is implemented compared to the raw measured data received directly from the Magnetometer.

As seen on the figure a delay of approximately 0.03 s occurs when utilizing the filter compared to the raw measured data. If the velocity of the vehicle is 1.4 $\mathrm{m} \cdot \mathrm{s}^{-1}$ then the vehicle will have moved approximately 4.2 cm in this time. By simulating a step response of the directional control loop with the filter, it is possible to see if the filter will make the directional control unstable. A simulation of the a step response in the directional control loop with and without filter is illustrated in *Figure 9.13*.

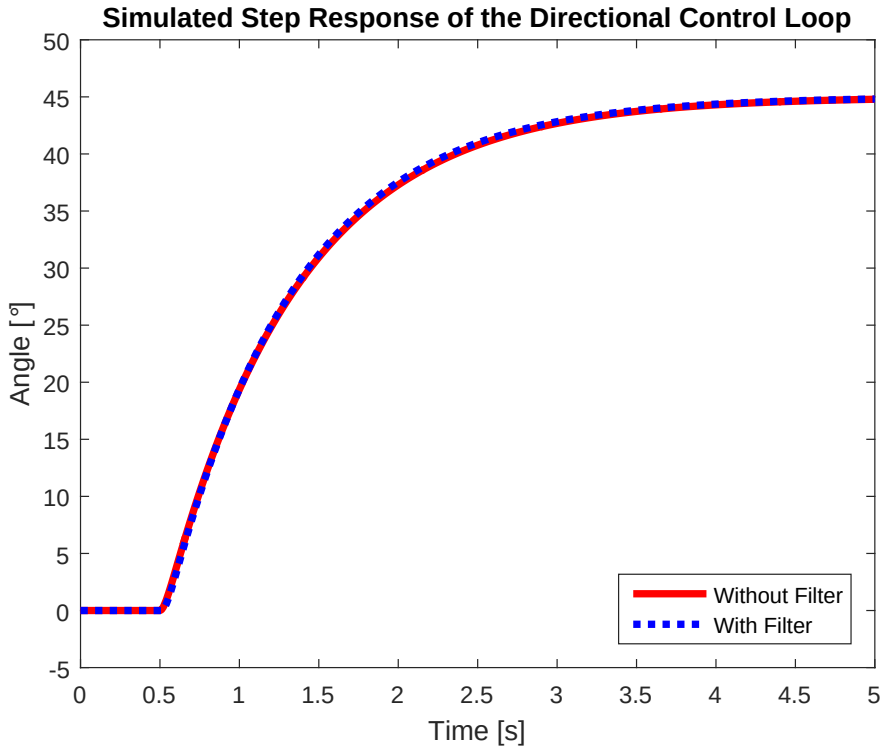


Figure 9.13: A plot illustrating a simulated step response of the directional control loop with and without [illegible]

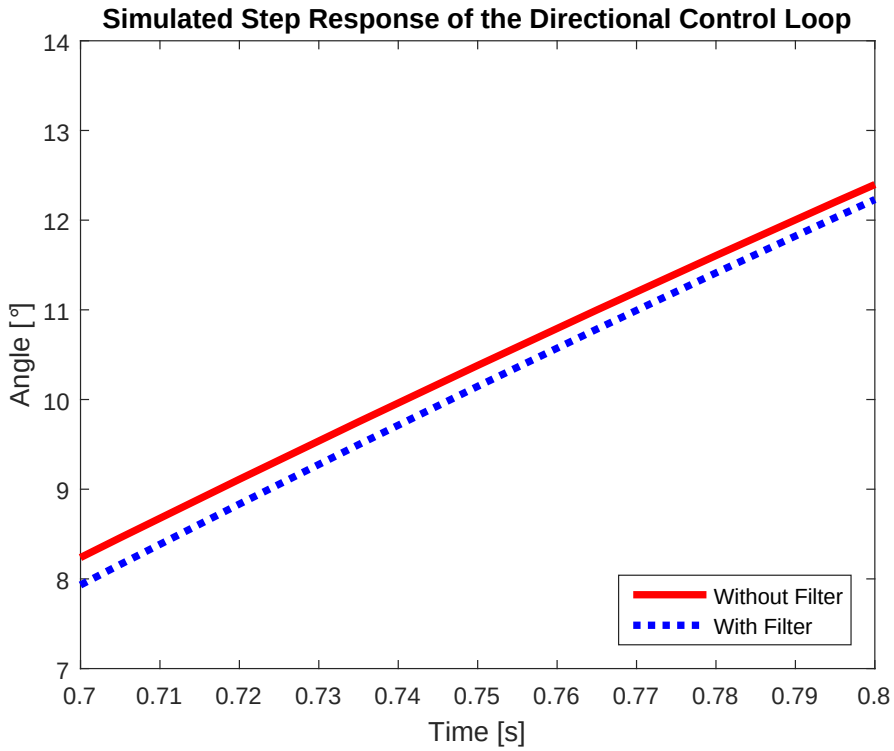


Figure 9.14: A zoomed in view.

From *Figure 9.13* it can be seen that the filter does not change the step response significantly. Only when zoomed in (*Figure 9.14*), a small delay can be seen in the response.

A bode plot is now made, with and without the filter, see *Figure 9.15*.

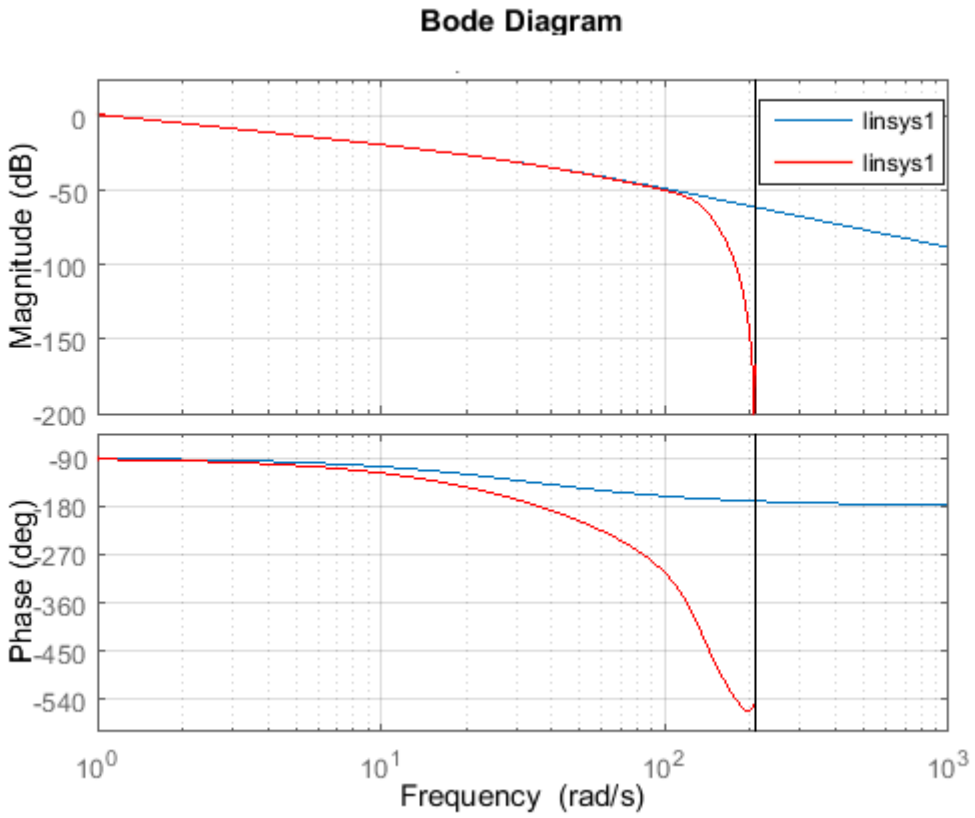


Figure 9.15: Open loop Bode plot of the angular controller, with(red) and without(blue) the IIR filter

As seen on the bode plot, the filter does not affect the magnitude response of the system until around 100 rad/s. The phase remains close until around 10 rad/s, and then starts to deviate. This frequency is well above the dominant pole, which explains why the step response remains nearly unchanged. As the filter introduces 4 extra poles, the total phase shift is expected to be $4 \cdot 90° = 360°$ larger, which is confirmed. As the loop has no gain, stability is ensured.

While the filter has been verified to be functional, without affecting the stability of the system, the results are not convincing. While the bode plots looks nice, and the largest peaks of the noise have been removed, the noise looks close to unchanged (*Figure 9.11*). As the microcontroller has no floating point calculation hardware, the filter will be expensive in processing time, with very little improvements in return. Considering this, the filter will not be implemented.

Part III

Test & Conclusion

10 | Acceptance Test

The system has been described and explained through the prototype and the modelling, to be able to understand the functionalities needed. The controllers have been implemented for the functions needed, according to the sensors used. For the last part of the project, tests of the different parts of the system and the system as a whole, will ensure that the system can handle the requirements determined in *Section 3*.

10.1 Test Procedure

Req. no.	Requirements	Test procedure	Expected output
1	It shall be possible for the vehicle to receive its own location wirelessly from the GoT system, through a computer.	The Arduino is programmed to print the current position of the vehicle to the serial port. A computer is connected to it and a serial terminal opened to receive the coordinates. The GoT system is activated, and instructed to send coordinates to the vehicle. The vehicle is not moving, to make the chance for disturbances smaller.	The coordinates of the vehicle, printed in the serial terminal, is equal to the coordinates, sent by the GoT system.
2	It shall be possible for the prototype to disregard incorrect packets transmitted from the computer	The GoT system is programmed to generate incorrect packages (wrong byte value or different length of packages) at random on purpose. The Arduino is programmed to print the error message, where a 0 will be correct and everything else a error. A serial terminal on a connected computer is monitored	The serial terminal shows error messages, that corresponds to the error in the packages sent.
3	The prototype must be able to disregard erroneous coordinates sent from the GoT system	The Arduino is programmed to print the current position of the vehicle, to a computer, monitoring it. The car is moved around, trying to disturb the readings from the GoT system.	The only coordinates received, is coordinates moving slower than 3 $m \cdot s^{-1}$.

Req. no.	Requirements	Test procedure	Expected output
4	The prototype must be able to access the route, which it has to follow, from a storage space located on the vehicle	This test will not be made, as a storage space have not been implemented in the system.	None
5	The prototype must be able to shut down, if the battery voltage is below its cut-off specification	A power supply set to 7,2 V is connected to the vehicle instead of the battery. The vehicle is turned on, and the voltage of the power supply adjusted down until the vehicle stops.	The vehicle stops when the voltage drops to 6V.
6	It shall be possible for the prototype to follow a predetermined route	A route is set up, that follows the points (0,0), (2000,0), (0,2000) and (0,0). The vehicle is placed in the first point and is set to follow the route. A computer is monitoring the coordinates received by the vehicle and the current line, the vehicle follows.	The coordinates received follows the route and the change to a new line, as soon as it is less than 28 cm away from the line's end point.
7	It shall be possible for the prototype to return to the predetermined route if disturbed	The vehicle is programmed to drive in a straight line, and powered on. While driving, the vehicle is pushed sideways, to make it deviate from the route. A computer is monitoring the distance controller output and the error distance.	The vehicle returns to the programmed line and the distance controller output and the error distance will become smaller, as the vehicle close in to the line.
8	The prototype shall be able to keep a velocity on 1,4 $m \cdot s^{-1}$, when going up - or downhill and when turning	The vehicle is set to run at 1,4 $m \cdot s^{-1}$ and is placed, so it will drive uphill and downhill, when set to run. A computer is monitoring the speed through a serial port.	The speed stays at 1,4 $m \cdot s^{-1}$, both on the uphill and downhill part.

10.2 Test Results

Req. no.	Results	Fulfilled?
1	The received coordinates from the GoT system is compared to the log file from the GoT system, shown in *Table 10.1*. All the coordinates is equal to the coordinates measured and sent from the GoT system, and the coordinates have therefore been received correctly.	Yes
2	The packages, including the error packages, that the GoT system sends, are received by the Arduino. The error handling parts of the protocol filters all the error packages away, shown in *Table 10.2*.	Yes
3	The speed between the positions, measured by the GoT system are shown on *Figure 10.1*, where the packages sent to the Arduino are green and those that is not sent are red. All positions, that have moved away faster than 3 $m \cdot s^{-1}$ are not sent.	Yes
4	Test not made	No
5	The vehicle is running, until the power supply is adjusted down to 6V, where the vehicle stops running.	Yes
6	As the steering controller does not work, as the magnetometer does not work in the room with the GoT system, the test could not be performed normally. Instead the vehicle is moved around manually, where the coordinates and the current line is monitored on a computer, which is shown on *Figure 10.2*. The figure shows that the route controls sets new lines correctly and the requirement should be fulfilled with a working steering controller.	No
7	As the steering controller does not work, as the magnetometer does not work in the room with the GoT system, the test could not be performed normally. Instead the vehicle is placed with a distance of 1 m away from the route line and then manually moved closer to the line, while the distance controller and error distance is monitored, which is shown on *Figure 10.3*. This shows that the controller behaves as planned and the requirement should be fulfilled with a working steering controller.	No
8	The vehicle is set to drive 1,4 $m \cdot s^{-1}$ uphill and downhill, which is shown on *Figure 10.4* and *Figure 10.5* respectively. The figures shows the controller keeping the speed at 1,4 $m \cdot s^{-1}$.	Yes

GoT log file	Arduino received	Equal?
53,-23,-271	53,-23,-271	Yes
53,-24,-264	53,-24,-264	Yes
53,-24,-269	53,-24,-269	Yes
55,-24,-267	55,-24,-267	Yes
55,-23,-269	55,-23,-269	Yes
56,-23,-274	56,-23,-274	Yes
57,-22,-274	57,-22,-274	Yes
59,-21,-274	59,-21,-274	Yes
57,-22,-270	57,-22,-270	Yes
55,-23,-269	55,-23,-269	Yes

Table 10.1: The coordinates send form the GoT system to the Arduino, where all 10 sets of coordinates is equals to each other.

Package	Error	Returned error value
240,128,224,28, 128,6,80,33, 216,98,169,15	None	0
240,128,96,27, 0,7,144,33, 216,222,192,15	None	0
240,128,96,28, 192,6,33,216, 30,177,15	In the first package, a byte is missing. The second package's first byte will be read as the 12th byte in the first package	5,1,1,1,1,1,1,1,1,1,1,1
240,128,96,27, 0,7,144,33, 216,222,192,15		
240,128,96,28, 192,6,240,33, 216,24,177,15	None	0
240,128,96,126, 192,6,16,34, 216,22,193,15	Byte 4 is changed	4
240,128,96,28, 192,6,80,34, 216,18,177,15	None	0
26,128,96,28, 192,6,48,34, 216,20,177,15	Byte 1 is changed	1,1,1,1,1,1,1,1,1,1,1,1
240,128,96,28, 192,6,240,33, 216,24,177,15	None	0
240,128,96,28, 128,6,208,63, 33,216,90,177, 15	The package is one byte too long. The system will see the first 12 bytes as a package and 12th byte as the start of a new package.	5,1
240,128,96,28, 192,6,144,33, 216,30,177,15	None	0
240,128,96,28, 192,6,176,33, 216,28,177,15	None	0

Package	Error	Returned error value
240,128,115,28, 128,6,208,33, 216,90,177,15	Byte 3 is changed	3,1,1,1,1,1,1,1,1,1
240,128,96,29, 64,6,112,34, 216,144,161,15	None	0
240,96,96,28, 64,6,48,34, 216,148,177,15	Byte 2 is changed	2,1,1,1,1,1,1,1,1,1

Table 10.2: The packages sent from the GoT system, where some packages purposely contains errors. The returned error value, is the output from the error handling from the protocol. 1 means wrong start byte, 2 is wrong destination, 3 is wrong length, 4 is failed checksum control and 5 is wrong end byte.

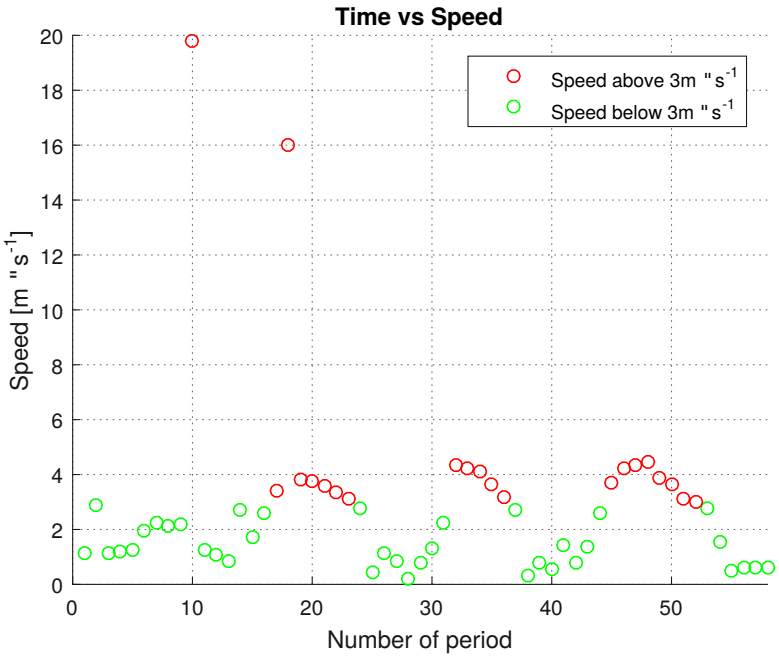


Figure 10.1: Plot of all the velocities between the measured positions, where the red marks indicates discarded measurements and green indicates the measurements sent to the Arduino.

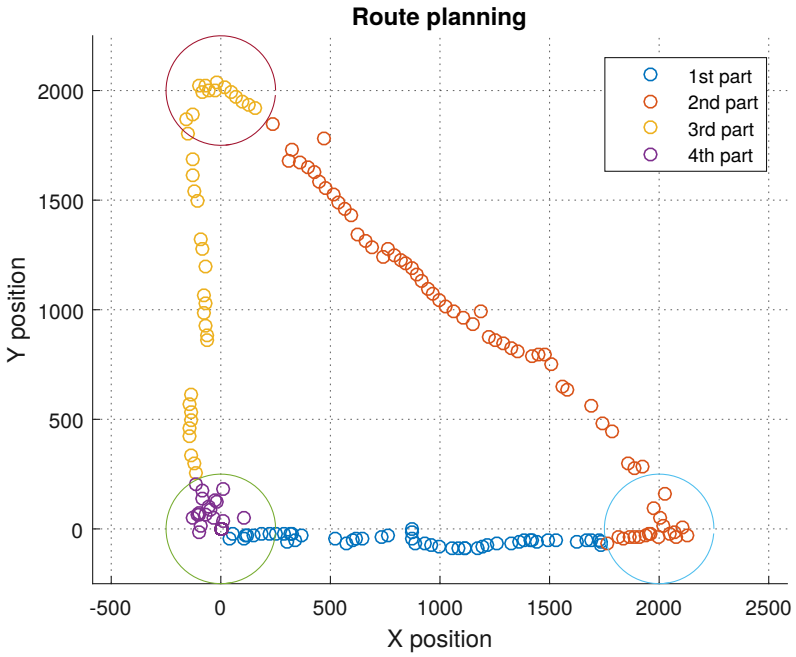


Figure 10.2: Plot of coordinates measured, where the different parts of the route are indicated as different colours. It is seen that the route control shifts to a new line, each time the vehicle gets less than 28 cm away from the end point.

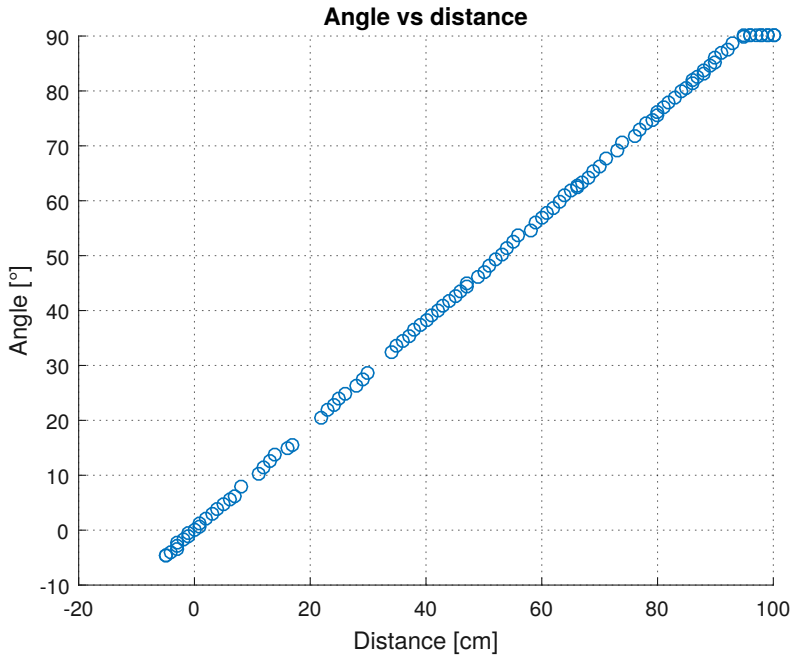


Figure 10.3: Plot of the angle output from the distance controller compared to the error distance. It is seen that as the distance increases, the angle does as well, and saturates at 90 °

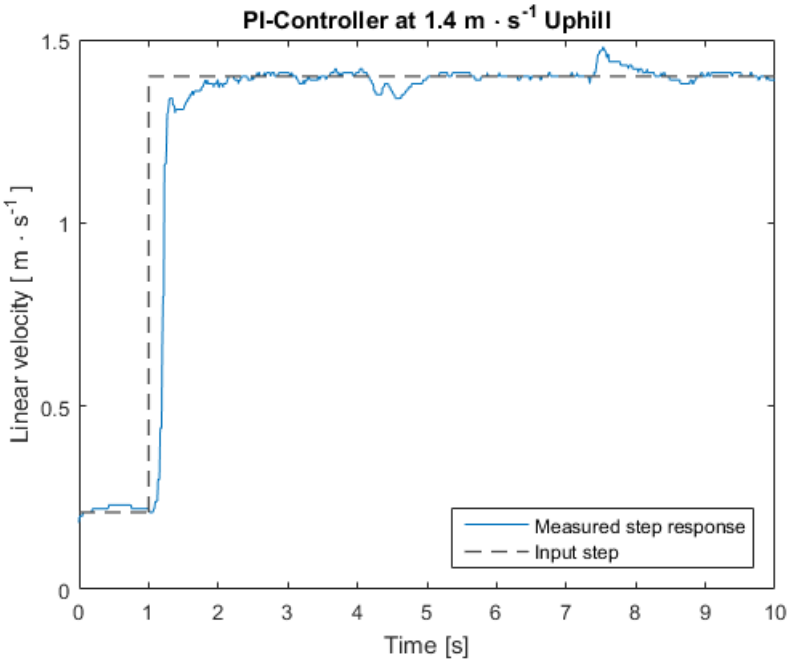


Figure 10.4: Step response for the vehicle, where the vehicle hits an uphill ramp after 4 seconds and hits flat terrain again after 7,5 seconds.

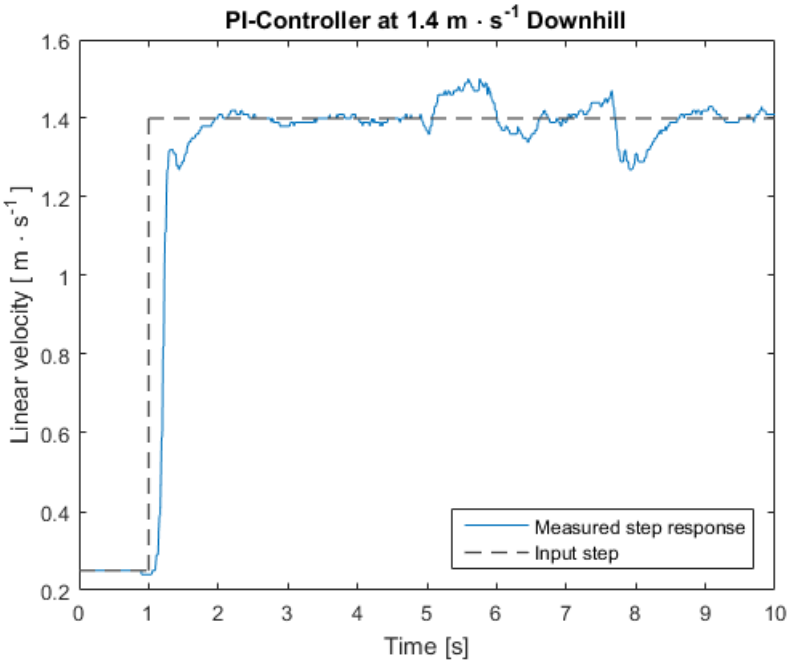


Figure 10.5: Step response for the vehicle, where the vehicle hits a downhill ramp after 5 seconds and hits flat terrain again after around 8 seconds.

The prototype has been tested in correlation with the requirements set for the prototype. 5 out of 8 requirements were met during the test, where another two is expected to be met, if it was possible to test.

11 | Conclusion

In this project, a robotic lawnmower has been designed, and a prototype has been developed.

Alternative navigation approaches to a wire dig-down system have been investigated, and the Games on Track system chosen as a base.

A belt vehicle was provided, for use as a prototype vehicle. This vehicle has been taken apart and analyzed, and prototype requirements have been developed.

For controlling the vehicle, and monitoring sensors, An Arduino Mega was chosen. A Real Time Operating System was implemented, to ensure that correct timings are met in the control loops.

A wireless connection, based on Xbee modules, is used to transfer coordinates from the navigation system to the microcontroller on the prototype. For this, a communication protocol with error handling has been developed, implemented and tested.

A digital filter was developed as an attempt to lower noise from the magnetometer. While the filter worked, the noise was not reduced significantly. The filter was therefore not implemented in the prototype.

To store the route to follow an SD card was chosen, but not implemented because of code compatibility issues with the RTOS.

Mathematical models have been developed for the motor, the linear velocity system and the steering system. The models, with the exception of the distance model, have been simulated and compared to real world tests. The comparisons showed, that the models are accurate enough to design a controller around them, using control theory.

The distance model could not be tested, because of problems with the magnetometer indoor, where the Games on Track system was set up. The controller for this part, was therefore based entirely on a non-testable model. A Proportional controller was deemed inadequate because of overshoots in the simulation, and a Lead Compensator was added to counteract this. The results looks promising, and the controller is therefore expected to work in real life, with a little fine tuning.

Different controller topologies were investigated for controlling the linear velocity of the vehicle, and a Proportional Integral Controller was chosen. For the directional control, a Proportional controller was deemed sufficient. Both of these have been simulated, implemented and tested on the vehicle with good results.

12 | Discussion

Throughout this project a prototype has been designed on the given technology base. The base, being a tracked vehicle is not the most obvious choice for general purpose lawn mowing. However the purpose of the project was to construct a functional prototype to demonstrate the idea of using a local positioning system, GoT, to control the vehicle's path on a lawn. This was achievable with the platform at hand, however the control designed for the tracked vehicle, while having common points with control of wheeled vehicles, is not as portable if an end product was to be designed.
The idea of using the GoT system for navigation has been investigated and even though the combined system test was not carried out, due to disturbances in the magnetic field in the Vicon room, the subsequent system tests were largely successful, and proved the concept. To provide further proof of concept however, it would be preferable to test the combined system outdoors. Furthermore, an enhancement on the positioning system could improve the route following by gaining in accuracy.
An other thing is to consider the route planning itself, which has not been done in its entirety. As of now, there is a route control system, that handle the execution of the route following. An algorithm to plan the path with a means of initializing the edges of the lawn, would be interesting additions to the current design. Here it was discussed that the GoT transmitter, placed on the vehicle, could be removable. This would allow the user to take the transmitter off and through a simple interface, e.g. holding down a button, could activate recording of coordinates and by walking along the edges of the lawn, be able to initialize the area in which the lawn mover had to move.
The prototype could also be taken further by implementation on an actual lawn mower, including control of the blade, concerning grass height. The speed of the vehicle might also need to be altered if the grass has different densities or length, in order to allow an evenly cut lawn.
Several other small features could be added, like detection of humidity to make sure that the lawn is only cut when it is dry enough that a good result is obtained. An obstacle detection functionality could also be added to account for unmapped objects on the lawn that could be in the lawn mower's path, e.g. stones, child toys or pets. The idea of a grass length detector was also suggested. This would allow to adapt the speed of the lawn mower according to the grass length, still to ensure that it is evenly cut.

Furthermore, the distance controller, and therefore the whole steering controller, did not get tested, as the magnetometer did not work properly inside the Vicon room. This made the two requirements for the steering untestable. By taking the GoT system outside, where the magnetometer works, and calibrating the system to the new location, a test could be made. This would be a sufficient test since, of course, a lawn mover operates outdoors. However, the time limitation of the project resulted in the test being down-prioritized, due to the somewhat time consuming setup and calibration of the GoT system.

The last requirement, that has not been fulfilled, was the requirement about nonvolatile

storage on board the vehicle. It was attempted to include an SD-card, and while it was possible to read and write to/from the SD-card, it was not made to function within the real time operating system. This problem can undoubtedly be solved, however the time limitation and other priorities resulted in constraining the prototype from this requirement. This functionality was not prioritized since it did not have any significant impact on the resulting state of the prototype.

Appendix

A | Motor Tests - Armature Resistance

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of the test is to measure the armature resistance R_a of the motor.

Setup

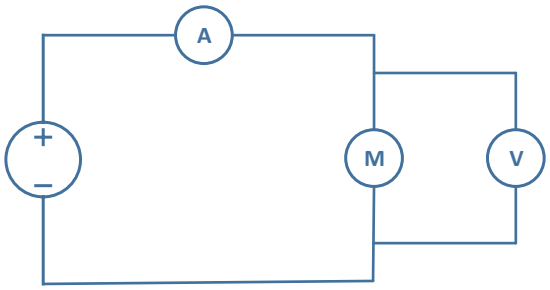


Figure 12.1: Setup diagram

List of Equipment

Instrument	AAU-no.	Type
Multimeter 1	60764	Fluke 189 True RMS
Multimeter 2	60769	Fluke 189 True RMS
Power Supply (0 - 32 V) (0 - 10 A)	77076	Ea - ps 7032 - 100
Clamp for fixing the motor	03039	

Procedure

1. Turn on the two multimeters and choose Voltage and Ampere settings respectively.
2. Fix the motor shaft so it can not turn.
3. Choose the first current value (0,0 A) on the current limiting of the power supply.
4. Turn on the power supply and adjust the current limiting in accordance with the ampere meter.
5. Read the voltage supplied to the motor from the volt meter.

6. Repeat the three previous steps for each measurement in 0,5 A increments up to 5 A.

7. Switch the poles of the power supply and repeat the measurements in the negative direction.

Results

Input (A)	Output (V)
−5,0	−0,71
−4,5	−0,65
−4,0	−0,59
−3,5	−0,54
−3,0	−0,43
−2,5	−0,36
−2,0	−0,27
−1,5	−0,20
−1,0	−0,14
−0,5	−0,07

Input (A)	Output (V)
0,5	0,16
1,0	0,34
1,5	0,53
2,0	0,62
2,5	0,64
3,0	0,75
3,5	0,78
4,0	0,80
4,5	0,83
5,0	0,88

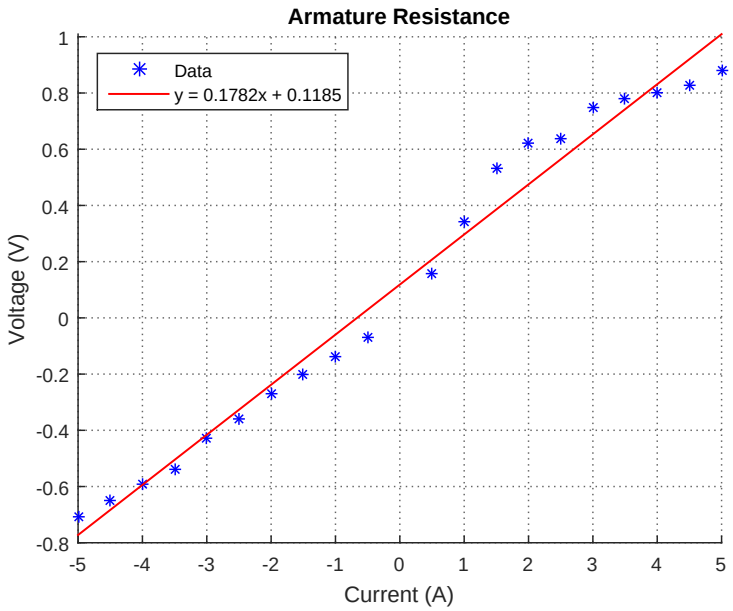


Figure 12.2: A plot of a measured armature resistance, with a red line indicating the average value.

During these measurements the motor is in steady state. This is necessary for the inductor in the armature coil to act as a short circuit, which ease the calculation of the armature resistance. In steady state we get:

$$R_a = \frac{U_a}{I_a} \quad [\Omega]$$

Where:

I_a	is the armature current	[A]
U_a	is the armature voltage	[V]
R_a	is the armature resistance	[Ω]

As seen on the data plot in *Figure 12.2* the result is a relatively linear function. The armature resistance is approximated directly as the slope of the least square line:

$R_a = 0{,}178\,\Omega$

B | Motor Tests - Armature Inductance

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of the test is to determine the armature inductance L_a of the motor.

Setup

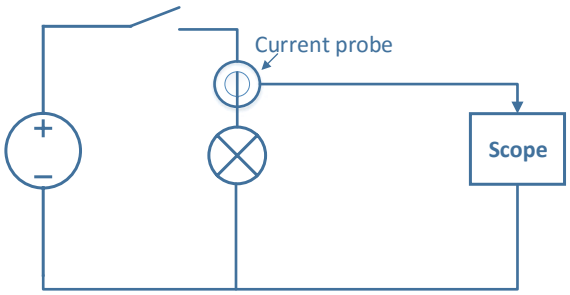


Figure 12.3: Setup diagram

List of Equipment

Instrument	AAU-no.	Type
Power Supply $(0-32\ V)\ (0-10\ A)$	77076	Ea - ps 7032 - 100
AC/DC Current Clamp (Output: 100 mV/A)	78550	FLUKE i30s
Oscilloscope	64672	Agilent DSO6034A
Clamp for fixing the motor	03039	

Procedure

1. Fix the motor shaft so it can not turn.
2. Start with the power supply disconnected and turn on the oscilloscope.
3. On the oscilloscope press the "trigger mode"-key choose the "normal"-option and push the "single"-key.
4. To prevent false triggering on the oscilloscope, set the trigger value to 113 mV.
5. To supply the motor a pulse of 5 V, adjust the power supply to 5 V and connect it.
6. Connect a USB-drive to the oscilloscope and press the save key to extract the data.

Results

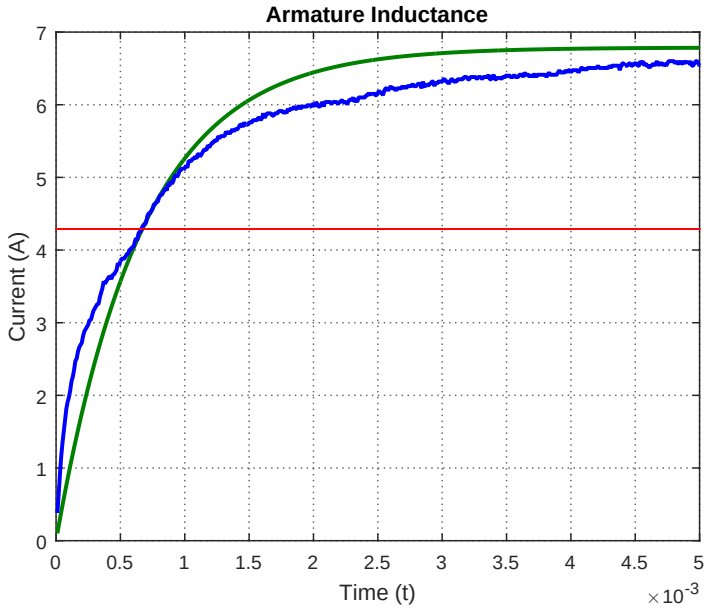


Figure 12.4: A plot of a current step response of the motor, where the blue line is data, the green line is the ideal step response and the red line is the time constant

The armature inductance, L_a, is calculated from the time constant given by:

$$\tau = \frac{L_a}{R_a} \quad [s]$$

Where:

τ	is the time constant	[s]
L_a	is the armature inductance	[H]
R_a	is the armature resistance	[Ω]

R_a is know from the previous test, *Armature Resistance*, where it was found to $0,178\Omega$. τ is the time at which the current reaches $63,2\%$ of the value at steady state. This value of τ is found by use of *Figure 12.4* and located in the data set:

$\tau = 0,67 \cdot 10^{-3}$ s

From this we get the armature inductance:

$$L_a = \tau \cdot R_a \quad [H]$$
$$L_a = 0,67 \cdot 10^{-3} \cdot 0,178 \quad [H]$$
$$L_a = 119,26 \quad [\mu H]$$

C | Motor Tests - Tachometer Constant

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of the test is to measure and verify that the tachometer constant is 0,030V multiplied by the motor velocity in radians per second.

Setup

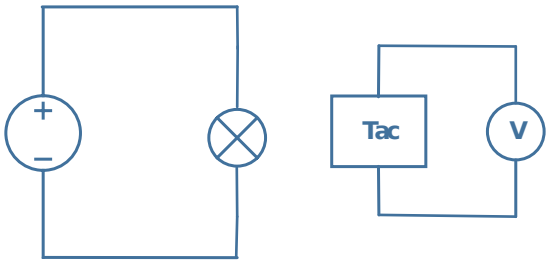


Figure 12.5: Setup Diagram

List of Equipment

Instrument	AAU-no.	Type
Power Supply $(0-32\ \text{V})\ (0-10\ \text{A})$	77076	Ea - ps 7032 - 100
Multimeter	60764	Fluke 189 True RMS
Optical tachometer	08246	Shimpo DT-205

Procedure

1. Adjust the voltage of the power supply so that the multimeter measures 6 V over the tachometer.
2. Measure the RPM with the Optical tachometer.

Results

The tachometer measured 1933 RPM at 6 V, which is used to verify a tachometer constant of 0,03 :

$$\frac{1933}{60} \cdot 2 \cdot \pi \cdot 0{,}03 = 6{,}07 \approx 6\ \text{V} \tag{12.1}$$

D | Motor Tests - Generator Constant

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of the test is to find the generator constant K_e by measuring the motor voltages, currents and velocities, in several steady states.

Setup

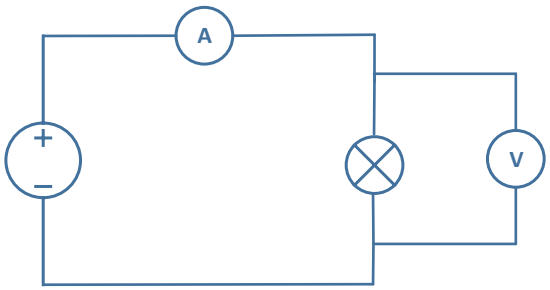


Figure 12.6: Setup diagram

List of Equipment

Instrument	AAU-no.	Type
Multimeter 1	60764	Fluke 189 True RMS
Multimeter 2	60769	Fluke 189 True RMS
Power Supply $(0-32\ \text{V})$ $(0-10\ \text{A})$	77076	Ea - ps 7032 - 100
Optical tachometer	08246	Shimpo DT-205

Procedure

1. Turn on the two multimeters and put them in ampere and voltage mode respectively.
2. Adjust power supply to 1 V using the voltage mode multimeter, and connect the motor.
3. Read out the current value from the ampere mode multimeter.
4. Read out RPM of the motor using the optical tachometer.
5. Repeat the past 3 steps up to 7 V in 1 V increments.

Results

Input (V)	Output (A)	Output (RPM)	Generator constant
1	1,7	3684	0,00181
2	2,2	8063	0,00191
3	2,6	12021	0,00202
4	3,3	16746	0,00195
5	4,1	21966	0,00186
6	4,8	26420	0,00186
7	5,6	31447	0,00182

The equation for the generator constant is given by:

$$K_e = \frac{U_a - R_a \cdot I_a}{\omega} \quad [\text{Wb}] \quad (12.2)$$

Where:

ω	is the angular velocity	$[\text{rad} \cdot \text{s}^{-1}]$
R_a	is the armature resistance	$[\Omega]$
I_a	is the armature current	[A]
K_e	is the generator constant	[Wb]

The generator constants for each measurement are not equal, but with a small margin in difference. The average generator constant is:

$K_e = 0,00189$ Wb

E | Motor Tests - Motor Constant

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of this of this test is to measure the motor constant K_t.

Setup

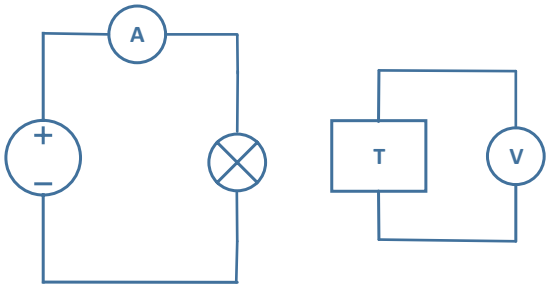


Figure 12.7: Setup diagram

List of Equipment

Instrument	AAU-no.	Type
Multimeter 1	60764	Fluke 189 True RMS
Multimeter 2	60769	Fluke 189 True RMS
Power Supply $(0-32\ \text{V})$ $(0-10\ \text{A})$	77076	Ea - ps 7032 - 100
Torque sensor	08772	Icom

Procedure

1. Connect the motor shaft to the torque sensor.
2. Turn on the two multimeter in current and voltage mode respectively.
3. Start by setting the power supply at 1 A current limiting.
4. Turn on the supply and note the voltage across the torque sensor.
5. Repeat the previous two steps up to 10 A with 1 A increments.

Results

Input (A)	Output (V)
1	0,107
2	0,113
3	0,118
4	0,122
5	0,131
6	0,142
7	0,149
8	0,158
9	0,169
10	0,180

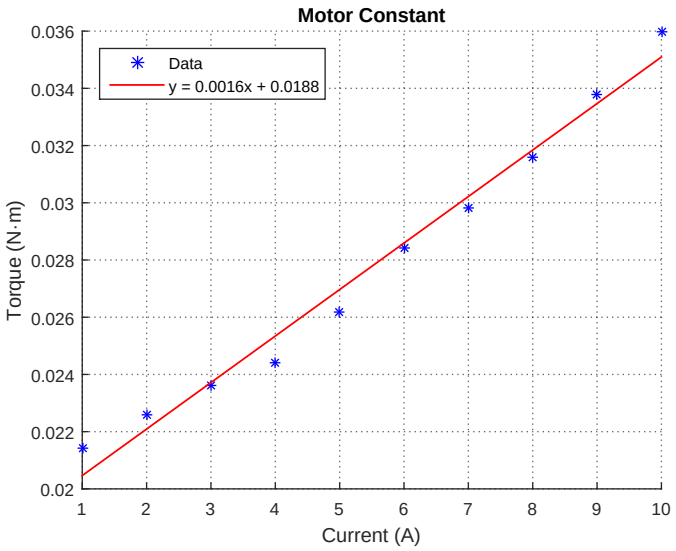


Figure 12.8: A plot of the torque at different currents, where the blue dots is the measurements and the red line is the least square line.

The voltages measured are scaled by factor of 0,2 since the torque sensor outputs is 5 V per 100N · cm giving a torque of 0,2 $\mathrm{N \cdot m \cdot V^{-1}}$ [46]. After scaling the voltages to the torques, the measurement is plotted and a least square line is added as seen on *Figure 12.8*. The relation between the current and torque is described as follows:

$$\tau = \mathrm{K_t \cdot I_a} \qquad [\mathrm{N \cdot m}]$$

$$\mathrm{K_t} = \frac{\tau}{\mathrm{I_a}} \qquad \left[\mathrm{N \cdot m \cdot A^{-1}}\right]$$

Where:

τ	is the torque	$[\mathrm{N \cdot m}]$
$\mathrm{I_a}$	is the current supplied to the motor	[A]
$\mathrm{K_t}$	is the motor constant	$[\mathrm{N \cdot m \cdot A^{-1}}]$

The value of $\mathrm{K_t}$ is then extracted directly as the slope of the least square regression:

$\mathrm{K_t} = 0{,}0016\ \mathrm{N \cdot m \cdot A^{-1}}$

F | Motor Tests - Friction

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of the test is to find the motor friction, B, by measuring the motor current and the corresponding velocities, in several steady states.

The data from the test *Generator Constant* is reused, the equipment and setup is then the same.

Results

Input (A)	Output (RPM)
1,7	3684
2,2	8063
2,6	12021
3,3	16746

Input (A)	Output (RPM)
4,1	21966
4,8	26420
5,6	31447

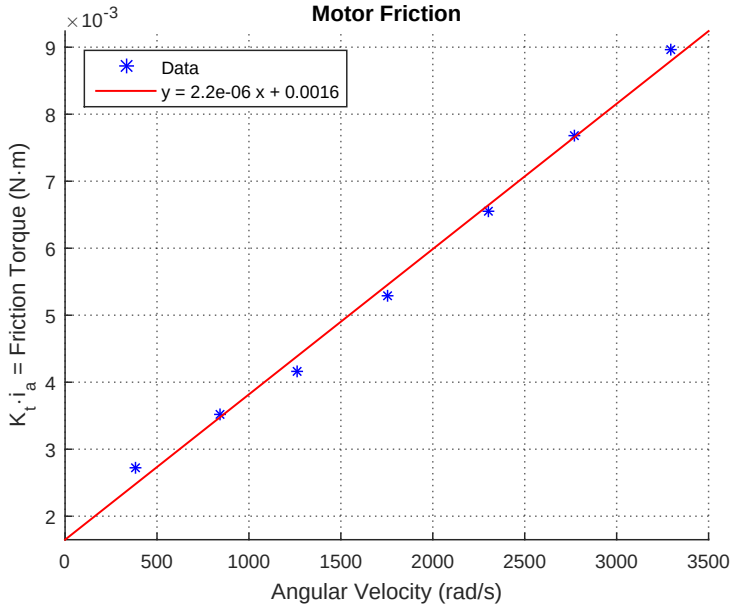


Figure 12.9: A plot illustrating the motor constant multiplied by the armature current to give the friction torque, which is plotted as a function of the angular velocity. The blue dots indicates the measurements and the red line indicates the least square line.

The following equation is applied to obtain the motor friction:

$$B = \frac{K_t \cdot I_a}{\omega} \qquad \left[N \cdot m \cdot rad^{-1} \cdot s\right]$$

Where:

K_t	is the motor constant	$[N \cdot m \cdot A^{-1}]$
I_a	is the armature current	[A]
B	is the motor friction	$[N \cdot m \cdot rad^{-1} \cdot s]$
ω	is the motor angular velocity	$[rad \cdot s^{-1}]$

This relationship is plotted in *Figure 12.9* and a least square line is added. The friction, B, is found as the slope and the Coulomb friction (stiction), τ_c, is the interception between the least square line and the y-axis.

$B = 2{,}2 \cdot 10^{-6}\ N \cdot m \cdot rad^{-1} \cdot s$
$\tau_c = 0{,}0016\ N \cdot m$

G | Motor Tests - Moment of Inertia

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of this test is to find the moment of inertia I, by measuring the motor velocity as a function of time.

Setup

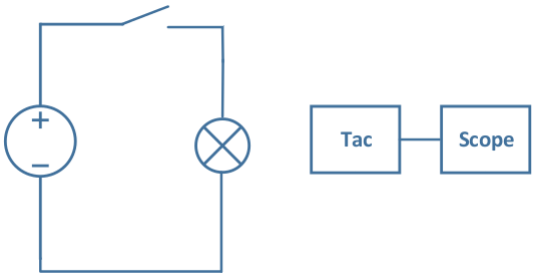


Figure 12.10: Setup diagram

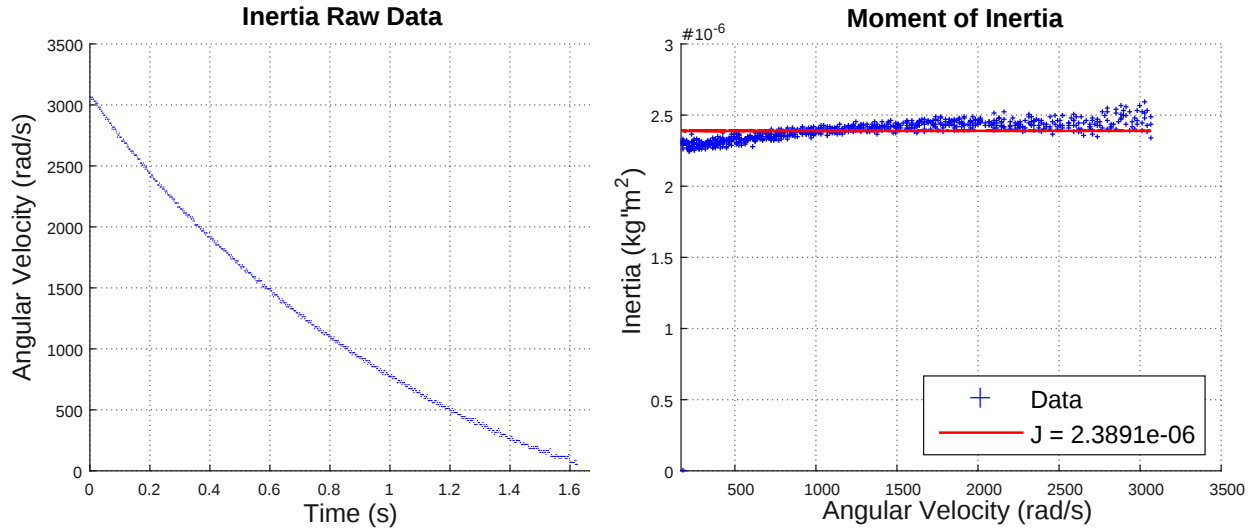
List of Equipment

Instrument	AAU-no.	Type
Oscilloscope	64672	Agilent DSO6034A
Power Supply $(0-32\ \text{V})$ $(0-10\ \text{A})$	77076	Ea - ps 7032 - 100
Optical tachometer	77087	Compact

Procedure

1. Turn on the oscilloscope, and connect the tachometer to one of the inputs.
2. On the oscilloscope press the "trigger mode"-key choose the "normal"-option, set the trigger to "falling edge".
3. To prevent false triggering on the oscilloscope set the trigger value to 1,175 V with the turn-key.
4. Turn on the power supply at 7 volt.
5. Press "single"-key on oscilloscope and cut the power of the motor.
6. Insert a USB-flash drive in the oscilloscope and press the save key to extract the data.

Results



(a) The measured angular velocity compared to time, where the blue line indicates the velocity after the power has been removed.

(b) The Blue dots indicates the moment of inertia calculated using *Equation (12.3)*, and the red line represents the average value.

Figure 12.11: A plot of the inertia measured as the angular velocity compared to time and a plot of the moment of inertia calculated from the raw data using *Equation (12.3)*.

The equation used is given in the Modeling and Control course of the 5th semester on Electronic and IT at Aalborg University, [47] . This equation which arises from the mechanical description of the motor, when $i_a = 0$ is used to find the Inertia, J.

$$\omega(s) = -\frac{\tau_c}{B} + (\omega_0 + \frac{\tau_c}{B})e^{-t\frac{B}{J}} \qquad \left[\text{rad} \cdot \text{s}^{-1}\right]$$

$$\Updownarrow$$

$$J = \frac{B \cdot t}{\ln(\frac{B \cdot \omega_0 + \tau_c}{B \cdot \omega + \tau_c})} \qquad [\text{N} \cdot \text{m}] \quad (12.3)$$

Where:

$\omega(s)$	is the angular velocity	$[\text{rad} \cdot \text{s}^{-1}]$
τ_c	is the coulomb friction torque	$[\text{N} \cdot \text{m}]$
B	is the friction	$[\text{N} \cdot \text{m}]$
J	is the inertia	$[\text{kg} \cdot \text{m}^2]$

The motor's inertia is calculated by using *Equation (12.3)* with the angular velocity compared to time illustrated on *Figure 12.11a*. Thereafter the average value of the results is found, seen in *Figure 12.11b* and used as the motor inertia. The motor's inertia is equal to:

$$J = 2{,}3891 \cdot 10^{-6}\ \text{kg} \cdot \text{m}^2$$

H | Motor Tests - Time Constant and Gain

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of this test is to find the motor's time constant τ and gain. This is done by measuring the motor's step response.

Setup

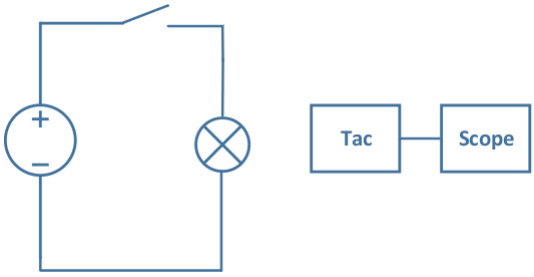


Figure 12.12: Setup diagram

List of Equipment

Instrument	AAU-no.	Type
Oscilloscope	64672	Agilent DSO6034A
Power Supply $(0-32\ \text{V})\ (0-10\ \text{A})$	77076	Ea - ps 7032 - 100
Optical tachometer	77087	Compact

Procedure

1. Turn on the oscilloscope, and connect one channel to the power supply, and another to the tachometer.
2. On the oscilloscope press the "trigger mode"-key choose the "normal"-option, set the trigger to "rising-edge", and the trigger source to the channel connected to the power supply.
3. To prevent false triggering on the oscilloscope set the trigger value to 4,50V with the turn-key.
4. Press "single"-key on oscilloscope.
5. Turn on the power supply at 5 volt.
6. Insert a USB-flash drive in the oscilloscope and press the save key to extract the data.

Results

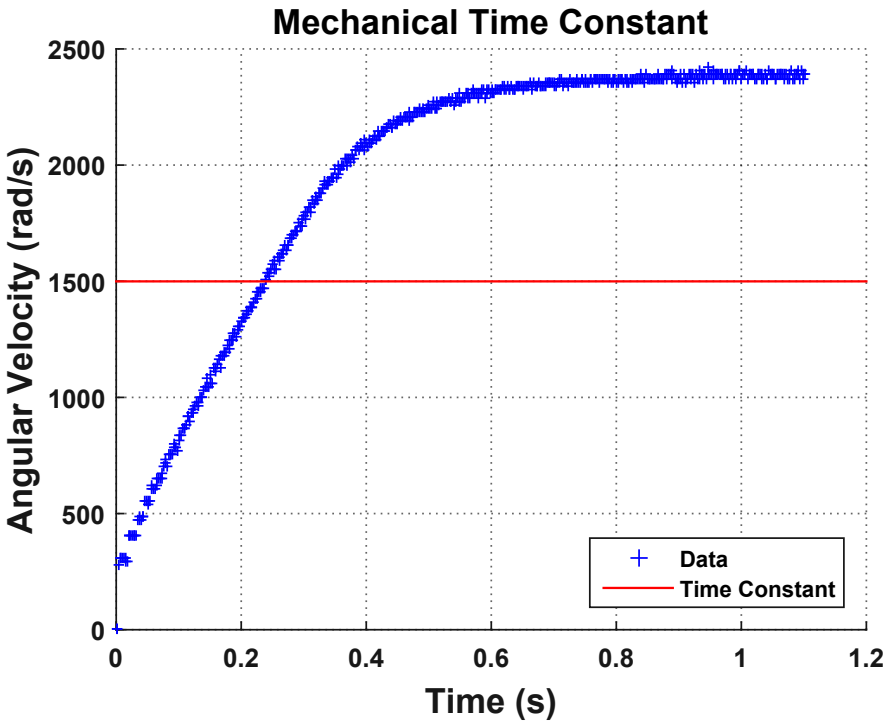


Figure 12.13: A plot of the step response illustrating the angular velocity over time. The blue dots is the measurements and the red line indicates the mechanical time constant.

The graph in *Figure 12.13* shows the angular velocity of the motor over time. The red line shows the time constant at 63,2% of the maximum angular velocity. In the data the mechanical time constant is found to be:

$\tau_{\text{mec}} = 0,238$ s

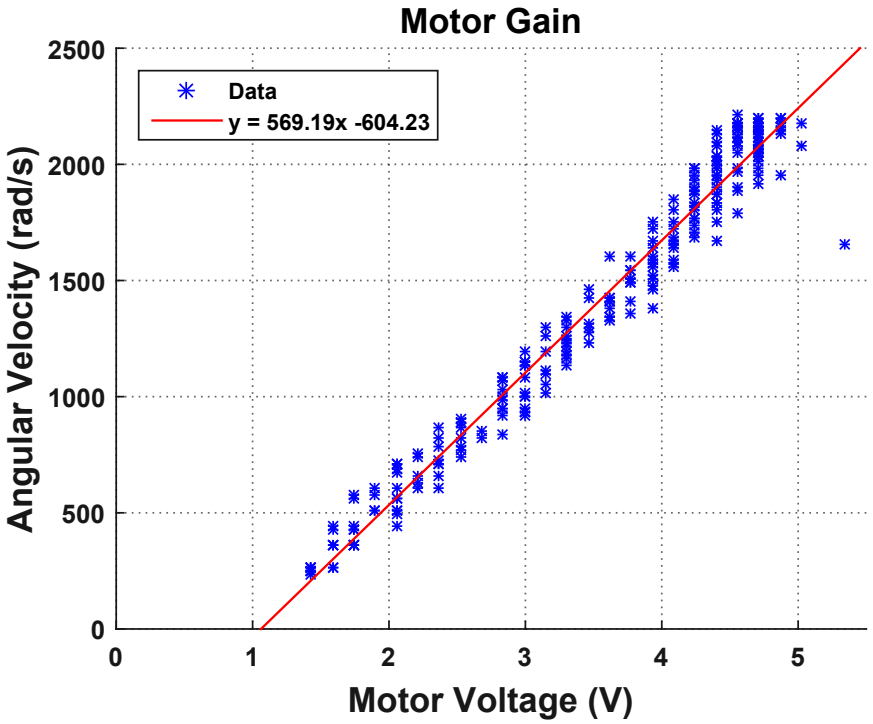


Figure 12.14: A plot of the motor's step response, where the input is motor voltage and the output is the angular velocity. The blue dots indicates the measurements and the red line is the tendency line.

The graph in *Figure 12.14* shows the motor voltage in relation to the angular velocity, which reveals the gain K of the system as the slope of the least square regression line:

$$K = 569,19 \text{ rad} \cdot s^{-1} \cdot V^{-1}$$

I | Sensor Test - Magnetometer Calibration

Name: Group 510
Date: 26/11 - 2015

Purpose

The purpose of this test is to ensure the fact that the magnetometer is calibrated so that it can tell a correct position relative to the Earth's magnetic field when placed on the vehicle.

Theory

The HMC5883L chip [34] is a magnetometer which uses the Earth's magnetic field as a reference. When using it for the first time it is necessary to calibrate it so the disturbances in the close field of the sensor are accounted in when doing measurements.
In this project the disturbances can be caused by all the metallic pieces on the vehicle, from the metal plate to the motor and to the wires. Thus, the calibration has to be performed with the all these components and the sensor placed on the vehicle at a fixed place and orientation. It is chosen to place it, so that its X axis points towards the front of the vehicle and the Z axis is pointed upwards. The chosen configuration shall not be changed after calibration.

As stated in *Section 5.2*, the magnetometer that is used gives out three space coordinates of the Earth's magnitude field relative to its own coordinate system. One good test to see if the sensor is operational in its environment is to turn it all around itself while retrieving the data it gives. When properly calibrated and isolated from any new disturbance, the 3D scatter plot of all the measured points should give a sphere that has its center in the origin (coordinates (0,0,0)) of the plot. This means that the measured coordinates vary smoothly accordingly with the rotation of the sensor in space. Otherwise, the sphere could be distorted into an ellipsoid and displaced from the center and the sensor would have to be recalibrated correctly in a magnetic-still environment, see example with the magnetometer alone on *Figure 12.15*.
A simpler test is used here to check for the good calibration with the vehicle. It consists in making four quarters of turn in the XY-plane (horizontally), around a static point. If each quarter corresponds to a measurement of 90° in the sensor's data and the resulting four quarters draw a circle, then the calibration is fine to use.

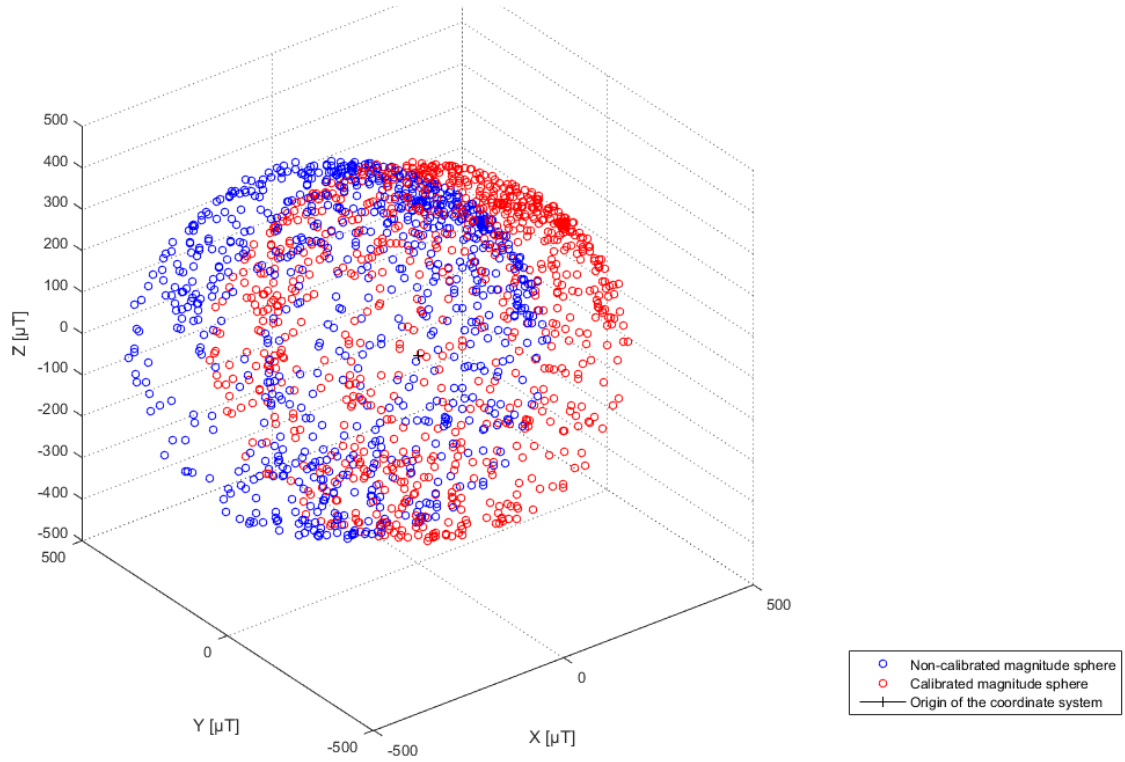


Figure 12.15: Magnetic sphere of calibrated and non-calibrated sensor alone

The calibration itself performed in this test does not change the intrinsic sensor configuration but rather applies a transformation to the coordinates sent to the Arduino board. This transformation, seen in *Equation (12.4)* is actually computed in the runtime by the microcontroller, each time a set of coordinate is received :

$$X_{cal} = M \cdot X_{measured} + B \quad [G] \quad (12.4)$$

Where:

X_{cal}	is a 3×1 vector of calibrated coordinates	[G]
M	is a 3×3 transformation matrix	[·]
$X_{measured}$	is a 3×1 vector of non-calibrated coordinates	[G]
B	is a 3×1 bias vector	[G]

The transformation matrix, M, allows for the transformation of an ellipsoid into a proper sphere, while the bias vector shifts the resulting sphere to the center of the sensor's coordinate system. The expanded matrix form is the following :

$$\begin{pmatrix} x_{cal} \\ y_{cal} \\ z_{cal} \end{pmatrix} = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} x_{measured} \\ y_{measured} \\ z_{measured} \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} \quad [G] \quad (12.5)$$

Each set of measured space coordinates, $X_{measured}$, is put into *Equation (12.5)*. The coefficients of the transformation matrix, M, and of the bias vector, B, are found from a dedicated software, MagMaster [48].
This software, runs on Microsoft Windows computers with the HMC5883L sensor connected through an Arduino. It requires to put the sensor in different pre-defined positions, so that it points towards all the axis successively, and finally computes the needed values. Then finally it has to be put into the software that needs to use the magnetometer readings and apply the desired calculations.

Setup

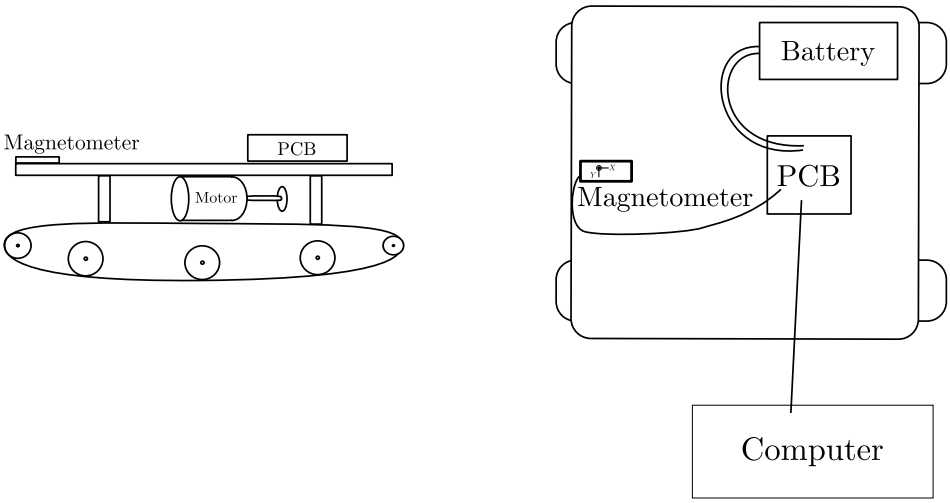


Figure 12.16: Setup diagram

List of Equipment

Instrument	**Type**
Computer	HP 8460P

Procedure

1. Set up the vehicle and the hardware so that the magnetometer is far from other non-constant magnetic objects (motor, PCB, wires) and with the X arrow pointing towards the front of the vehicle and the Z pointing upwards.

2. Attach everything, including the wires and serial USB cable, as it is, so that it cannot move. The configuration should stay the same as long as the magnetometer is used, otherwise, it has to be re-calibrated.

3. Put the vehicle in the environment in which the magnetometer has to operate.

4. Plug the sensor to the Arduino board and the Arduino to the computer via the USB serial cable.
5. Run the MagMaster software.
6. Select the right USB serial port.
7. Start the measurements by putting the vehicle in the position indicated by the software tutorial [48]. Click on the associated measurement button.
8. While performing the test, make sure that no magnetic part is moving (especially the USB cable).
9. When all positions are measured through the software, it is possible to unplug the serial connection with the Arduino.
10. Click on 'Calculate Transformation Matrix and Bias'.
11. Write down the calculated coefficients for later reference and put them in the Arduino sensor code.
12. Test the sensor by using the four-quarters of turn method.

Results

After having done the measurements and computed the transformation matrix and bias vector with the software, with the magnetometer on the vehicle, the coefficients used further on in the code are :

$$M = \begin{pmatrix} 1,206 & -0,002 & 0,039 \\ 0,013 & 1,193 & -0,004 \\ -0,004 & -0,025 & 1,414 \end{pmatrix} \quad [\mathrm{G}] \quad (12.6)$$

$$B = \begin{pmatrix} -52,455 \\ -164,666 \\ -143,707 \end{pmatrix} \quad [\mathrm{G}] \quad (12.7)$$

The four-quarters of turn test made with these settings is illustrated in *Figure 12.17*.

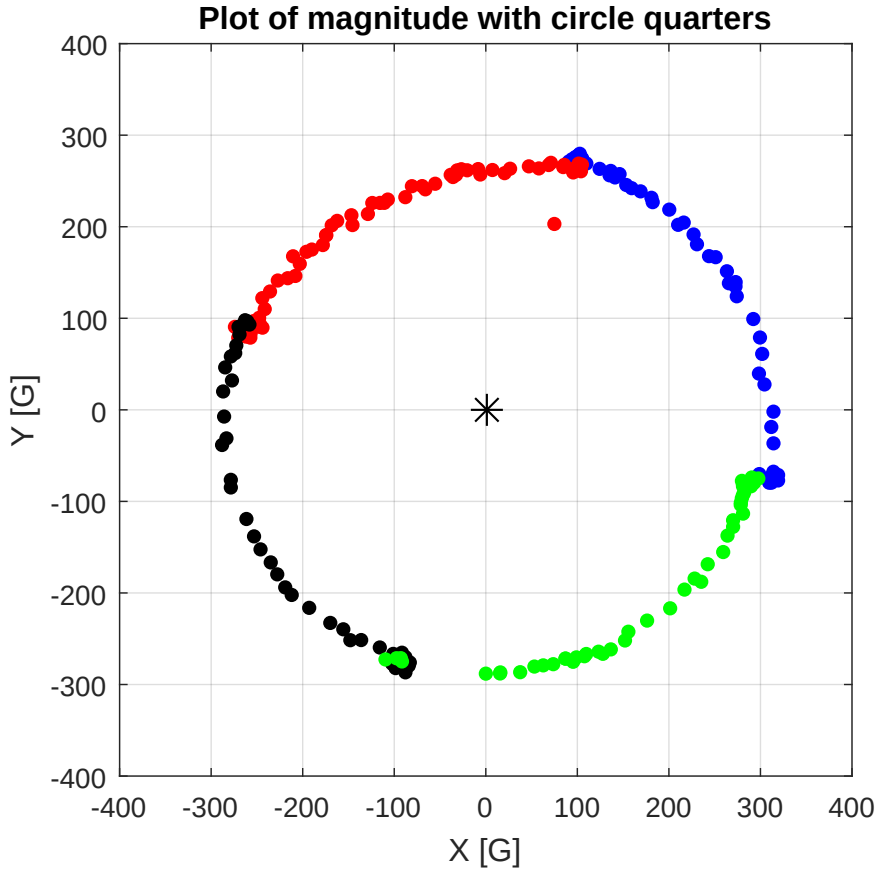


Figure 12.17: Test of the calibrated sensor on the the vehicle that rotates doing four quarters of turn

The cross on *Figure 12.17* has right angles and passes through the center of the circle which is in (0,0). This shows that each quarter of turn in reality is measured by 90° by the sensor, and that it is correctly calibrated to be used on the vehicle.

J | Vehicle Tests - Friction

Name: Group 510
Date: 28/10 - 2015

Purpose

The purpose of the test is to find the total friction, B, of the vehicle.

Setup

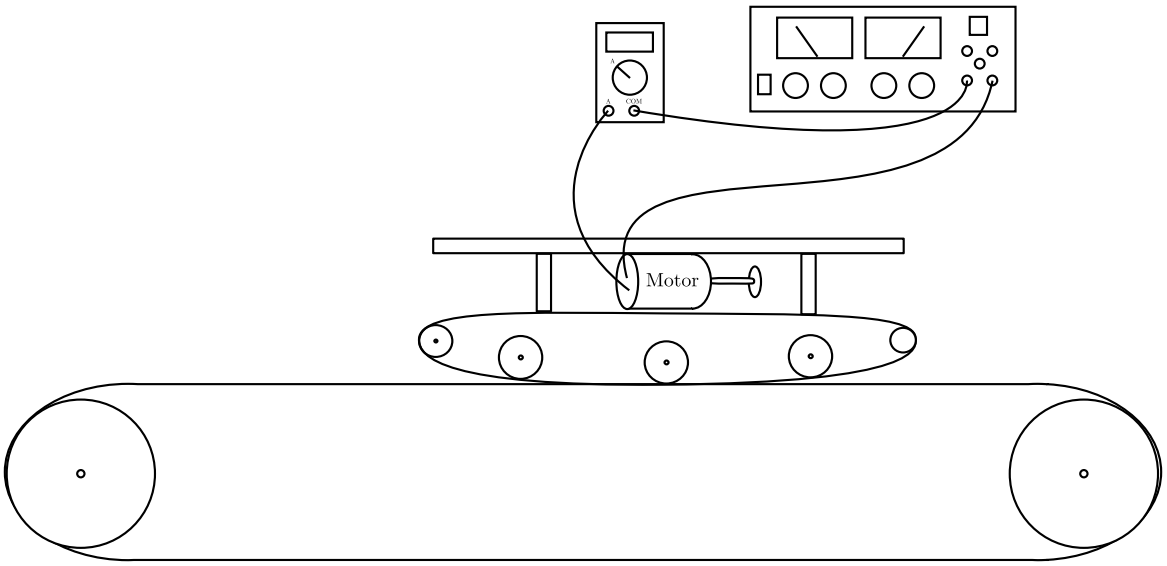


Figure 12.18: A diagram of the test setup

List of Equipment

Instrument	AAU-no.	Type
Multimeter	60764	Fluke 189 true RMS
Power Supply $(0-32$ V$)$ $(0-10$ A$)$	77075	Ea - ps 7032 - 100
Treadmill	75483	Rodby

Procedure

1. Set up the electrical circuit to power the motor with current, so that it is possible to read the current value while changing it with the power supply.

2. Place the vehicle in the middle of the treadmill.

3. Set the power supply to current limiting at a maximum of 4 A, the vehicle is not running, but ready to run when the power supply is set above 4 A later.
4. Activate the treadmill to run at the required speed, and at the same time correct the power supply so the vehicle receives more than 4 A and thereby making it move.
5. Correct the speed of the vehicle with the power supply until the vehicle has the same velocity as the treadmill.
6. Read the current on the multimeter.
7. Repeat the process with different velocities.

Results

The raw data extracted from the measurements is the vehicle's velocity in $[km \cdot h^{-1}]$ and the current needed for achieving the specific velocity. The velocity is calculated from $[km \cdot h^{-1}]$ to a linear velocity $[m \cdot s^{-1}]$:

$$\mathrm{V} = \mathrm{V_m} \cdot \frac{10^3}{3600} \qquad \left[\mathrm{m \cdot s^{-1}}\right] \quad (12.8)$$

Where:

V_m	is measured velocity of the vehicle	$[\mathrm{km \cdot h^{-1}}]$
V	is the vehicle's linear velocity	$[\mathrm{m \cdot s^{-1}}]$

By using linear velocity $[m \cdot s^{-1}]$, the motor's angular velocity $[rad \cdot s^{-1}]$ can be calculated:

$$\omega_\mathrm{m} = \frac{\mathrm{V} \cdot \mathrm{N}}{\mathrm{r_t}} \qquad \left[\mathrm{rad \cdot s^{-1}}\right] \quad (12.9)$$

Where:

ω_m	is the motor's angular velocity	$[\mathrm{rad \cdot s^{-1}}]$
N	is the gear ratio of the vehicle	$[\cdot]$
r_t	is the radius of the two wheels driving the belts	[m]

The motor's torque and angular velocity is plotted:

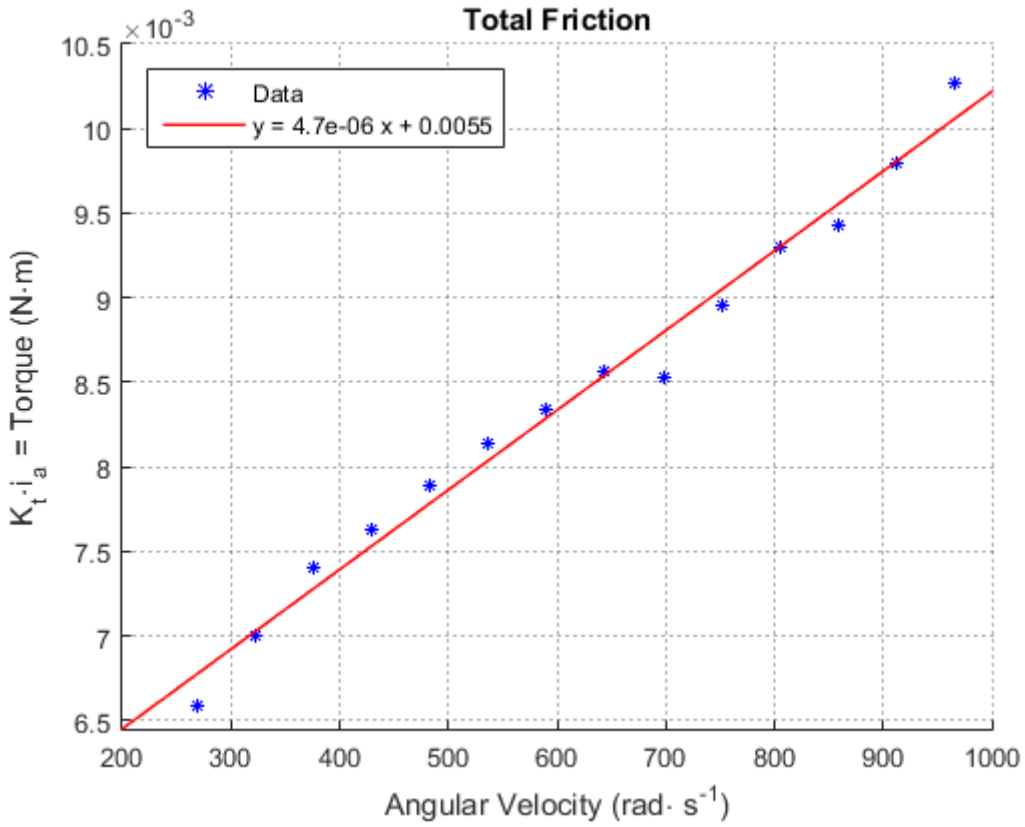


Figure 12.19: A plot of the motor's torque over the motor's angular velocity. The blue dots indicates the measurements and the red line is the tendency line.

The red line in *Figure 12.19* is the tendency line, the slope indicates the friction and the offset is the stiction of the system. The total friction of the system is therefore:

$$B_{tot} = 4,7 \cdot 10^{-6}\ \mathrm{N \cdot m \cdot rad^{-1} \cdot s} \tag{12.10}$$

K | Vehicle Tests - Inertia and Time Constant

Name: Group 510
Date: 02/11 - 2015

Purpose

The purpose of the test is to measure the velocity system's time constant and find the inertia of the vehicle, J_{tot}, using this time constant.

Theory

Using two steps

In this test a two-step input is used to get the system's response. The reason for this is to avoid the stiction (also called Coulomb friction) when the vehicle starts. The second step input is not sent until the vehicle is in a steady and non-zero velocity. Without these two steps the first data points of the response are unusable, as seen in the results on *Figure 12.20*.

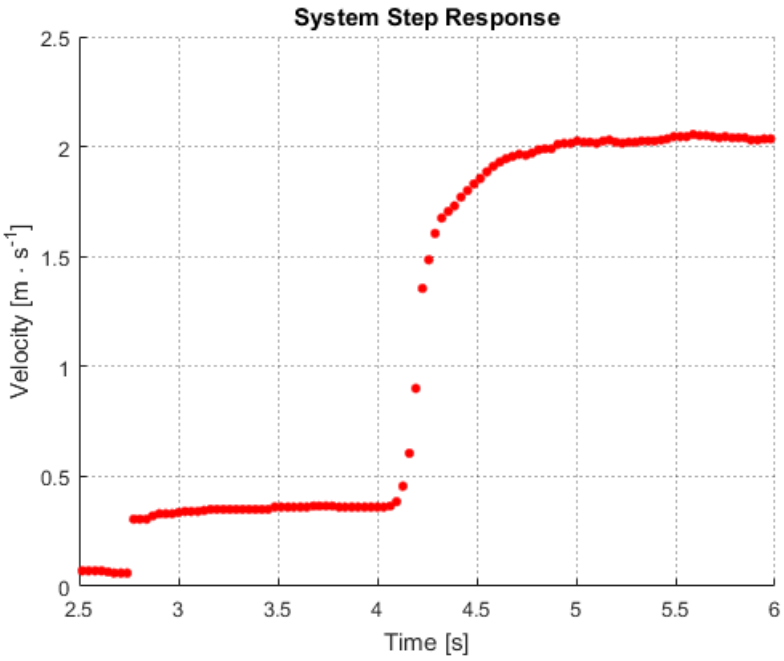


Figure 12.20: Vehicle's velocity response to a two-steps input

Inertia formula

From *Figure 7.10*, it is possible to neglect the motor's armature inductance, L_a, which is very small before the armature's resistance R_a, see *Appendix A*.

The derived simplified transfer function of the motorshaft's angular velocity, relative to the voltage input is then:

$$\frac{\omega(\mathrm{s})}{\mathrm{U_a(s)}} = \frac{\frac{\mathrm{K_t}}{\mathrm{R_a}} \cdot \frac{\frac{1}{\mathrm{B_{tot}}}}{\frac{\mathrm{J_{tot}}}{\mathrm{B_{tot}}}\cdot \mathrm{s}+1}}{1 + \frac{\mathrm{K_t}\cdot\mathrm{K_e}}{\mathrm{R_a}} \cdot \frac{\frac{1}{\mathrm{B_{tot}}}}{\frac{\mathrm{J_{tot}}}{\mathrm{B_{tot}}}\cdot \mathrm{s}+1}} \tag{12.11}$$

Where:

J_{tot}	is the vehicle's total inertia	$[\mathrm{kg}\cdot\mathrm{m}^2]$
B_{tot}	is the vehicle's total friction	$[\mathrm{N}\cdot\mathrm{m}\cdot\mathrm{s}]$
R_a	is the motor's armature resistance	$[\Omega]$
K_t	is the motor constant	[Wb]
K_e	is the motor's generator constant	[Wb]

This can be rewritten in the standard form as :

$$\frac{\omega(\mathrm{s})}{\mathrm{U_a(s)}} = \frac{\frac{\mathrm{K_t}}{\mathrm{B_{tot}}\cdot\mathrm{R_a}+\mathrm{K_t}\cdot\mathrm{K_e}}}{\frac{\mathrm{J_{tot}}\cdot\mathrm{R_a}}{\mathrm{B_{tot}}\cdot\mathrm{R_a}+\mathrm{K_t}\cdot\mathrm{K_e}}\cdot \mathrm{s}+1} \tag{12.12}$$

That means that the time constant of the entire velocity model, τ, can be expressed as:

$$\tau = \frac{\mathrm{J_{tot}}\cdot\mathrm{R_a}}{\mathrm{B_{tot}}\cdot\mathrm{R_a}+\mathrm{K_t}\cdot\mathrm{K_e}} \quad [\mathrm{s}] \tag{12.13}$$

Eventually, since the time constant is found by use of the system step response, it is possible to rearrange *Equation (12.13)* to obtain $\mathrm{J_{tot}}$:

$$\mathrm{J_{tot}} = \tau\cdot\left(\mathrm{B_{tot}} + \frac{\mathrm{K_t}\cdot\mathrm{K_e}}{\mathrm{R_a}}\right) \quad \left[\mathrm{kg}\cdot\mathrm{m}^2\right] \tag{12.14}$$

Setup

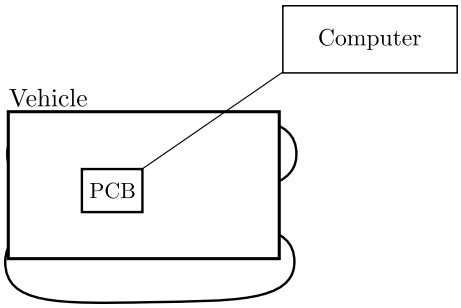


Figure 12.21: Setup diagram

List of Equipment

Instrument	Type
Computer	Asus N73JN

Procedure

1. Disconnect the battery.
2. Connect the Arduino to the computer.
3. Upload the test code to the Arduino board using the Arduino IDE [12].
4. Open a serial terminal via PuTTY [49].
5. Plug in the battery immediately after opening the terminal.
6. Wait two seconds, then follow the vehicle with the connected computer.
7. Wait until the vehicle stops before ending the measurements by unplugging the connected computer from the Arduino.
8. Plot the speed of the vehicle using Matlab.

Results

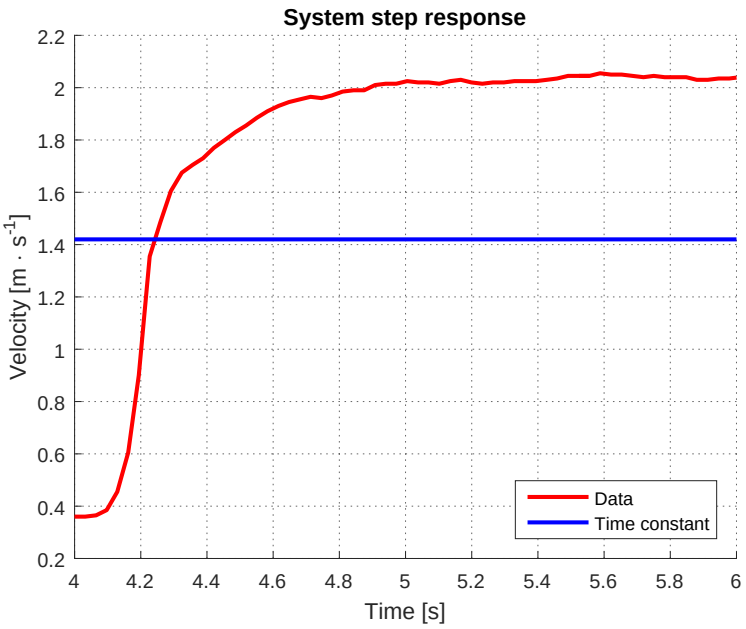


Figure 12.22: Time constant on a step-response

In this test, a first voltage step is applied to the motor so that the vehicle runs at a minimal speed. A second bigger step is then applied. The analyzed data is the one retrieved between these two steps and it can be seen on *Figure 12.22*. A blue line is drawn at 63,2% of the final velocity to represent the time constants.

Velocity System's Time Constant

Only the second step is used. By knowing the maximum velocity of the vehicle at the end of the test and the velocity when it is in a first steady state, just before the second step, it is possible to deduce the time at which the velocity reaches 63,2% of its final value. The obtained time is the step response time constant of the system τ.

Since the velocity is taken at 63,2% between the first steady state and the final value, it is needed to shift it up with the starting value V_{ss1} :

$$V(\tau + t_0) = (V_{max} - V_{ss1}) \cdot 0,632 + V_{ss1} \qquad \left[m \cdot s^{-1}\right] \quad (12.15)$$

Where:

$V(\tau + t_0)$	is the velocity at 63,2% of the final velocity	$[m \cdot s^{-1}]$
t_0	is the time at which the second step starts	[s]
V_{max}	is the maximum velocity	$[m \cdot s^{-1}]$
V_{ss1}	is the velocity in steady state corresponding to the first step	$[m \cdot s^{-1}]$

According to the data, the velocity of the vehicle at that point is :

$V(\tau + t_0) = (2,04 - 0,36) \cdot 0,632 + 0,36 \qquad V(\tau + t_0) = 1,42 \text{ m} \cdot \text{s}^{-1}$

The corresponding time stamp to that velocity t_1 is:

$t_1 = 4,243$ s

However, it is also needed to shift down the time scale so that the time t_0 at which the second step begins is the time origin :

$$\tau = t_1 - t_0 \qquad [s] \quad (12.16)$$

Therefore, the time constant is:

$\tau = 4,243 - 4,032$

$\tau = 0,211$ s

Total Inertia Calculation

Eventually, the *Equation (12.14)* is used to calculate J_{tot}. Using values found in previous tests leads to J_{tot} being :

$J_{tot} = 4,5782 \cdot 10^{-6}$ kg $\cdot$ m^2

L | Vehicle Tests - Velocity Gain

Name: Group 510
Date: 13/11 - 2015

Purpose

The purpose of this test is to find the vehicle's velocity gain.

Setup

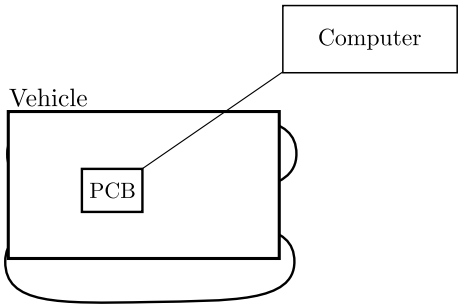


Figure 12.23: Setup diagram

List of Equipment

Instrument	Type
Computer	HP 8460P

Procedure

1. Disconnect the battery.
2. Connect the Arduino to the computer.
3. Upload the test code to the Arduino board using the Arduino IDE [12].
4. Open a serial terminal via PuTTY [49].
5. Plug in the battery immediately after opening the terminal.
6. Wait two seconds, then follow the vehicle with the connected computer.
7. Wait until the vehicle stops before ending the measurements by unplugging the connected computer from the Arduino.
8. Plot the speed of the vehicle using Matlab.

Results

To get as close as possible to the vehicle's exact gain, a test where the vehicle drives at different speed has been set up. In *Figure 12.24* the data from the performed test is illustrated.

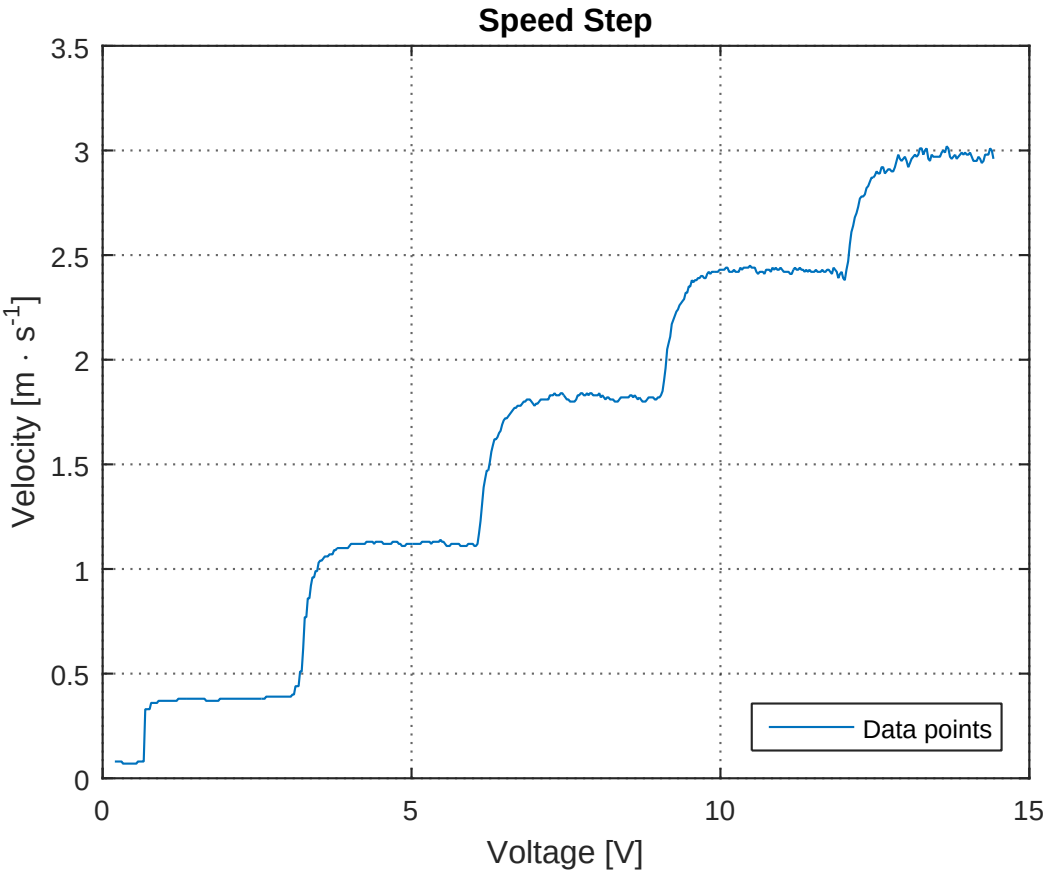


Figure 12.24: A plot of the measured speed as the speed steps are performed.

From *Figure 12.24* it is seen that for each two seconds the vehicle's velocity is increased by 20%. By taking the average value of the vehicles steady state speed at each velocity, it is possible to get a gain of the vehicle, this is illustrated in *Figure 12.25*.

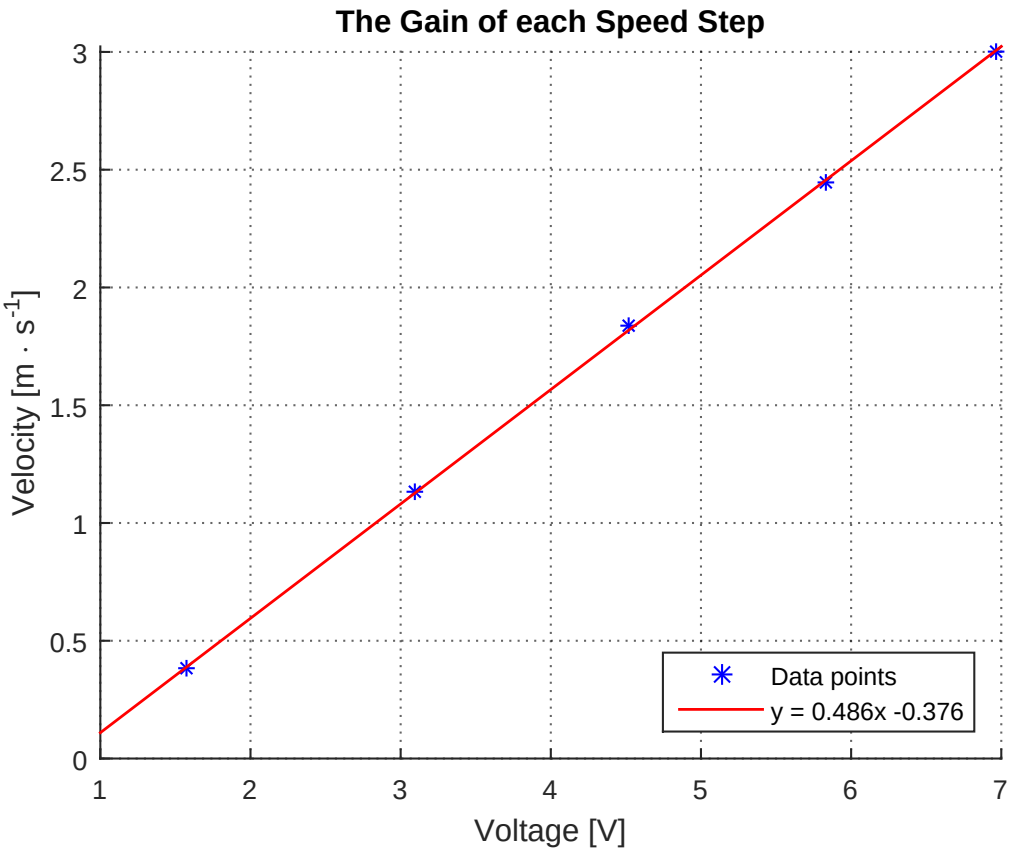


Figure 12.25: A plot of the calculated gain of each speed step, with a red least square line

By making a least square line going through the average gain value at each velocity, it is possible to get the gain and the offset created by the stiction. The equation for the least square line is seen on *Figure 12.25*. The Gain and the offset created by the stiction is therefore:

$$K_G = 0,485 \text{ m} \cdot \text{s}^{-1} \cdot \text{V}^{-1} \tag{12.17}$$

$$\Delta v = 0,376 \text{ m} \cdot \text{s}^{-1} \tag{12.18}$$

M | Vehicle Test - Proportional Controller Test

Name: Group 510
Date: 06/10 - 2015

Purpose

The purpose of this test is to find out how the system reacts to closed loop control.

Theory

According to [42], a P-controller makes it possible to change the natural frequency, and thereby the time constant of a system. To test this, a P-controller with a gain of 1 is implemented, and compared to a normal step response. A side effect of the Proportional control is a steady state offset. This can be easily shown by an example:

A non moving system, with a desired final speed of 2 $m \cdot s^{-1}$, and a P-gain of 1 is examined.

In the beginning, the error will be calculated as 2 - 0 = 2 $m \cdot s^{-1}$. The system is therefore instructed to accelerate to 2 $m \cdot s^{-1}$.

After a period of time, the system will cross 1 $m \cdot s^{-1}$. Calculating the error now, results in 2 - 1 = 1 $m \cdot s^{-1}$ The system is therefore instructed to stay at 1 $m \cdot s^{-1}$, and stop accelerating. This effectively causes a steady state error of 50%.

A way to reduce this offset is to use Feed Forward. [**Aastrom**]. This will be tested as well, with a Feed Forward gain of 1.

Setup

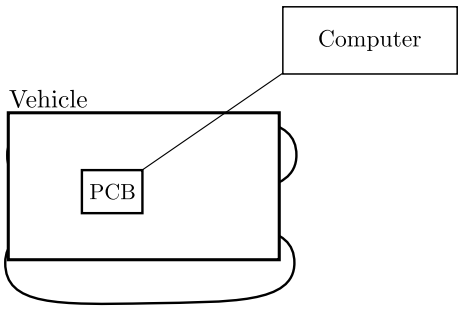


Figure 12.26: Setup diagram

List of Equipment

Instrument	Type
Computer	HP 8460P

Procedure

1. Disconnect the battery.
2. Connect the Arduino to the computer.
3. Upload the test code to the Arduino board using the Arduino IDE [12].
4. Open a serial terminal via PuTTY [49].
5. Plug in the battery immediately after opening the terminal.
6. Wait two seconds, then follow the vehicle with the connected computer.
7. Wait until the vehicle stops before ending the measurements by unplugging the connected computer from the Arduino.
8. Plot the speed of the vehicle using Matlab.

Results

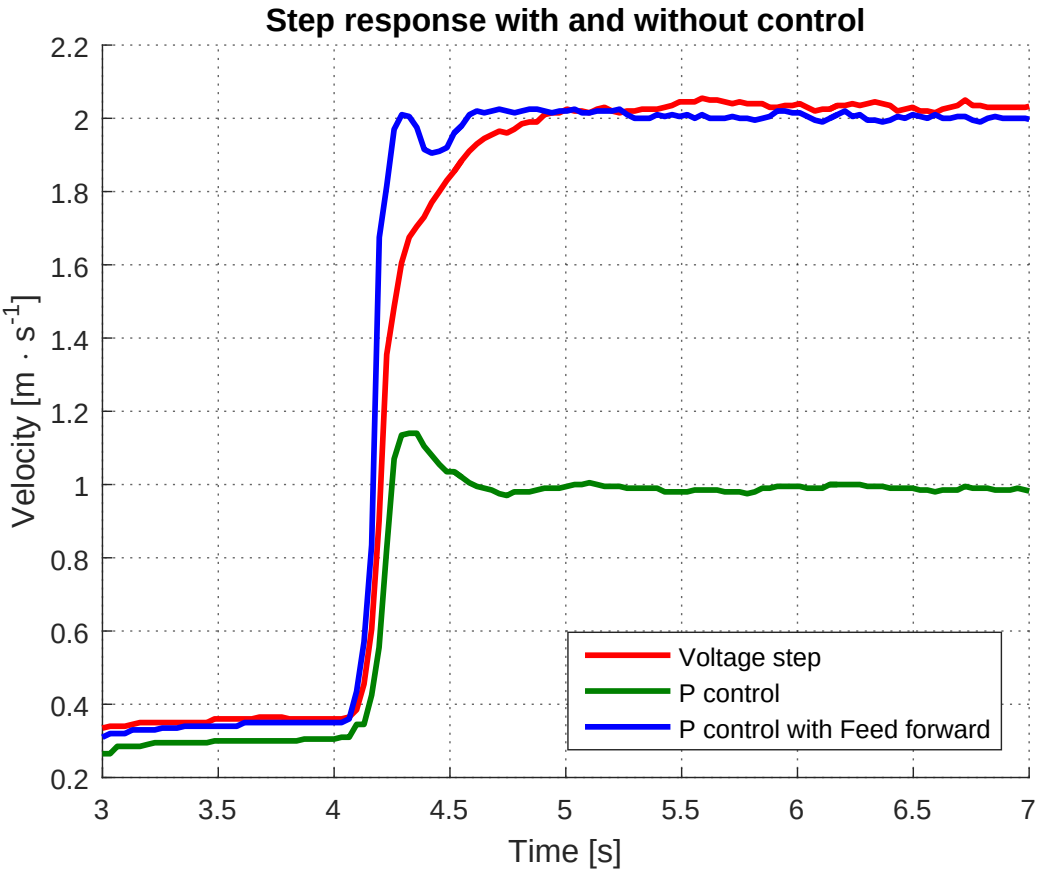


Figure 12.27: A plot illustrating the step responses with and without P control

In all tests, the vehicle was instructed to change its speed from 0,3 $\mathrm{m} \cdot \mathrm{s}^{-1}$ to 2 $\mathrm{m} \cdot \mathrm{s}^{-1}$. The resulting responses of the system can be seen in *Figure 12.27*. It can be clearly seen, that the P-controller creates an offset of 50%, just as expected. The feed forward eliminates this error. The three time constants can now be reviewed:

$$\tau_\mathrm{s} = 0{,}211\,\mathrm{s}$$

$$\tau_\mathrm{p} = 0{,}152\,\mathrm{s}$$

$$\tau_\mathrm{f} = 0{,}152\,\mathrm{s}$$

Where:

τ_s	is the time constant for the open loop response	[s]
τ_p	is the time constant for the response with a P-controller	[s]
τ_f	is the time constant for the response with a P-controller and feed forward	[s]

It can be concluded, that the P controller lowers the time constant of the system. Furthermore it is clear, that the feed forward does not affect the time constant, just the final speed.

N | Vehicle Tests - Steering - Steering Gain

Name: Group 510
Date: 29/11 - 2015

Purpose

The test has two purposes:

1. To find the relationship between the vehicle's velocity and its steering. This relationship is described as a steering gain, $K_{steering}$.
2. To find a range of speeds where this relationship is linear.

Theory

As seen on *Section 7.2*, *Figure 7.18*, the steering behavior is described with two gain coefficients, K_s and K_v. Since their respective value are not needed independently, it is possible to combine them to find a value more easily. The steering gain, $K_{steering}$, is therefore defined as :

$$K_{steering} = K_s \cdot K_v$$

Moreover, to get reliable data from which to draw conclusions, a basic proportional controller is implemented to stabilize the angle and realize a step response, as seen on *Section 8.2*, *Figure 8.13*. However, this model is simplified by ignoring the servomotor's delay. Since the test is only a one step response, this delay should only appear once between the step command to the servo and the actual reaching of the desired heading. Furthermore, since the two summation blocks add a $\pm K_{off}$, the offset in the microcontroller cancels the offset from the plant. This way, the model used in this test includes the controller, K_p, the steering gain, $K_{steering}$, and the integrator which allows to get headings out of the vehicle's angular velocity, see *Figure 12.28*.

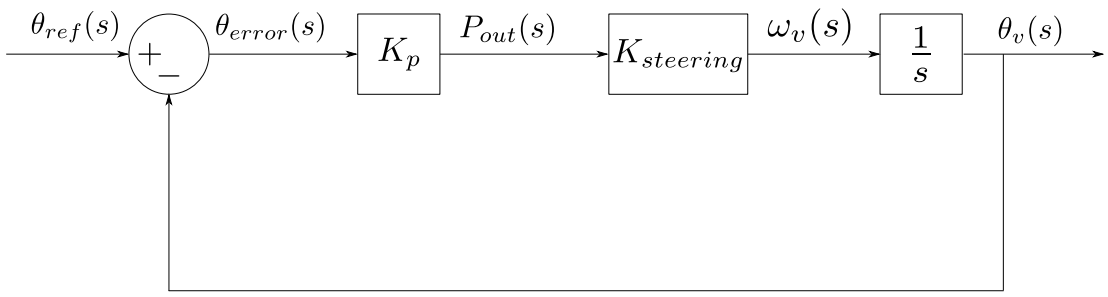


Figure 12.28: A diagram of the test setup

The resulting transfer function of this system can be expressed in standard form as:

$$\frac{\theta(s)}{\theta_{ref}(s)} = \frac{1}{\frac{1}{K_p \cdot K_{steering}} \cdot s + 1} \tag{12.19}$$

By applying a known reference heading with a fixed gain in the controller and using a steady velocity, the measured parameters are the real heading, real velocity and the timestamp of each measurement point.

Setup

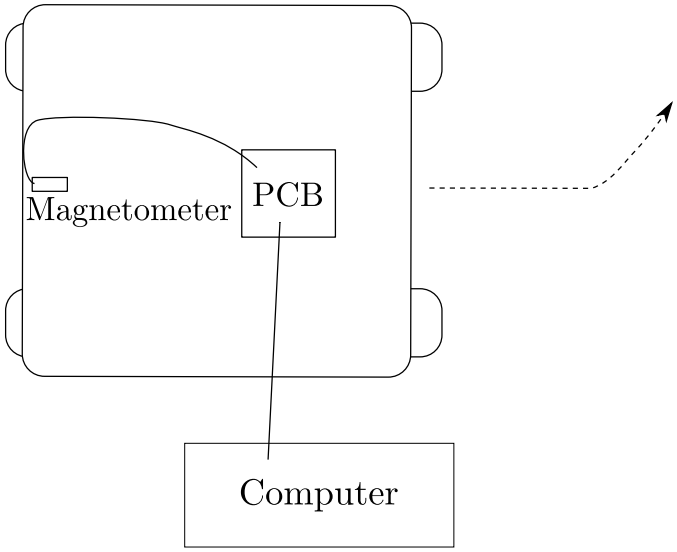


Figure 12.29: A diagram of the test setup

List of Equipment

Instrument	Type
Computer	Acer C720p

Procedure

1. Make sure the vehicle's power is off.
2. Connect the Arduino to the computer.
3. Upload the test code to the Arduino board using the Arduino IDE [12].
4. Place the vehicle with an orientation of approximately 0 °, i.e. heading North.

5. Put the general power on.
6. Open a serial terminal via PuTTY [49] immediately after plugging the power in.
7. Wait two seconds, then follow the vehicle with the connected computer.
8. Wait until the vehicle stops before ending the measurements by unplugging the connected computer from the Arduino.
9. Shut the vehicle's power off.
10. Redo the test multiples time while increasing the wanted speed.
11. Plot the angle of the vehicle using Matlab.

Results

The vehicle is made to run for 2000 ms with a reference heading of 0°. The vehicle shall try to head towards the exact magnetic North within these two seconds. After that, the reference angle is put to −45° (East) which means the vehicle turns to the right to reach this heading. It runs for 15s.

From *Equation (12.19)*, the time constant is found as:

$$\tau = \frac{1}{\mathrm{K_p} \cdot \mathrm{K_{steering}}} \quad [\mathrm{s}] \quad (12.20)$$

From this equation and by findig the time constant in each test, it will then be possible to calculate $\mathrm{K_{steering}}$:

$$\mathrm{K_{steering}} = \frac{1}{\mathrm{K_p} \cdot \tau} \quad \left[\mathrm{rad} \cdot \mathrm{s}^{-2}\right] \quad (12.21)$$

The different steering time constants are obtained thanks to the measurement tests. The value of τ corresponds to the time when the angle reaches $63,2\%$ of the final value. A plot of these tests can be seen on *Figure 12.30*.

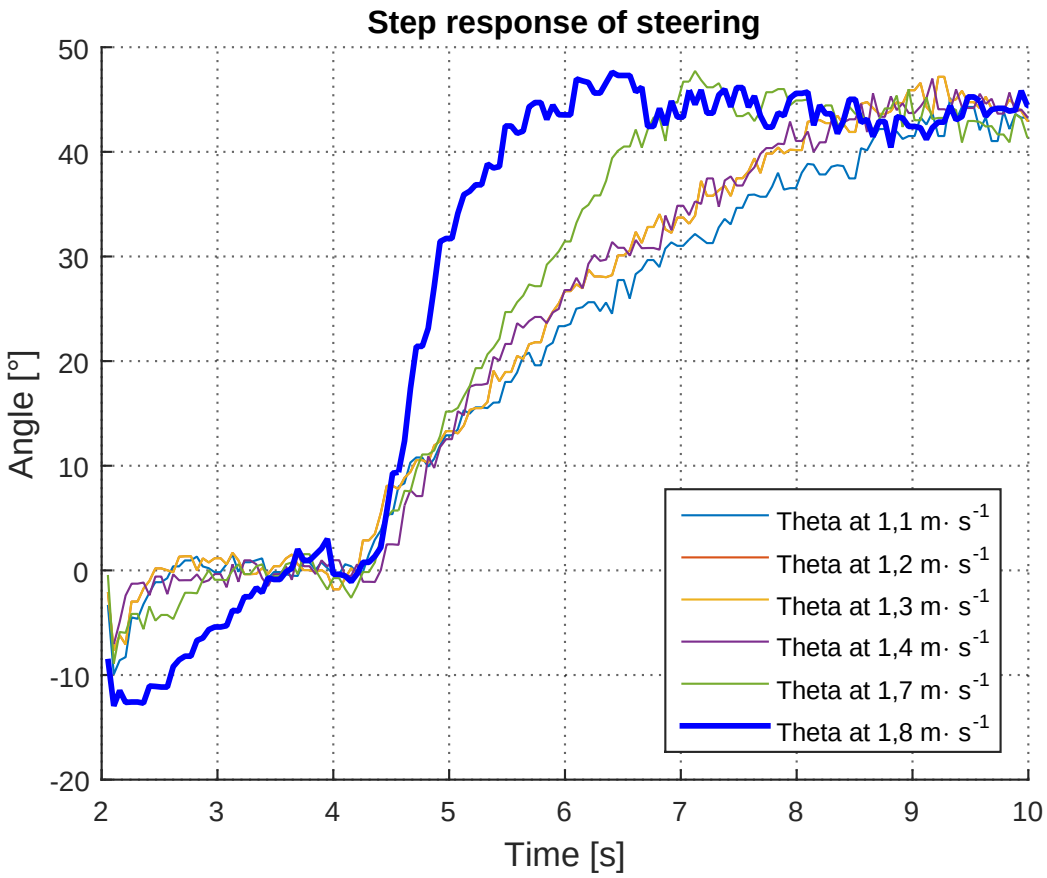


Figure 12.30: Plot of multiples step responses for finding the time constant τ relative to a specific speed.

As the test were made with a choosen control gain $K_p = 1,5$, the steering gain is calculated and plotted against the real speed of the vehicle during the turnings, see *Figure 12.31*.

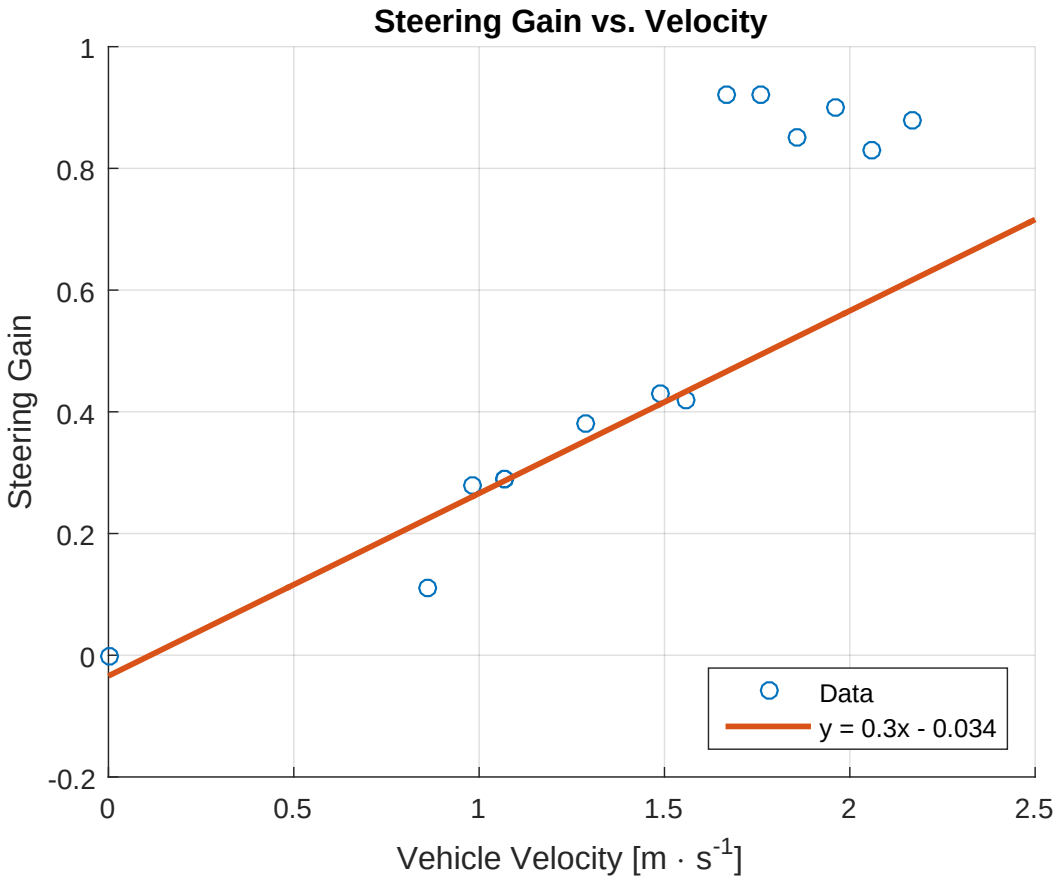


Figure 12.31: Plot of the gain K_v against the real speed.

From this plot a linear region of the gain proportional to the speed can be established:

$$K_{steering} = 0,3 \cdot v - 0,034 \tag{12.22}$$

Observing the data, speeds between 1,0 and 1,6 $m \cdot s^{-1}$ can be interpreted as linear.

O | Vehicle Tests - Steering - Linear Area for K_p

Name: Group 510
Date: 30/09 - 2015

Purpose

The purpose of this test is to find a minimum and maximum values that could be used for the steering proportional controller's gain, K_p.

Setup

The setup is the same as in the steering gain test, see *Appendix N*.

List of Equipment

Instrument	Type
Computer	Acer C720p

Procedure

1. Disconnect the battery.
2. Connect the Arduino to the computer.
3. Upload the test code to the Arduino board using the Arduino IDE [12].
4. Switch the vehicle's power on.
5. Open a serial terminal via PuTTY [49] immediately after plugging the battery.
6. Wait two seconds, then follow the vehicle with the connected computer.
7. Wait until the vehicle stops before ending the measurements by unplugging the connected computer from the Arduino.
8. Shut the vehicle's power off.
9. Redo the test multiples while increasing the wanted controller gain.
10. Plot the angle of the vehicle using Matlab.

Results

In this test, instead of changing the velocity between tests, the proportional controller's gain is changing from 1 to 4 with a step of 0,5.

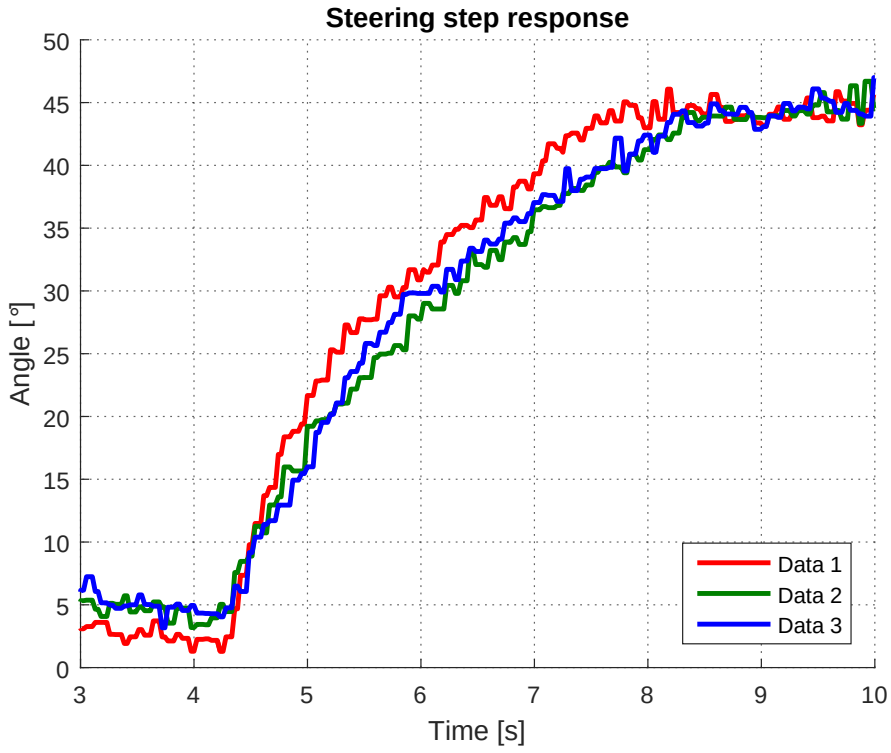


Figure 12.32: Plot of three test where the vehicle is turning, where the x-axis is time and the y-axis is angle.

By utilizing 18 measurements, of which three are illustrated in *Figure 12.32*, 18 time-constants are found on the graph. The corresponding values for the steering gain, $K_{steering}$, are calculated by using the formula from *Equation (12.21)*, see *Table 12.1*.

Given the fact that the velocity should be maintained constant, the steering gain, $K_{steering}$ should remain the same as long as the proportional controller's gain, K_p is within the right range. Outside this particular area, the steering might become non-linear and this should therefore be avoided as much as possible to simplify the controlling the process.

Test no.	K_p	$K_{steering}$	Test no.	K_p	$K_{steering}$
1	1	0.180831826	10	2.5	0.344827586
2	1	0.180342651	11	2.5	0.294117647
3	1	0.17088175	12	2.5	0.291970803
4	1.5	0.266666667	13	3	0.362318841
5	1.5	0.259403372	14	3	0.256410256
6	1.5	0.254452926	15	3.5	0.840336134
7	2	0.265957447	16	3.5	0.549450549
8	2	0.284090909	17	3.5	0.476190476
9	2	0.37037037	18	4	0.549450549

Table 12.1: Steering gains calculated from the tests

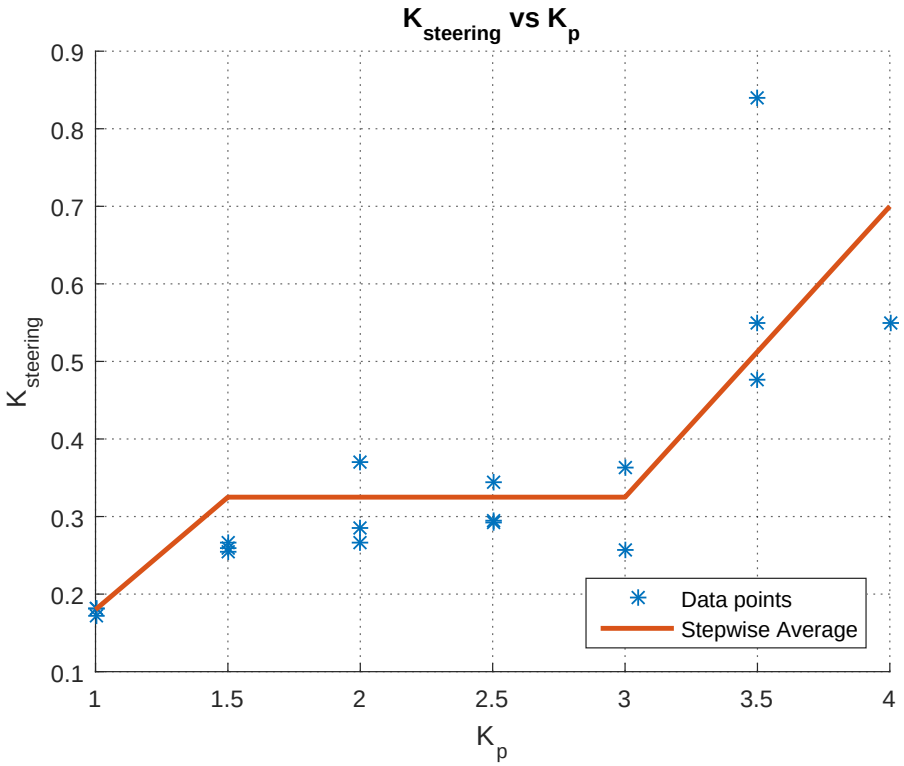


Figure 12.33: Plot of the gain K_v against the real speed.

The actual values from the test are plotted on *Figure 12.33*. It can be seen on this graph, that the steering gain is constant only for values of K_p from approximately $1,5$ to 3. The choice of a proportional controller gain should therefore account for this and stay within this linear area.

P | Schematics

Arduino Mega Pinout

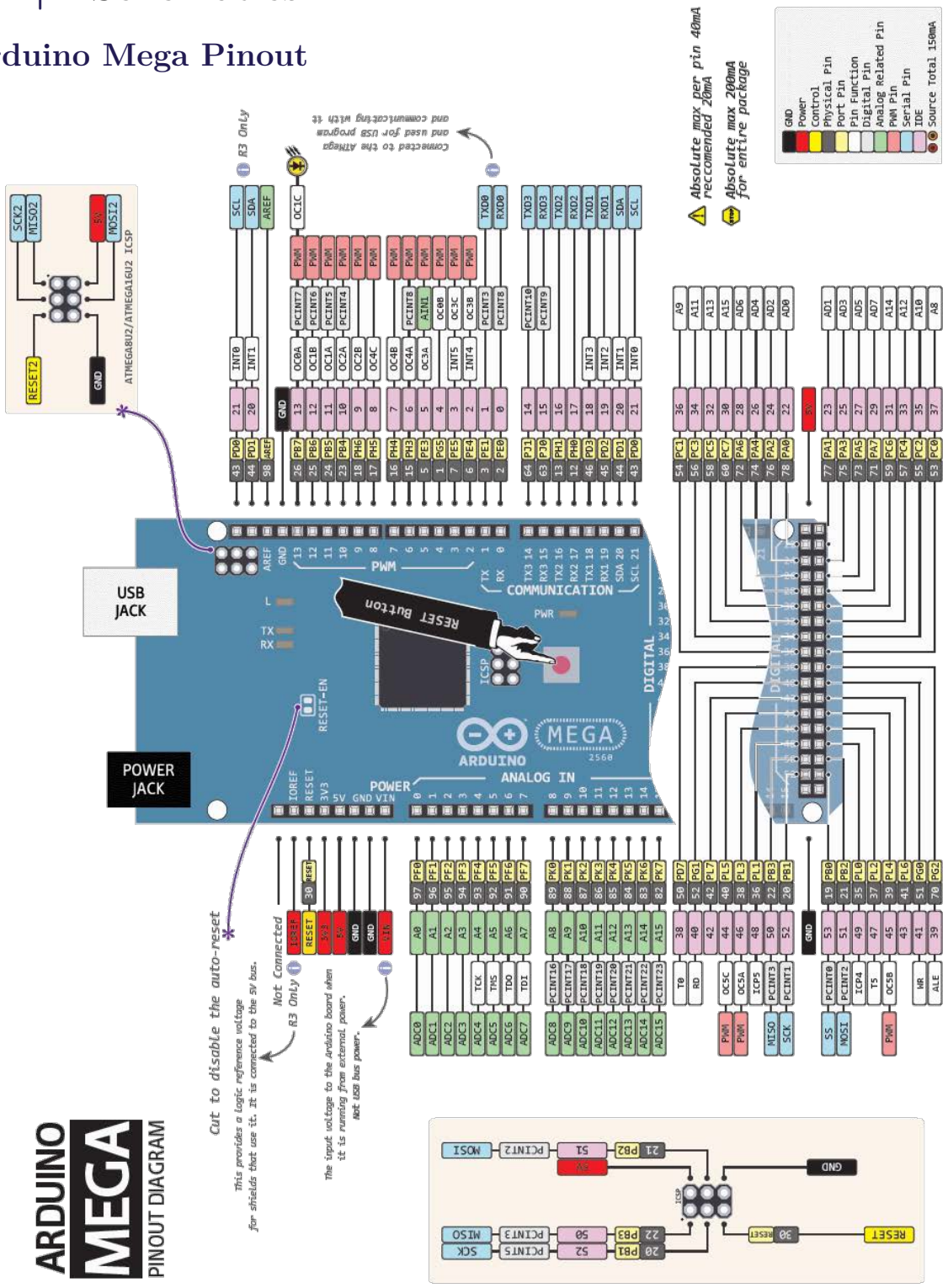


Figure 12.34: Arduino Mega Pinout [23]

Electrical schematic

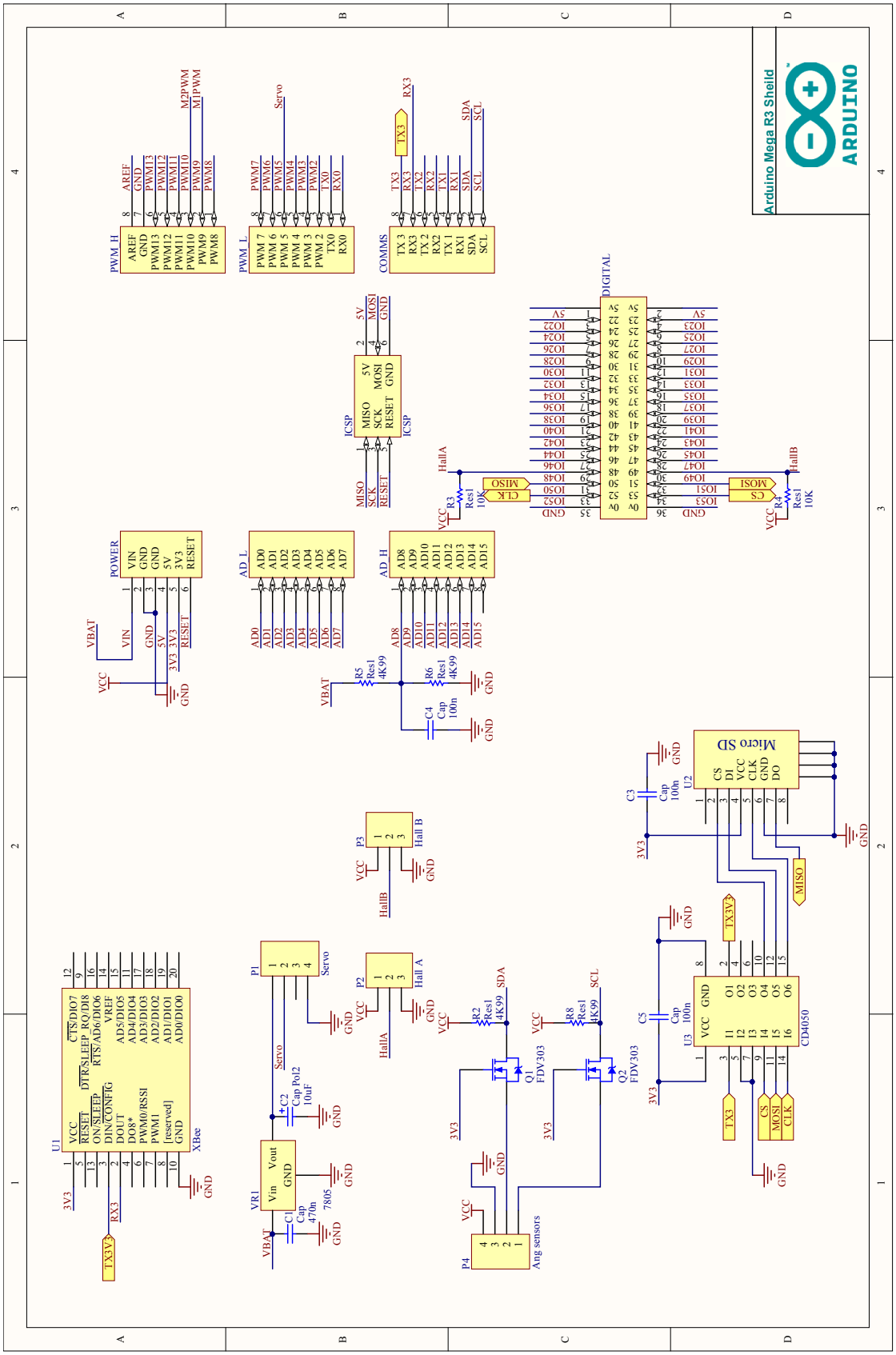


Figure 12.35: Electrical schematic

Bibliography

[1] Bosch, ed. *Bosch Indego*. URL: https://www.bosch-garden.com/gb/en/garden-tools/indego-logicut.jsp (visited on 12/15/2015).

[2] Robomow. *Operating Manual*. 2014. URL: http://robomow.com/wp-content/themes/robomow/images/user-manual/user-manual-rcmc-en-GB.pdf.

[3] GamesOnTrack A.S, ed. *Games on Track GT-Position*. 2012-2015. URL: http://www.gamesontrack.co.uk/pages/webside.asp?articleGuid=64556.

[4] Navigation USA National Coordination Office for Space-Based Positioning and Timing, eds. *GPS Accuracy*. 2014. URL: http://www.gps.gov/systems/gps/performance/accuracy/.

[5] Keith M. Miller. „A Review of GLONASS“. In: *Hydrographic Journal* (2000).

[6] *Construction working of differential assembly*. July 2011. URL: http://mechanicalmania.blogspot.dk/2011/07/construction-working-of-differential.html.

[7] *Futaba S3003*. Oct. 23, 2015. URL: http://www.servodatabase.com/servo/futaba/s3003 (visited on 11/26/2015).

[8] Kim Hylling Sørensen et al. *Measurement Car*. Technical Report. Aalborg University, Dec. 20, 2012.

[9] *NiMH RC - Battery Pack 7.2V 470011mAh*. May 14, 2012. URL: http://www.farnell.com/datasheets/1722828.pdf.

[10] Arduino, ed. *Arduino Mega Specifications*. URL: https://www.arduino.cc/en/Main/ArduinoBoardMega2560.

[11] gvz. *avrdude Github Repository*. 2000. URL: https://github.com/gvz/avrdude (visited on 12/04/2015).

[12] Arduino, ed. *Arduino IDE*. October 2015. URL: https://www.arduino.cc/en/Main/Software.

[13] SD Association. *SD standards*. URL: https://www.sdcard.org/developers/overview/index.html.

[14] elasticsheep. *Reading an SD card with an ATMEGA168*. URL: http://elasticsheep.com/2010/01/reading-an-sd-card-with-an-atmega168/.

[15] *Pololu Dual VNH5019 Motor Driver Shield User's Guide*. 2014.

[16] Pololu. *Pololu Dual VNH5019 Motor Driver Shield for Arduino*. URL: https://www.pololu.com/product/2502.

[17] STMicroelectronics. *Automotive fully integrated H-bridge motor driver*. Datasheet. Version Rev 9. VNH5019A-E. Sept. 19, 2013.

[18] Digi International. *Specs on the XBee DigiMesh 2.4*. URL: http://www.digi.com/products/xbee-rf-solutions/modules/xbee-digimesh-2-4#specifications.

[19] jeromeabel. *Xbee and Arduino setup*. URL: http://jeromeabel.net/ressources/xbee-arduino.

[20] Digi International. *XBEE 802.15.4 RF MODULES*. English. 2015. URL: http://www.digi.com/pdf/ds_xbeemultipointmodules.pdf (visited on 12/16/2015).

[21] Sparkfun, ed. *SparkFun 9 Degrees of Freedom - Sensor Stick*. URL: https://www.sparkfun.com/products/10724 (visited on 12/15/2015).

[22] Ken Shirriff. *Secrets of Arduino PWM*. Ed. by Arduino. URL: https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM.

[23] Atmel, ed. *Atmega 2560 datasheet*. Feb. 2014. URL: http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf.

[24] Leandro. *Information on the Arduino Mega*. URL: http://saber.patagoniatecnology.com/arduino-mega-2560-atmega-mega-arduino-clon-compatible-argentina-tutorial-basico-informacion-arduino-argentina-ptec/.

[25] Sparkfun. *Accelerometer, Gyro and IMU Buying Guide*. 2014. URL: https://www.sparkfun.com/pages/accel_gyro_guide.

[26] Siemens. *TLE4905 datasheet*. 1992. URL: http://www.komponenten.es.aau.dk/fileadmin/komponenten/Data_Sheet/Linear/TLE4905.pdf.

[27] ST. *L4941 datasheet*. 2004. URL: http://www.komponenten.es.aau.dk/fileadmin/komponenten/Data_Sheet/Spaendingsreg/L4941.pdf.

[28] NXP. *HEF4050B datasheet*. 2011. URL: http://www.nxp.com/documents/data_sheet/HEF4050B.pdf.

[29] Philips. *Bi-directional level shifter for I2C-bus and othersystems*. 1997. URL: https://www.adafruit.com/datasheets/an97055.pdf.

[30] Fairchild Semiconductor. *FDV303N datasheet*. 1997. URL: http://www.farnell.com/datasheets/276083.pdf.

[31] David Cook. *Intermediate Robot Building*. Technology in action series. Apress, 2010. ISBN: 9781430227540. URL: https://books.google.dk/books?id=08cxDBE2icUC (visited on 12/15/2015).

[32] Biezl. *Vierquadrantensteller*. Oct. 23, 2010. URL: https://commons.wikimedia.org/wiki/File:Vierquadrantensteller.svg (visited on 12/15/2015).

[33] *Pulse Width Modulation*. Slides. Dec. 16, 2015.

[34] *3-Axis Digital Compass IC HMC5883L*. English. Version 900405 Rev E. Feb. 2013. URL: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf (visited on 12/05/2015).

[35] Microsoft support. *The OSI Model's Seven Layers Defined and Functions Explained*. 2014. URL: https://support.microsoft.com/en-us/kb/103884.

[36] IEEE Computer Society. *IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*. English. Apr. 17, 2009. (Visited on 12/16/2015).

[37] Jens Myer. *Introduction to communication networks*. Presentation. Slides located on CD. 2015.

[38] *Analogue and Digital Control 1*. Oct. 2, 2015. URL: http://www.moodle.aau.dk/ (visited on 08/12/2015).

[39] William S. Levine. *The Control Handbook*. Taylor & Francis, 1996. ISBN: 9780849385704.

[40] Karl Johan Åström and Tore Hägglund. *Advanced PID Control*. ISA-The Instrumentation, Systems, and Automation Society, 2006. ISBN: 9781556179426.

[41] Bela G. Liptak. *nstrument Engineers' Handbook, Fourth Edition, Volume Two: Process Control and Optimization*. Instrument Engineers' Handbook. CRC Press, 2005. ISBN: 9781420064001.

[42] Emami-Naeini Franklin Powell. *Feedback Control of Dynamic Systems*. 2014.

[43] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. 2014.

[44] Ole kiel Jensen and Ming Shen. *EIT/ITC Signal Processing*. Presentation. Slides located on CD. 2015.

[45] Lyons R.G. *IMPULSE INVARIANCE IIR FILTER DESIGN METHOD*. 2013. URL: http://flylib.com/books/en/2.729.1.69/1/.

[46] *MWA-W8-1-P*. Oct. 9, 2015. URL: http://www.icom-memo.de/PDF%20Kataloge/MWAs/Dyn.%20MWAs/Dyn.%20MWAs%20zyl.We/MWA-W8-P.pdf.

[47] *Mechanical systems II*. Slides. Nov. 9, 2015.

[48] Yury Matselenak. *3-Axis Digital Compass IC HMC5883L*. English. June 1, 2014. URL: http://diydrones.com/profiles/blogs/advanced-hard-and-soft-iron-magnetometer-calibration-for-dummies (visited on 12/08/2015).

[49] PuTTY.org, ed. *PuTTY software*. October 2015. URL: http://www.putty.org/.