Development, Modelling and Control of an Agricultural Robot

Morten Westeraa

Aalborg University, June 2019

Master Thesis

This report has been typeset in mathpazo and formatted in Latex, using an Aalborg University report template by Jesper Kjær Nielsen. Photographs are either produced by the author or found online. The graphic figures are generated in Matlab or on www.draw.io. The programming of the software was done in Matlab.

Contol & Automation
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Development, Modelling and Control of an Agricultural Robot

Theme:
Modelling and Control of mobile robots

Project Period:
Spring Semester 2019

Author:
Morten Westeraa

Supervisor:
Karl Damkjær Hansen

Page Numbers: 102

Date of Completion:
June 5, 2019

Abstract:

Agricultural operations requires a lot of labour. A solution could be the use of small farm robots that can run around the clock and solve a lot of the jobs at hand. This could e.g. be harrowing, planting. spraying etc. This work describes the development of a control scheme for a versatile agricultural robot, that can solve a lot of different tasks. The project includes the use of a robot prototype, build prior to this project, where focus is on setting up hardware and communication with motor drives via CAN-bus. The project also includes an approach to mathematical modelling of the robot and a description of the sensor suite used to determine position and orientation in a field. Finally the development of a trajectory following controller is discussed.

Contents

Preface

This Master Thesis was written during the spring semester 2019, as a 10^{th} semester project in the education "Control & Automation" at Aalborg University. It not only marks the end of the semester, but also the end of a six year long journey, that I started back in 2013, to pursue a Masters degree in Engineering. I would like to send deep thanks to my family and friends for moral support and encouragement, also in times when the outcome seemed less achievable than now. In this last semester I have enjoyed the supervision of Karl Damkjær Hansen, whom I would also like to offer thanks for his support and our numerous meetings, where we discussed not only engineering related topics, but also sometimes life in general. Also my fellow students deserves a special acknowledgement. Our daily interaction and technical discussions have been a big help to me and served as welcomed breaks during our common struggle. I enjoyed our time together.

Morten Westeraa
Aalborg University, June 5, 2019

Nomenclature

Symbol	Description	Unit
v	Robot Linear velocity	m/s
ω	Robot angular velocity	rad/s
v_l	Angular velocity - left wheelpair	rad/s
v_r	Angular velocity - right wheelpair	rad/s
r	Wheel radius	m
S	Global coordinate system	.
x	x-position in global coordinate system	m
y	y-position in global coordinate system	m
θ	Robot heading in global coordinate system	rad
l	Robot wheeltrack	m
m	Robot mass	kg
k_p	Proportional coefficient	$N/m/s$
k_i	Internal coefficient	N
F_v	Force needed to move robot in linear direction	N
τ	Summed axle torque	Nm
i	Wheel number	$1-4$
g_r	Gear ratio - motor/axle	.
k_τ	Torque/current ratio	.
I	Summed current	$Amp.$
τ_ω	Angular torque	Nm
a	Vector from wheels to centre of vehicle	.
F_{res}	Resulting vector	.
T	Transformation matrix	.
τ_c	Counter torque	Nm
$F_{f\omega}$	Sliding force	N
F_n	Normanl force	N
μ_d	Friction coefficient	.
v_ω	Ground velocity	m/s
F_{fl}	Longitudinal resistance force	N
γ	Moment of inertia	kgm^2
I_m	Motor current	$Amp.$
x_g	x-position, GPS	m
y_g	y-position, GPS	m
θ_g	Heading, GPS	rad

Symbol	Description	Unit
x_r	Raw data, x-direction, magnetometer	.
y_r	Raw data, y-direction, magnetometer	.
x_{max}	Maximum value, x-direction, magnetometer	.
x_{min}	Minimum value, x-direction, magnetometer	.
y_{max}	Maximum value, y-direction, magnetometer	.
y_{min}	Mimimum value, y-direction, magnetometer	.
M_x	unit vector, x-direction, magnetometer	.
M_y	unit vector, y-direction, magnetometer	.
θ_m	Heading, magnetometer	rad
o	Odometry on each wheel	m
T_e	Ticks from drive encoder	.
T_r	Ticks per revolution	.
C	Curcumference, wheels	m
o_l	Odometry left side	m
o_r	Odometry right side	m
θ_o	Heading, odometry	rad
x_o	x-position, odometry	m
y_o	y-position, odometry	m
x_r	Reference x-position	m
yr	Reference y-position	m
θ_r	Reference heading	rad
n	Trajectory index	.
e_f	Frame error	$[m, m, rad]^T$
p	Position vector	m
d	Direction vector	$[\cos(rad), \sin(rad)]^T$
p_r	Reference position vector	m
d_r	Reference direction vector	$[\cos(rad), \sin(rad)]^T$
R	Rotation matrix	.
v_r	Reference velocity	m/s
ω_r	Reference angular velocity	rad/s
p_{ex}	Position error in x-direction	m
p_{ey}	Position error in y-direction	m
d_{ex}	Direction error, x-component	$\sin(rad)$
d_{ey}	Direction error, y-component	$\cos(rad)$
Φ	State vector	.
u	Input vector	.
K	Feedback gain matrix	.
Q	Weight matrix, state	.
R	Weight matrix, input	.

1 Introduction

This master thesis will focus on development of a mobile robot that can be used for farming applications. In farming operations, a lot of hours are spend on cultivating the soil, preparing seedbed, nursing plants and harvest. These operations are, at present time, carried out by big heavy machinery. Although this machinery has increased very much in efficiency over the last century, the field work is still very time consuming and a task that, every year, requires a lot of labour around the globe. Furthermore, the use of big and heavy machinery carries the risk of compaction of the soil, that can impact the possibility of optimal plant growth, leading to lower yields.

The implementation of small agricultural robots that can handle some of the aforementioned tasks, could be a solution. Such small robots should be able to autonomously work the fields by carrying or pulling tools fitted to the job at hand. These robots should use an adequate sensor suite for determining position and orientation, combined with sensors that can guarantee a high safety level. The focus of this project will be on development, modelling and control of an agricultural robot, that can solve some of the challenges listed above. This will be done, using a prototype as proof of concept. Based on the aforementioned, this report will stand on four legs:

1. Initial analyses and design thoughts, leading to the use of agricultural robots.
2. Reverse engineering on a pre-build prototype, setting up hardware communication and security.
3. Collecting and filtering sensor signals, to determine the correct position and orientation of the robot.
4. Trajectory and controller development, which will make the robot act in a controlled manner.

2 Introducing the concept

This chapter will start by analyzing some challenges to modern farming operations regarding field work. This will lead to some initial thought on how some of these challenges can be meet. The chapter also contains a brief introduction in to what research is performed in the field. Finally the robot used in this project will be introduced.

2.1 Modern farming operations

Farming operations today, are under a high pressure to raise efficiency and lowering costs. This means that a constant search for higher productivity takes place. A large part of the workload on a farming operation is related to agricultural work. Today this is carried out using versatile tractors, controlled by human labour, pulling different kinds of attached machinery. To cut labour costs, there is an ongoing search for higher productivity during field operations.

There are two main approaches to improve efficiency regarding field work. The first one is to use wider machinery, covering a larger area in one go. The second one is to drive with a higher velocity. There is although a natural non-linear behaviour to these two approaches. Using wider machines requires larger and heavier framework that can translate the forces from the tractor. This means that the wider this machinery is, the larger framework is needed, resulting in very heavy machinery that requires large and heavy tractors to manage and control them.

Driving with a higher velocity also have some implications. The relationship between velocity and force, needed to pull a tillage tool in the ground is not linear [16]. This means that increasing the speed will require an excessive force, again resulting in very large tractors. The result of these factors have today lead to the use of very heavy tractors, sometimes exceeding weighs of 20 tons (see fig. 2.1). The use of heavy machinery and the risk of applying structural damage to the soil have been studied quite intense in the last decades. The interested reader can consult [22], [14], [2] for further knowledge. It is well known that high soil compaction

(a) John Deere 9620 RX
weigth approx 24,950 kg.[8]

(b) CaseIH Magnum 380
weigth approx. 12,850 kg.[17]

(c) Fendt 1050 Vario
weight approx. 14,000 kg.[12]

Figure 2.1: Examples of modern agricultural tractors, used in farming operations today, all operated by manual labour.

leads to lower yields, meaning that soil compaction can result in lower economical success for the farmer.

Regarding soil compaction, there is two main concerns. One is the pressure on the surface of the soil. This is very much related to the size and air pressure of the tires. The other one is compaction in the deeper layers, which is mainly a result of total weight of the machinery. The effect of these factors have e.g. been studied in [19]. Further investigation into these topics are considered out of scope of this report.

To sum up this section, we can conclude that in modern farming operations there is an urge to reduce labour hours, which have resulted in big heavy machinery, which again can result in structural damage to the soil resulting in lower yields.

2.2 Field robots

Using field robots would not require the same amount of human labour, which means that instead of using big heavy equipment, small robots that could run around the clock, could be a way of overcoming the challenges mentioned above. This leads to the the main theme of this project. The idea is to develop a field robot that can do some of the work related to cultivation of the soil, preparing the seedbed and sowing. Furthermore it could be used during the growing season, for nursing the plants by row cultivating, spraying, or other ways of fighting weeds and pests.

As a side note it should be mentioned that this robot project is not aimed at any of the grain harvesting. Since the harvest is performed during a small time horizon and under stressful weather conditions, this work is today done with very big harvest combines, that can cover large areas in a short time frame. Furthermore

(a) The little field robot "Oz", developed by the French company Naio.[20]

(b) The field robot "Robotti", developed by the Danish company Agrointelli.[1]

Figure 2.2: Examples of modern agricultural robots. All though used in farming operations today, they must still be considered as being in early development.

harvest is a quite complex process which would require dedicated and thoroughly investigation and development before any robotic solutions could prove useful. Regarding harvest of specialised row crops, such as fruits and vegetables, a robotic solution seems more feasible.

Based on the aforementioned, the main idea is to develop a versatile universal field robot, where different tools can be attached with respect to what kind of work is present at any given time, this could also be referred to as a robotic tractor.

2.3 Other agricultural robotic concepts

Even a very quick research on the web for agricultural robots will reveal that this topic enjoys a quite large attention around the globe. Many robotic university departments, are working with different kinds of set-ups. The interested reader can, as examples, take a look at [18], [23], [10], [5]. However these intense efforts have not lead to a breakthrough in the commercializing of robotic concepts to use for farmers today. In Denmark at least 3 companys have announced robots that should be ready for sale in 2019. The first one is the "Oz" robot from the French company Naio. This little robot is meant to be used for cultivating in row crops to fight weeds (see fig. 2.2a). The other robot is a bigger and more versatile robot called "Robotti". This robot can also be used for cultivation but also many other jobs depending on the application attached to it (see fig. 2.2b). The third example is the FarmDroid [11], which is capable of sowing and removing weeds. Other interesting solutions, that are on the edge of commercializing, could be the Thorvald robot from the Norwegian SagaRobotics or Bonirob from Bosch.

2.4 Design thoughts

Since one of the main arguments of building field robots were the risk of compaction of the soil, it is obvious that the robot should not be too heavy. On the other hand, it should still be large and heavy enough to be able to operate in the uneven, harsh environment of a field, while producing some force to pull the attached tool. Small remote controlled "tractors" have already been developed. These are e.g. used for mowing in steeply inclined terrain. A representation of such a machine, build by Bomford-Turner®, can be seen in fig. 2.3. A machine like this could also be build as an autonomous tractor with a central processor to run the control loop instead of the remote control. The design thoughts for a field robot is based on concepts like this.

Figure 2.3: A conceptual representation of a field robot, equipped with tracks. This specific machine is not designed for autonomous use, but is remotely controlled by the user. Weight 1,200 kg. [6]

The shown example has a weight of approx. 1,200 kg. It has a length of 2,09 m. and a width of 1,39 m. The machine can handle attached machinery with a width of 1,5 m. A big agricultural tractor can, depending of the application, easily handle tools that are 4 times wider. This means that the robot has a capacity 4 times smaller than the tractor if the same velocity is used. On the other hand, the robot can

run 24/7 which in case of the tractor would require at least 2 employees working in shifts. The conclusion is that, under normal working conditions, a big modern tractor could be replaced by 2-3 robots, but it must be imposed that these considerations are very theoretical and can vary a lot under different circumstances.

Also transmission of power to the ground is important. How much power needed is dependent on the width of the tool, the depth of the cultivation, the velocity, terrain, etc. The machine shown in the figure can produce 30 kW, but also this topic is very much dependent on application. If e.g a deep cultivation is desired, the power needed is much higher than e.g light cultivation or spraying

To lower compaction as much as possible and to increase traction, the robot should be equipped with tracks. The introduction of tracks would also make the modelling and control of the robot more simple, since booth steering and velocity can be controlled by adjusting the angular velocity of each track. Regarding the power line, such a robot could be powered by two electrical motors, one for each track. The electricity could then come from either a diesel generator or a battery pack. Also more unconventional power sources, such as solar panels or fuel cells could be used in later iterations of the development. The robot would of course have to be equipped with a sensor suite to determine position (GPS/RTK), orientation and obstacles in the nearby environment.

All these thoughts are not done to come up with an optimal size and design for an farm robot, but just meant to show that the design of such a robot is a function of application, weight, velocity, depth of cultivation, terrain etc. Further investigation in to these design thougths is out of scope of this project and the focus will be aimed at developing and implementing an autonomous control algorithm, for the robot used as proof of concept.

2.5 Proof of concept

For use in this project, Aalborg University have a field robot available (see figure 2.4). This robot is based on the RobuROC4 from the French company Robosoft. The robot is build with four wheels in a rigid frame constituting it as a skid-steered vehicle. This steering concept is very close to that of a tracked vehicle. Although this robot do not have the completely desired dimensions, it is still of a size where it makes sense to use it as a farm robot for (very) light applications. The robot is not heavy and powerful enough to pull heavy attached machinery, but since this project will focus on prototype development, this is not important. Furthermore it is equipped with the aforementioned needed sensor suite. Based on these con-

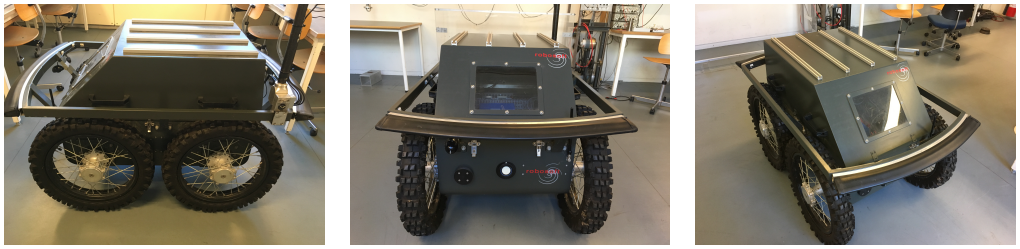


Figure 2.4: The robot, seen from different angles

siderations it was decided to use this robot as proof of concept and as base of the mathematical modelling, controller development and implementation that is needed before such a robot is able to perform work in the real world. Throughout the rest of this report, this robot will be used for prototype development and referred to as "the robot".

2.6 Mapping and trajectorys

Using autonomous field robots would require that the robot can use a map where the actual position of the robot can be related to a position on the map. A specific field must then be represented by some array of GPS-coordinates that the robot can use. One way of building such a map could simply be done, by manually controlling the robot and letting it run one complete trip around the borders of the field, while collecting position data. In that way the field will then be defined by all the GPS-coordinates inside this border. Another way of doing it, is to read in some predetermined map in to the robot on beforehand. The Danish authority "Styrelsen for Dataforsyning og Effektivisering", have made it possible to define and download maps online. These maps contains GPS-coordinates that can be used for robotic applications. An example of such a field map can be seen in fig. 2.5. Many farms today already have made GPS-maps used for Autonomous steering and farm management. It should also be possible to use these maps for robotic farming.

Just defining a map is of course not enough. The robot must be given some predetermined trajectory that it should follow on a given field. When starting to work in a field the trajectory must consist of parallel lines with a space between, corresponding to the width of the attached machinery. These trajectory lines must include a starting point and target point, where the robot lower/raise the tool to start/stop the treatment of the soil. The trajectory of the headland must also be taken into consideration. The problem of following a trajectory will be handled in a later chapter.

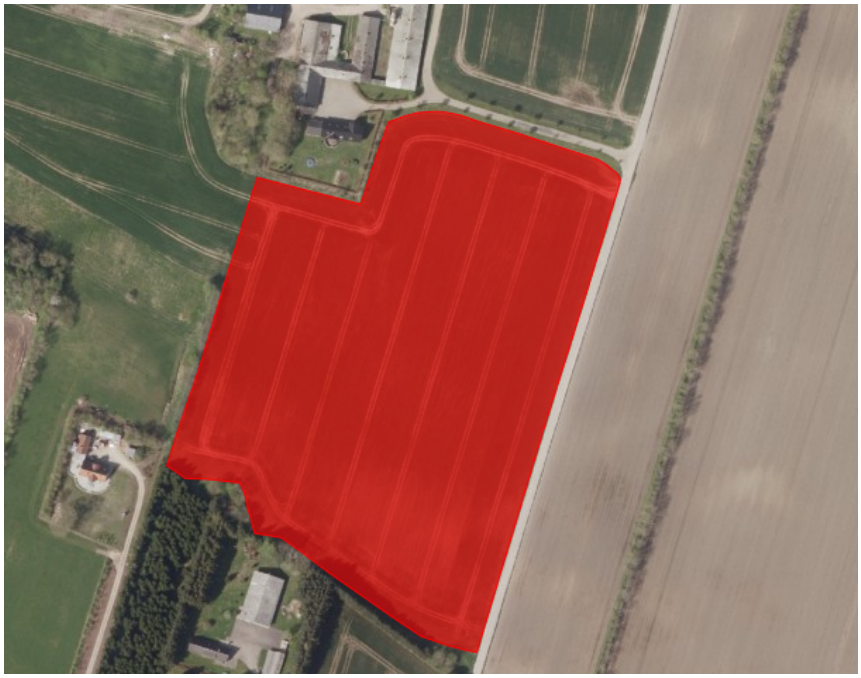


Figure 2.5: A representation of a field map, produced and downloaded from sdfe.dk

2.7 Problem formulation

Based on the aformentioned topics it is now possible to express a problem formulation for this specific project:

Is it possible to model and control a field robot prototype, based on the RobuROC4, that can run autonomously and follow a given trajectory.?

This chapter has now described some of the reasons for igniting this project. Furthermore the concept has been introduced along with some initial design thoughts of a field robot. The chapter has also briefly gone through mapping and trajectory generation. Finally the problem formulation has been ignited. The robot that will be used for proof of concept has been discussed. A more deep technical description of the robot is of course needed for further development, which will be handled in the next chapter.

3 The robot

The robot is of the type RobuROC4 built by French company Robosoft. As mentioned in section 2.5, it has a rigid frame with four wheels that constitutes it as a skid-steered vehicle. Each wheel is driven by an electrical 3-phase A/C servo motor, controlled by a digital servo drive. The communication with the four motor drives is handled by a CAN-bus connection, that also is connected to a central computer. The robot is powered by an 48 volt LiFePO4 battery with a capacity of 40 Ah. Besides these main parts, the robot holds quite an amount of relays and other electric circuits, used for safety circuits, brakes etc. A look inside the robot can be seen in figure 3.1

3.1 Control setup

Before start-up of this project, the robot was controlled by a build-in Windows CE embedded micro-controller, running Windows CE®. This computer would handle the overall control and make sure that commands, given by the user, would be transformed to commands for the motor drives. The user would then be responsible for developing algorithms that could act as a higher level of control and be put on top of the programming, already done in the Windows CE computer. This higher level program should then deliver commands given as velocity and angular velocity.

There were two main problems with this approach: The first thing was that the Windows computer had to be treated as a black box, in the sense that the programs running on it, was not known. This sometimes made it very hard to foresee how the robot would react. The other problem with this approach was that, seen from a control perspective, it would be better to control each motor directly, by sending current or velocity commands to each motor drive, instead of just giving overall commands to the robot.

Because of these aforementioned problems, it was decided to disconnect the Win-

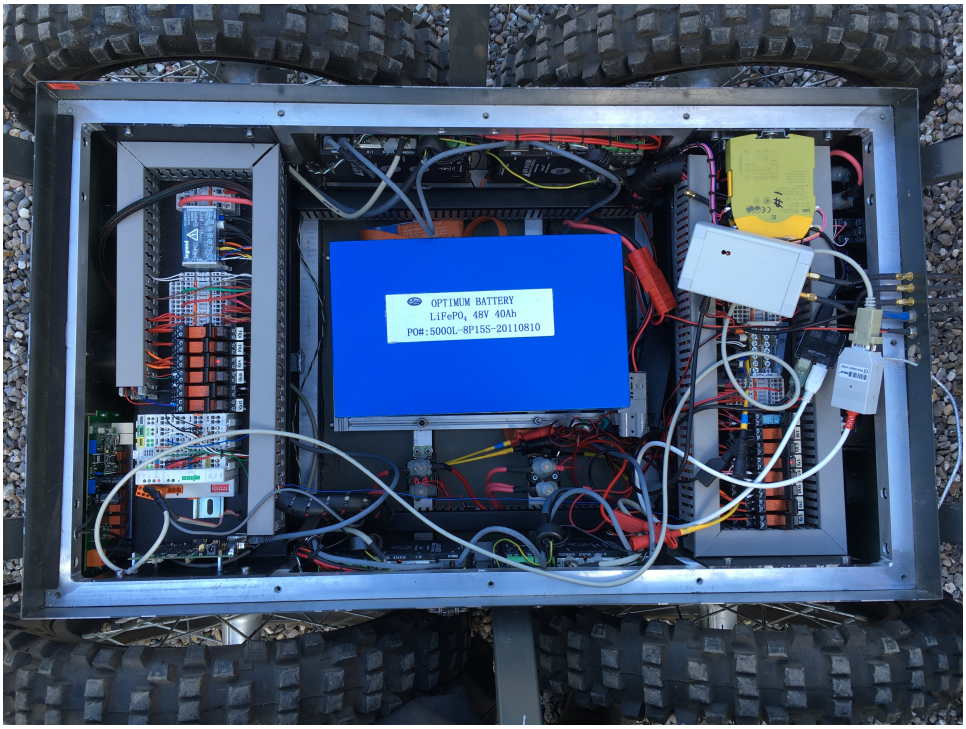


Figure 3.1: The inside of the robot holds a lot of electronic circuits. In the middle, the battery is seen. The small orange boxes are the relays. The yellow box in the top right corner is the security system that has been decoupled. The white box right below, holds the GPS-chip and the radio communication for the RTK signal. Below the battery, the main relays are mounted.

dows computer and connect a pc directly to each drive via the CAN-bus. A part of this project is therefore dedicated to perform some reverse engineering on the robot and set it up, so that a laptop can be used as a central control unit, where the user is free to build algorithms that controls the drives directly. A graphical representation of this approach can be seen in figure 3.2.

3.2 Motor drives

The motor drives, seen in figure 3.3 are of the type DPCANTR-040B080 from Avanced Motion Controls® (AMC). They are designed to power 3 phase AC induction motors, with a constant current up to 20 Amp. and a peak current up to 40 Amp. The motor drives are programmable and can be configured to operate in torque, velocity or position mode. Furthermore they can be configured for a variety of external command signals.

The drives can be configured using the software DriveWare®, which can be downloaded from the company web page. This software can be used for a lot of op-

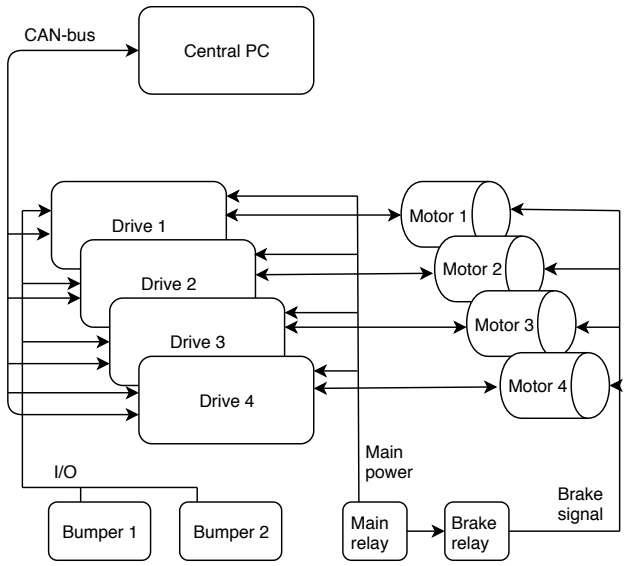


Figure 3.2: A chart of the control structure at the robot. A central laptop is directly connected to each drive via CAN-bus. Each drive handles the correct voltage and current to a motor, which sends feedback signals back to the drive. The two bumpers on the robot is connected directly to the drives via the I/O connection. From the main relay, the power goes to the drives. An additional connection was made to the brake relay, which unlocks the brakes.

erations e.g. configuring the drives, setting motor parameters, define inputs and outputs (analog and digital), setup events, setup communication and much more. When using DriweWare, it is necessary to connect to the auxiliary port on the drive, using a USB-RS232 cable with a Phoenix 1881338 plug mounted. A user manual for using DriveWare can also be found on the web page.

During operation of the drive, the command signals must be sent via the CAN-bus. An introduction to this type of communication and how to communicate with the drives, will be done in chapter 4.

3.3 Motors

The robot is equipped with four electric motors. They are produced by Infranor-Mavilnor® and are of the type BLS-072. The motors have a stall torque of 1.8 Nm and a peak torque of 7.2 Nm. The torque/current ratio of the motors is 0.16 Nm/A. To increase the output torque, the motors are equipped with a gearbox on which the wheels are mounted. The gear-ratio between the motor and the wheels is 1:32. As mentioned earlier, the motors are 3-phase AC servomotors. The motors are mounted with a resolver to produce a feedback signal. Furthermore the motors have brakes that can be activated from the drive. For this project the brakes were

Figure 3.3: One of the four AMC DPCANTR-040B080 motor drives, situated in the robot, one for each motor. Seen from the top left to right is the CAN-bus connection, followed by the input/output port. Next comes the feedback port, followed by the power connection, both connected to the motor. Finally the power input to the drive is seen, which is split in high power to the bridge and logic power. On the left side the auxiliary port for communication with the drive is seen.

not used in the control loop, however it is neccesary to supply a voltage to the brake circuit to unlock the brakes. This is done by connecting a wire to the main relay, so that the brakes are unlocked, when the robot is turned on.

3.4 Bumpers

Besides the motors and drives the robot has a security system that, via a remote control, can turn off the drives and stop the robot. It was decided not to focus on the security aspect in this project, so this system has been decoupled. However the robot has bumpers mounted on the front and back, that are capable of giving a signal if the robot collides with something in the surroundings. To maintain some level of security these bumpers are used to trigger an emergency stop if the bumpers are activated. This is done on the hardware level, where the bumpers are connected directly to an input pin on the drives. The drives are then set-up to disable the power-bridge, if a signal comes in from the bumpers. This approach ensures that the bumpers always works disregarding any unintentional bug in the software.

Figure 3.4: One of the four A/C servo motors situated in the robot. A gearbox is mounted on the motor and each wheel is then mounted on the end of the gear shaft. There is two cables connected to the motor, one for power and one for the feedback signal to the drive.

This chapter has now described some of the technical hardware components on the robot. The communication on the robot is, as previouly, mentioned handled via CAN-bus. This topic is explained in chapter 4. Furthermore the sensors, mounted on the robot, will be analysed in chapter 6.

4 CAN-bus

Since CAN-bus is an essential part of the communication on the robot, a short description of this network type will be given in this chapter. Furthermore the most important parts of the communication with the motor drives, will be discussed.

4.1 CAN in general

Controller Area Network (CAN) was originally designed by Bosch in 1983 with the first dedicated CAN controller chips produced in 1987 [4]. CAN is a bus standard and was originally designed for the automotive industry, but has later expanded to be used more generally in the industry as well. CAN is used as a communication network between numerous Electronic Control Units (ECU's) e.g micro-controllers, sensors, actuators etc. In 1991 Bosch published the CAN 2.0 specification, Which has an A and B-part. A is a standard format with a 11-bit identifier and B is an extended version that uses 29 bit for identification. In 2012 the CAN FD 1.0 was released which uses flexible data rates.

The International Organization for Standardization (ISO) has released standards for CAN messages. ISO 11898-1 covers the datalink layer while 11898-2 and 11898-3 covers the CAN physical layer for high and low-speed CAN respectively. These standards provides the basis for communication, but does not specify how the raw data can be decoded. A set of standardized protocols exists to further define how data is communicated. Among these protocols are e.g. SAE J1939, used for heavy vehicles (busses, trucks etc.) and CAN-open which is widely used in industrial automation applications and embedded systems. The CiA 301 specification defines the basic device and communication profiles for CAN-open. CAN-open is also the protocol used for communication on the robot in this project.

To answer why CAN has become popular, several reasons can be mentioned:

- **Low cost** - The wiring is very simple and no further analog wiring is required.

- **Efficient** - The messages are prioritized so that higher priority messages are not interrupted.
- **Centralized** - Centralized configuration, control and diagnoses are possible.
- **Flexible** - It is easy to add new nodes to the network.
- **Robust** - A CAN network is robust to failure of subsystems and interference/noise.

When connecting to a CAN network a dedicated chip is needed. This chip will handle the transmission on the bus and take care of prioritizing the messages. The data rate will vary with the length of the wire/network. With a length of 40 meters the data transmission can be as high as 1 Mb/sec., while a length of 1000 meters will reduce the data rate to 50 kb/sec. The physical properties of the bus is simply consisting of three wires: Ground, high and low, where signals is determined by the difference in voltage between the high and low wires.

4.2 CAN message

A graphical representation of a CAN message can be seen in figure 4.1. Such a message contains several attributes, listed below:

- **SOF** - Start Of Frame, is a dominant 0, which tells other ECU's (nodes) on the network that a message is coming.
- **Can ID** - Is a mixture of ID's for the different nodes and the priority of the message. The lower number, the higher priority.
- **RTR** - Remote Transmission Request. Allows one node to request messages from other nodes. This can also be between master and slave.
- **Control** - Give information of the length of the data field in bytes.
- **Data** - Contains the actual data in the message. This field can contain from 0-8 bytes.
- **CRC** - Cyclic Redundancy Check, is done to check the integrity of the data.
- **ACK** - is an acknowledgement of the CRC
- **EOF** - Marks the End Of Frame.

When using a CAN network, the only interesting parts is the CAN ID, RTR, Control and Data fields, while the rest is automatically handled by the dedicated CAN-chipset.

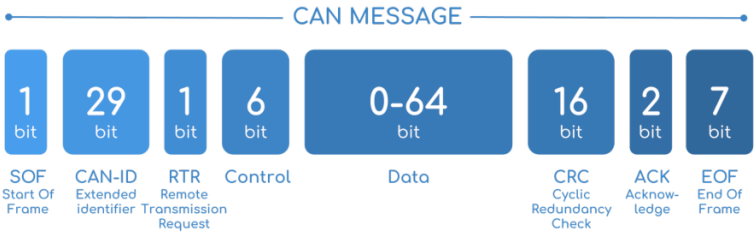


Figure 4.1: A representation of a CAN message. Note that this particular message is the 2.0-B extended version with the 29 bit identifier.

4.3 Connecting to the robot

To access the CAN network on the robot it was necessary to disconnect the on-board Windows computer and solder a new connector to the CAN wires. A CAN-USB IPEH-002021 adapter from PEAK-systems is then used to connect a laptop to the CAN-network on the robot. This makes it possible to use this laptop as a "centralized control unit", where from the robot can be controlled. This adapter comes with driver and a pc-program that can be used to analyse data on a CAN-network.

As mentioned earlier the robot contains four AMC motor drives that can be accessed via the CAN network using the CAN 2.0-A specification with an 11-bit identifier. The protocol used is CAN-open, following the CiA 301 communication profile and the CiA 402 device profile. The communication now works by sending CAN messages to the drives and thereby giving command signals, that the drives will convert to a current on the motors. An extensive communication manual for the drives are supplied at the company website (www.a-m-c.com). Each drive function is defined by a group of objects, which holds a specific parameter. These parameters are used to perform all the functions that the drive can do, e.g. current control, velocity control, input/output functions etc. There is an unique object for each parameter. Depending on the type, each object can be either writeable, readable or both. Each object is accessible via the CAN-communication with a 16 bit address called the object index. Some objects also have 8 bit sub-indexes.

Information between the nodes on the network are exchanged via CAN-messages. These messages can be generated by request in an interupt driven matter, or they can be set up to periodic transfer. Every message starts with an arbitration field which consists of the CAN ID and the RTR bit. The values in the arbitration field set the priority of the message. The lower number the higher priority. The arbitration between the different messages are done at the hardware level. The data format is "Little Endian" meaning that the data must be sent with the least significant byte first.

Message type	Description	CAN-ID
NMT	Network management(broadcast)	0h
NMT error control	Network management error control	701h-77Fh
Boot-up	Boot-up message	701h-77Fh
Sync	Syncronization message(broadcast)	80h
Emergency	Emergency messages	81h-FFh
Time stamp	Time stamp(broadcast)	100h
PDO	Process Data Objects	181h-57Fh
SDO	Service Data Objects	581h-67Fh

Table 4.1: The 8 different CAN message types.

The CANopen standard divides the 11-bit CAN frame id into a 4-bit function code and 7-bit CANopen node ID. This means that the CAN-id begins with a number, which relates to the type of message, and ends with the node id. In the CANopen system there are 8 different message types, as seen in figure 4.1. We will not discuss all these different types, but only shortly mention the ones of interest to this project, namely Network Management (NMT), Process Data Objects (PDO) and Service Data Objects (SDO).

- NMT - This type of message has the highest prority. It controls the communication with the state-machine on the drives. These messages can set the drive in five different states: Start and stop of the drive, pre-operational state, reset drive and reset communication with the drive. It is important to handle these messages correct during startup, to make the communication work. An example can be seen in table 4.2. The NMT messages is also used for lifeguarding during runtime where the master polls each node regularly to recieve information of the drive state.

- PDO - A PDO message is a single unconfirmed message that must be configured prior to startup. PDO's are restricted to contain only 8 bytes of data and are generally used during runtime to give target commands to the drive and send e.g. velocity or position back to the host. An example can be seen in table 4.3.

- SDO - An SDO message consists of an outgoing message from the host to the node and a reply from the node to the host. There is no restrictions on the size of a SDO message and they are typically used for configuration during start up. An example can be seen in table 4.4.

Arbitration Field		Ctrl	Data Field	
CAN-id	RTR		Byte 1	Byte 2
000h	0	2	81h	02h

Table 4.2: An example of a NMT message during startup. This specific message will reset drive 2 (code 81h).

Arbitration Field		Ctrl	Data Field			
CAN-id	RTR		Byte 1	Byte 2	Byte 3	Byte 4
403h	0	4	0Fh	00h	FFh	7Fh

Table 4.3: An example of a PDO message. This specific message will give a velocity target command to drive 3.

Arbitration Field		Ctrl	Data Field							
CAN-id	RTR		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
603h	0	8	40h	64h	60h	00	00	00	00	00

Table 4.4: An example of a SDO message. This specific message is sent from host and will ask drive 3 for the data contained in object 6064h (code 40)

During startup it is a good idea to reset the drives. This ensures that the counters in the drives are reset and the drive is in the proper state. After reset, the communication state machine must be set to operational, so that communication can take place. These two steps are handled by NMT messages. A periodic NMT mesage must now be set up to handle the lifeguarding of the drives. This message must have the RTR bit set to high and do not carry any data. After these initial steps it is now possible to send commands to the drives.

This chapter has now discussed the communication with the robot through the CAN-bus protocol. Only the most important commands have been mentioned, but it must be emphasised, that the drives have a very large communication protocol, which makes it possible to get a lot of information from the drives during runtime. A thoroughly examination can be found in the communication manual for the drives. The report will now move on to describe the robot from a mathematical perspective.

5 Mathematical model

This chapter will describe the mathematical modelling of the robot. The intention with this chapter, is to create a simple model, that can be used to control the robot. The modelling will therefore be done by setting up simple equations of motion and then determine the parameters by testing the robot in a real world situation.

5.1 Kinematics

When building a model of the robot, it can make sense to think of the motion, expressed as a linear velocity, denoted v and an angular velocity denoted ω. When the model is implemented in a computerized algorithm, the motion must be translated in to a velocity on each wheel that can be send as commands to the drives. In this section the translation between these factors will be done.

As mentioned earlier the robot is skid steered, which means that the two wheels on each side can be considered as pairs. If we choose this solution, the left front wheel will therefore have the same velocity as the left rear wheel. The same goes for the wheels on the right side. We will denote the angular velocity of the left and right wheel-pairs (expressed in rad/s) as v_l and v_r respectively. With r denoting the radius of the wheels, the overall velocity v of the robot can now be described as:

$$v = \frac{r(v_l + v_r)}{2} \tag{5.1}$$

When describing the model of the robot, we assume that the robot is working in a flat field. This means that the position of the robot can be described in a Cartesian global plane $S \in \mathbb{R}^2$, as x and y-coordinates, and the orientation, denoted by θ is expressed as the angle between the x-axis and a line following the central axis of the robot (see fig. 5.1). With the angular velocity of the robot expressed as ω the

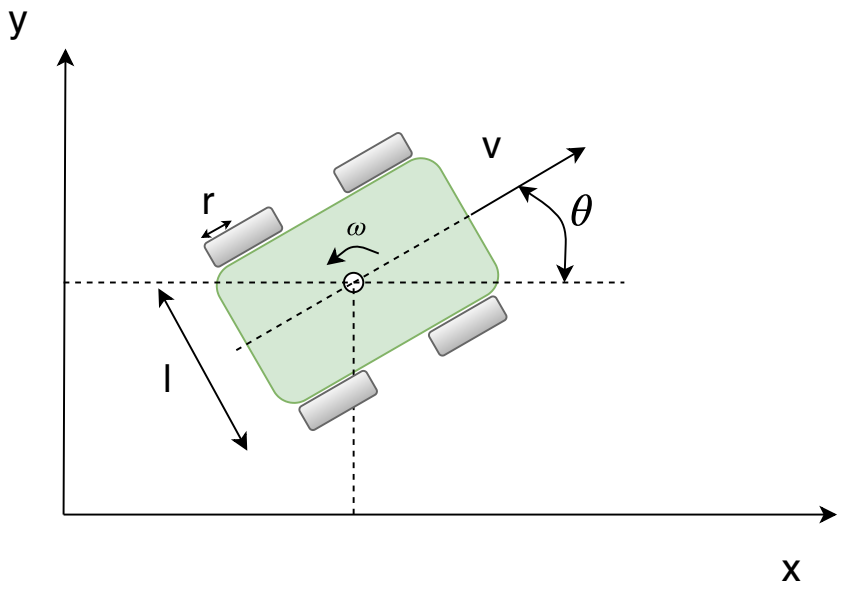


Figure 5.1: Kinematic overview. The robot is situated in a global frame with an x and y-position. v denotes the velocity in the longitudinal direction , while θ denotes the robot heading with respect to the x-axis. The wheel-track is denoted by l, while radius of the wheels is denoted by r.

movements can be descibed as:

$$\dot{x} = v\cos(\theta) \tag{5.2}$$
$$\dot{y} = v\sin(\theta) \tag{5.3}$$
$$\dot{\theta} = \omega \tag{5.4}$$

Taking the wheelbase, denoted as l in to consideration, we can also describe this as:

$$\dot{x} = \frac{r}{2}(v_l + v_r)\cos(\theta) \tag{5.5}$$
$$\dot{y} = \frac{r}{2}(v_l + v_r)\sin(\theta) \tag{5.6}$$
$$\dot{\theta} = \frac{r}{l}(v_l - v_r) \tag{5.7}$$

Merging these equations, we come to:

$$v_l = \frac{2v - \omega l}{2r} \tag{5.8}$$
$$v_r = \frac{2v + \omega l}{2r} \tag{5.9}$$

Which means that we can now relate v and ω to the velocity on each wheel-pair v_l and v_r.

5.2 Dynamics

Before considering dynamics of the robot, it must by emphasised that any attached machinery is not taken in to consideration. If this was the case, terms for inertia of the machinery and resistance from pulling this machinery through the ground would have to be added. Since no machinery is attached to the robot prototype and to simplify the model, these parts will be left out of the equations.

An overview of the forces working on the robot can be seen in figure 5.2. Some of the considerations in this chapter are based on actual tests with the robot. These tests are described in chapter 11. First we will consider how the forces can be modelled, when the robot moves in the longitudinal direction.

According to Newton's law, the robot must overcome a force from acceleration related to the inertia of the mass m. Furthermore a test, described in section 11.5, showed that forces related to velocity, also acts on the robot. These forces are split in an element that act proportional to the velocity, which we will denote k_p, and an element expressing some internal resistance in gears and motors that must be overcome before the robot can move, which we will denote k_i. The force F_v that the robot must produce, when moving in a linear direction, can therefore be calculated as:

$$F_v = m\dot{v} + k_p v + sgn(|v|)k_i \tag{5.10}$$

This force must now be translated to a summed torque τ on the axles of the robot. The formula is given as:

$$\tau = \sum_{i=1}^{4} \frac{F_v r}{4} \tag{5.11}$$

where i denote a number from 1-4 describing the four wheels. As explained in section 3.3, the motors are mounted with gears to increase the axle torque. Denoting the gear ratio g_r and the torque/current ratio of the motors k_τ, the necessary current, denoted I, needed to drive the robot, can be calculated as:

$$I = \frac{\tau g_r}{k_\tau} \tag{5.12}$$

Inserting eq. 5.10 and 5.11 in to 5.12, we can now calculate how much current is needed with respect to v and $\dot{v}$ by:

$$I = \dot{v}\frac{m r g_r}{k_\tau} + v\frac{k_p r g_r}{k_\tau} + sgn(|v|)\frac{k_i r g_r}{k_\tau} \tag{5.13}$$

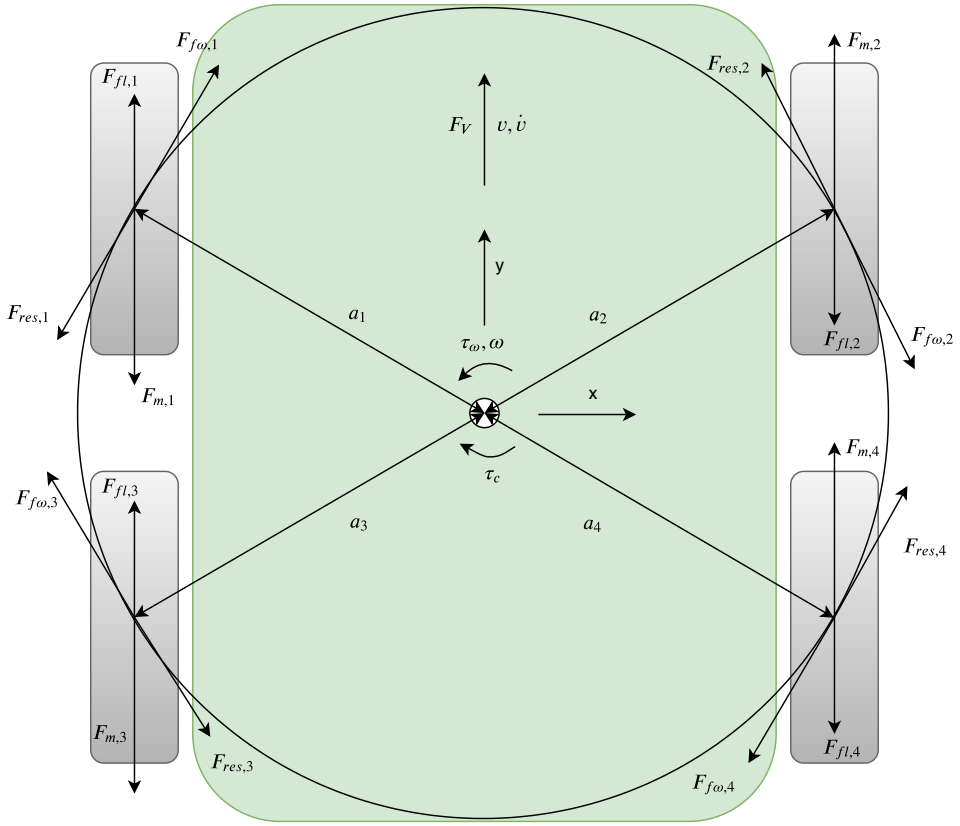


Figure 5.2: An overview of the forces working on the robot. When an angular torgue (τ_ω) is put on the robot, this will produce resulting force vectors (F_{res}) on each wheel. This will again result in a friction force ($F_{f\omega}$) from the sliding of the wheels, when the robot turns. F_m describes the force produces by the motors and F_{fl} describes friction in the linear direction of the wheel.

To analyse how the forces, created by the motors, affects an angular torque (τ_ω), needed to turn the robot, it is necessary to also consider the friction between the wheels and the ground. This can be done by defining vectors describing the forces.

Let a_i define vectors from the centre of the vehicle to the point where the wheels touch the ground. Consider resulting force vectors ($F_{res,i}$) on each wheel.

The relationship between τ_ω and $F_{res,i}$ can be determined by calculating the sum of the cross product between a_i and $F_{res,i}$. This will produce a perpendicular vector, that will represent the torque τ_ω acting on the robot. The calculations can be done

by defining a_i as matrices (denoted by $[a_i]_x$) and then multiply with $F_{res,i}$.

$$\tau_{\omega,i} = [a_i]_x F_{res,i} \tag{5.14}$$

$$= \begin{bmatrix} 0 & -a_{i,z} & a_{i,y} \\ a_{i,z} & 0 & -a_{i,x} \\ -a_{i,y} & a_{i,x} & 0 \end{bmatrix} \begin{bmatrix} F_{res,i,x} \\ F_{res,i,y} \\ F_{res,i,z} \end{bmatrix} \tag{5.15}$$

By incorporating the $[a_i]_x$ matrices in to one matrix, we can define a transformation matrix $T \in \mathbb{R}^{3\times 12}$ by

$$T = \begin{bmatrix} [a_1]_x & [a_2]_x & [a_3]_x & [a_4]_x \end{bmatrix} \tag{5.16}$$

In the same way we can incorporate the $F_{res,i}$ vectors in to a single vector $f_F \in \mathbb{R}^{12\times 1}$.

$$f_F = \begin{bmatrix} F_{res,1} \\ F_{res,2} \\ F_{res,3} \\ F_{res,4} \end{bmatrix} \tag{5.17}$$

Now τ_ω can be calculated by

$$\tau_\omega = T f_F \tag{5.18}$$

To calculate the resulting forces acting on the wheels, based on τ_ω, it is necessary to calculate in the opposite direction and use τ_ω to calculate f. With † denoting the pseudo inverse, this is calculated by

$$f_F = T^\dagger \tau_\omega \tag{5.19}$$

Not all the resulting forces ends up in actual motion of the robot. Some of the force is used to overcome friction. This friction will add up and result in a counter torque (τ_c), acting on the robot in the opposite direction of τ_ω. For ease of notation, the calculations will be done in general for all four wheels leaving out the numeric notation (i). Let $F_{f\omega}$ denote the force used to slide the wheels in the direction perpendicular to the line a. This can be modelled as dry (kinetic) friction since it describes the resistance of sliding between two solid surfaces. With F_n denoting a normal force from the wheels on to the ground and μ_d denote the coefficient of the dry friction, the formula is given by:

$$F_{f\omega} = F_n \mu_d \tag{5.20}$$

Let F_{fl} describe the resistance force produced in the longitudinal direction of the wheels, opposite to the force produced by the motors. As in equation 5.10, this

resistance is composed of two parts, k_p and k_i. With v_w denoting the ground velocity of the wheel, this vector is calculated as:

$$F_{fl,i} = k_p v_w + sgn(|v|)k_i \tag{5.21}$$

$$\tag{5.22}$$

Also considering moment of inertia γ related to the angular acceleration, The counter torque τ_c is now

$$\tau_c = \dot{\omega}\gamma + [a]_x(F_{fl,i} + F_{f\omega,i}) \tag{5.23}$$

The determination of γ, F_n, μ_d, k_p and k_i is done in the next section.

5.3 Determining parameters

To use the model, the coefficients must be determined. The wheel-track and the wheel-radius can just be directly measured. The robot mass is given in the specifications of the robot. The gear ratio and torque/current ratio can be read directly on the gears and motors in the robot.

The moment of inertia can be approximated by defining the robot as a box with an even distribution of mass. With R_w and R_l denoting the robot width and length respectively, the formula is given as:

$$\gamma = \frac{1}{12}m(R_w^2 + R_l^2) \tag{5.24}$$

$$= \frac{1}{12}140(0.68^2 + 0.93^2) = 15.5[kgm^2] \tag{5.25}$$

The normal force F_n on the four wheels, is calculated as:

$$F_n = mg \tag{5.26}$$

$$= 140 \cdot 9.82 \tag{5.27}$$

$$= 1375[N] \tag{5.28}$$

k_p and k_i is determined by running the robot at different velocity's and then measure the actual current used. This is described in section 11.2, where a function was fitted to the test data and estimated as:

$$I = 1.26v + 1.8 \tag{5.29}$$

Comparing this function to equation 5.13, without considering acceleration:

$$I = v\frac{k_p r g_r}{k_\tau} + sgn(|v|)\frac{k_i r g_r}{k_\tau} \tag{5.30}$$

and knowing that the relationship between the current used, by the motors (I_m) and the produced force (F_m) is calculated by:

$$F_m = I_m \frac{k_\tau}{g_r r} \tag{5.31}$$

k_i can now be estimated to

$$1.8 = sgn(|v|)\frac{k_i r g_r}{k_\tau} \tag{5.32}$$

$$= k_i \frac{0.29 \cdot \frac{1}{32}}{0.16} \tag{5.33}$$

$$k_i = 32.8 \tag{5.34}$$

k_p can be estimated to

$$1.26v = v\frac{k_p r g_r}{k_\tau} \tag{5.35}$$

$$1.26 = k_p \frac{0.29 \cdot \frac{1}{32}}{0.16} \tag{5.36}$$

$$k_p = 22.2 \tag{5.37}$$

The friction coefficient μ_d will of course heavily rely on the terrain, moisture, tire type etc. In future developments a changing coefficient could therefore perhaps seem usefull. For this project, μ_d is set to 0.36, which according to [9] is the friction coefficient between rubber and grass.

Parameter	Description	Value
l	Wheel-track	0.69 $[m]$
r	Radius of wheels	0.29 $[m]$
m	Robot mass	140 $[kg]$
g_r	Gear ratio	1/32
τ_I	Torque/current ratio motors	0.16 $[Nm/A]$
γ	Moment of inertia (robot)	15.5 $[kgm^2]$
F_n	Normal Force	344 $[N]$
k_i	Internal resistance coefficient	28.3 [N]
k_p	Proportional resistance coefficient	22.2 [N/m/s]
μ_d	Friction coefficient	0.36 [.]

Table 5.1: The parameters used in the model.

This chapter has now described how the robot can be modelled from a kinematic and dynamic point of view. Especially the dynamic modelling of the friction during turns, are done with large uncertainties. The forces acting on a wheel sliding

over different terrains are highly non-linear and could be modelled much deeper. Since the purpose of this model chapter, was to build a simple model, that can be used for control, a deeper investigation in to these aspects are considered out of scope. The next chapter will describe the sensor suite, used on the robot.

6 Sensors

When dealing with mobile robots, one of the core problems is to determine the exact position and orientation of the robot at any given time. To gain this knowledge, the robot is, as mentioned earlier, equipped with a suite of sensors. This chapter will describe the different sensors used in this project.

6.1 GPS

A GPS-sensor is used to determine the position of the robot. This sensor is of the type Ashtech MB100 from Trimble and is combined with a base station to receive a Real Time Kinematic (RTK) signal for further precision. This makes is able to deliver exact positioning data with an accuracy down to 2 cm. The system can also deliver an estimate on the heading of the robot. The specific sensor on the robot is from an older iteration of development and can only recive signals from the GPS system. Newer versions is able to receive signals from the international Global Navigation Satellite Systems (GNSS), which, besides GPS (USA), consists of GLONASS (Russia), Beidou (China), and Galileo (Europe).

The GPS-system is connected to the central pc by a serial (USB) connection. This means that by reading the serial port on the pc, it is possible to take in the GPS-signal for use in the control system. The signal from the sensor is defined as a string of characters, based on specifications defined by the National Marine Electronics Association (NMEA). A representation of such a NMEA sentence can be seen in table 6.2.

$PASHR,POS,1,6,154938.90,5715.2082709,N,01003.0444774,E,077.064,1.8,000.0,000.014,-000.005,2.4,1.6,1.8,1.2,Hp23*35

Table 6.1: NMEA sentence from the GPS-sensor. In this specific case the position is 57° 15.2082709 min. North, 10° 3.0444774 min. East

The NMEA sentence is divided by commas, between which the different portions of information is situated. The most important information is shown in table 6.2
To get the robot position, the latitude and longitude must be extracted. These are given given in degrees and minutes. This must then be transformed in to coor-

Pos	Information
2	Fix mode
3	Number of satellites
4	Time of position
5	Latitude
6	North or South
7	Longitude
8	East or West
9	Altitude
11	Heading
12	Linear speed
13	Rate of climb

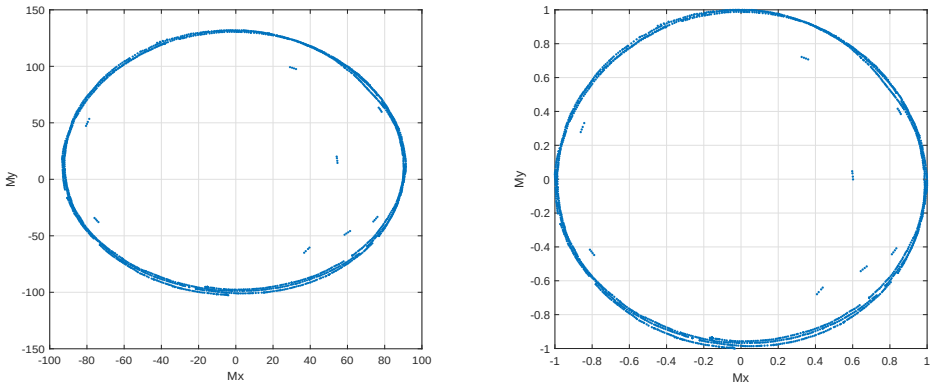
Table 6.2: The most important pieces of information that can be extracted from the NMEA sentences. Pos refers to the position between the commas.

dinates in a Cartesian space coordinate system, where we will use x_g and y_g to denote the position determined by the GPS. Furthermore the heading can also be extracted, this we will denote θ_g. As a note to this heading it must be mentioned that it must be treated carefully in the control loop. This is due to the fact that if the robot is not moving, the GPS-sensor will give a heading of 0. Therefore the velocity of the robot must be taken in to consideration before using this heading value.

During tests of the system (see section 11.6), it was unfortunately not possible to establish an RTK signal for the robot. The robot will therefore only have the raw GPS signal without any correction data from the base station. This means that the signal will drift over time. Since the prototype tests done in this report is performed over short periods of time it was decided that the signals could still be used. However if the robot, in future projects, are tested for a prolonged period, this problem must be taken in to consideration.

6.2 Magnetometer

The magnetometer on the robot is of the type OS5000 from OceanServer and is used to get a measurement on the heading. In the same way as the GPS-sensor the magnetometer is connected via USB, making it possible to read the data from the sensor in to the control system. The data is, also in this case, defined as a character string where different portions of data can be extracted. The data used is the magnetometer readings in Eastern and Northern direction. The raw data is however not fully symmetric and evenly distributed, so it must be corrected before



(a) Raw data **(b)** Corrected data

Figure 6.1: The data from the magnetometer. As seen, the raw data (a) is not distributed evenly in the x and y-directions. Furthermore it is not centred correctly. In (b) the data is corrected and calculated as unit vectors.

it can be used. This is done by measuring the raw data x_r and calculate a minimum and maximum value in the x direction, denoted by x_{min} and x_{max} respectively. The corrected value x_c can then be calculated as:

$$x_c = x_r - \frac{(x_{max} + x_{min})}{2} \tag{6.1}$$

After this correction it is useful to convert the data to a unit vector denoted M_x

$$M_x = \frac{2x_c}{(x_{max} - x_{min})} \tag{6.2}$$

The same approach is used in the y-direction yielding the unit vector M_y. The raw and corrected data can be seen in figure 6.1a and 6.1b respectively. After these corrections, the heading θ_m can be calculated as the four-quadrant inverse tangent:

$$\theta_m = atan2\left(\frac{M_y}{M_x}\right) \tag{6.3}$$

Which will give a heading between $-\pi$ and π.

6.3 Odometry

Another way of determining position and heading in mobile robotics, is to use odometry. This is done by reading signals from the rotary encoders on the wheel motors of the robot and calculate the odometry based on a kinematic model of the robot. The odometry can then describe how the robot moves in the global coordinate system. The encoder data can be accessed via the CAN bus. To make

the motor drives put out the encoder data to the CAN bus, a transmit PDO must be set-up in each drive, that reads the value of object 6064.00h. During runtime, it is then necessary to send a periodic remote transmission request to each drive specifying what PDO to activate. When a drive reads the RTR message it will read the data and transmit it on the CAN-bus.

The raw encoder data must be scaled to express a velocity on each wheel. Let o express the odometry on each wheel. This is then given by:

$$o = \frac{T_e}{T_r} C \tag{6.4}$$

Where T_e denotes the raw ticks from the encoder, T_r denotes ticks per revolution on the wheel axle and C denotes the circumference of the wheel. To simplify the odometry it was decided to treat the robot as a differential drive vehicle with only one wheel on each side. This means that the two odometry signals on each side is summed and divided by 2. o_l and o_r will hereafter denote the odometry from left and right side respectively. After this operation, we can express the heading of the robot given by the odometry, denoted as θ_o as:

$$\theta_o = \frac{(o_r - o_l)}{l} \tag{6.5}$$

The kinematic model does not consider any slippage of the wheels during turn operations. This slippage is however quite high on a skidsteered vehicle. To determine the slippage a test was performed, where the robot was turned in manual mode and the odometry was then measured. This test revealed a scaling factor of 0.54 that must be multiplied to the heading measurement.

Measured position in the x and y-direction can now be expressed as:

$$x_o = (o_l + o_r)cos(\theta_o) \tag{6.6}$$

$$y_o = (o_l + o_r)sin(\theta_o) \tag{6.7}$$

In this chapter the different sensors, used in this project, have been described. These sensor signals must be processed and used on the robot. The sensor signals however needs to be filtered and fusioned in a proper way before use. This process in done using a Kalman filter, which is described in chapter 7.

7 Kalman Filter

As described in chapter 6, different sensors are used to determine the position and orientation of the robot. These sensor signals is subject to disturbances and measurement noise. To merge these sensor signals and use the best estimate, a Kalman Filter (KF) is used, based on [13].

A Kalman filter is a recursive estimator, that in the first step, uses a model of the system, the input and the previous state to estimate the state at the current time step. In the next step the estimate is updated by measurements. A regular KF requires the system model to be linear, which our model of the robot is not. Instead, an Extended KF (EKF) is used. The EKF works by linearising the model around a working point. This is done by calculating the Jacobians.

The kinematic model used in chapter 5, equation 5.5 - 5.7 is used as model to the filter in a discretized version. With T_s representing a timestep, the model is written as:

$$x(k+1) = T_s\frac{r}{2}(v_l(k) + v_r(k))cos(\theta(k)) + z_p(k) \tag{7.1}$$

$$y(k+1) = T_s\frac{r}{2}(v_l(k) + v_r(k))sin(\theta(k)) + z_p(k) \tag{7.2}$$

$$\theta(k+1) = T_s\frac{r}{l}(v_l(k) - v_r(k)) + z_p(k) \tag{7.3}$$

z_p and z_m denotes process noise and measurement noise respectively. In both cases this noise is assumed to be Gaussian distributed with a mean of zero and a variance denoted Σ_p and Σ_m.

$$z_p(k) \sim \mathcal{N}(0, \Sigma_p) \tag{7.4}$$

$$z_m(k) \sim \mathcal{N}(0, \Sigma_m) \tag{7.5}$$

The system is represented as:

$$q(k+1) = f(q(k), u(k)) + z_p(k) \tag{7.6}$$

$$y(k) = h(q(k)) + z_m(k) \tag{7.7}$$

where q is the state vector, defined as:

$$q(k) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} \tag{7.8}$$

and $u(k)$ is the input vector defined as:

$$u(k) = \begin{bmatrix} v_l(k) \\ v_r(k) \end{bmatrix} \tag{7.9}$$

$y(k)$ is defined as

$$y(k) = \begin{bmatrix} x_g(k) \\ y_g(k) \\ \theta_g(k) \\ x_o(k) \\ y_o(k) \\ \theta_o(k) \\ \theta_m(k) \end{bmatrix} \tag{7.10}$$

The measurement vector h is defined as:

$$h = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.11}$$

Before the sensor signals can be used, the variances must be determined. This was done by programming the robot to run straight forward with a constant velocity of 0.2 m/s. The sensor signals from the odometry and magnetometer was logged and the variances measured on this data. The determination of the GPS-variance is described in section 11.6. As described in section 6.1, the GPS heading θ_g returns a zero when the velocity is zero. It can therefore be hard to know the variances at different velocity's. It was therefore decided to use the same variance as for θ_m and then perhaps adjust if problems are experienced with estimating the heading. The measured variances are shown in table 7.1.

Sensor	Variance (σ^2)
$\sigma^2_{x_g}$	$9.3 \cdot 10^{-4}$ $[m^2]$
$\sigma^2_{y_g}$	$9.3 \cdot 10^{-4}$ $[m^2]$
$\sigma^2_{\theta_g}$	$5.6 \cdot 10^{-4}$ $[rad^2]$
$\sigma^2_{x_o}$	$3.5 \cdot 10^{-6}$ $[m^2]$
$\sigma^2_{y_o}$	$3.5 \cdot 10^{-6}$ $[m^2]$
$\sigma^2_{\theta_o}$	$2.5 \cdot 10^{-7}$ $[rad^2]$
$\sigma^2_{\theta_m}$	$5.6 \cdot 10^{-4}$ $[rad^2]$

Table 7.1: The measured variances for the sensors

Before running the filter, the initial state and covariance matrices must be determined. This can either be done by setting the initial state estimate to zero and the covariance large. The other approach is to get a measurement from the sensors and use this as a best initial guess combined with a relative low covariance. The latter approach is used in this project by running a small script that runs until sensor data from the GPS and Magnetometer is received. This data is then used as a best initial guess. With P denoting the covariance matrix, The initial covariances was set to:

$$P_0 = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \tag{7.12}$$

The Kalman filter is now ready to run. Every iteration can be split in two parts: prediction and update.

Prediction

$\hat{q}(k+1|k)$ denotes the estimated state at the next timestep, given the current state and inputs:

$$\hat{q}(k+1|k) = f(\hat{q}(k|k), u(k)) \tag{7.13}$$

$$\tag{7.14}$$

Calculating the covariance is done by:

$$P(k+1|k) = F(k)P(k|k)F(k)^T + \Sigma_p \tag{7.15}$$

where F is a Jacobian matrix:

$$F = \left.\frac{\partial f}{\partial q}\right|_{\hat{q}(k|k),u(k+1)} \tag{7.16}$$

At this point a prediction of the current state has been calculated, based on the system model and the input. Furthermore the covariance has been calculated. In the next step the measurements are taken into consideration:

Update

The innovation residuals, denoted $\tilde{y}$ are calculated as:

$$\tilde{y}(k+1) = y(k+1) - h\hat{q}(k+1|k) \tag{7.17}$$

followed by the innovation covariance S

$$S(k+1) = H(k+1)P(k|k-1)H^T(k+1) + \Sigma_m \tag{7.18}$$

where H is a Jacobian matrix

$$H = \left.\frac{\partial h}{\partial q}\right|_{\hat{q}(k+1|k)} \tag{7.19}$$

The Kalman gain (K) is calculated by:

$$K(k+1) = P(k+1|k)H(k+1)S^{-1}(k+1) \tag{7.20}$$

The states are now updated, using the Kalman gain and the residuals:

$$\hat{q}(k+1|k+1) = \hat{q}(k+1|k) + K(k+1)\tilde{y}(k+1) \tag{7.21}$$

Finally, with I_m denoting the identity matrix, the covariance is updated:

$$P(k+1|k+1) = (I_m - K(k+1)H(k+1))P(k+1|k) \tag{7.22}$$

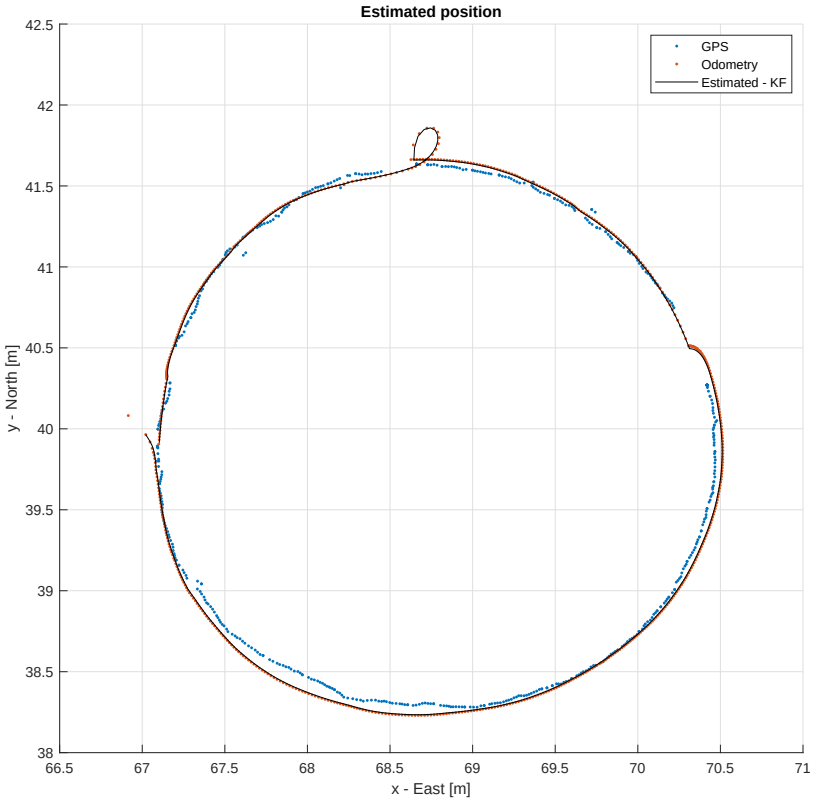


Figure 7.1: A test of the KF estimating the position. The robot was programmed to run in a circle. The KF is capable of following the sensor signals quite well. Note that on the upper right side, the GPS signal is missing, for approx 0.5 m, but the filter is still able to estimate an almost correct position.

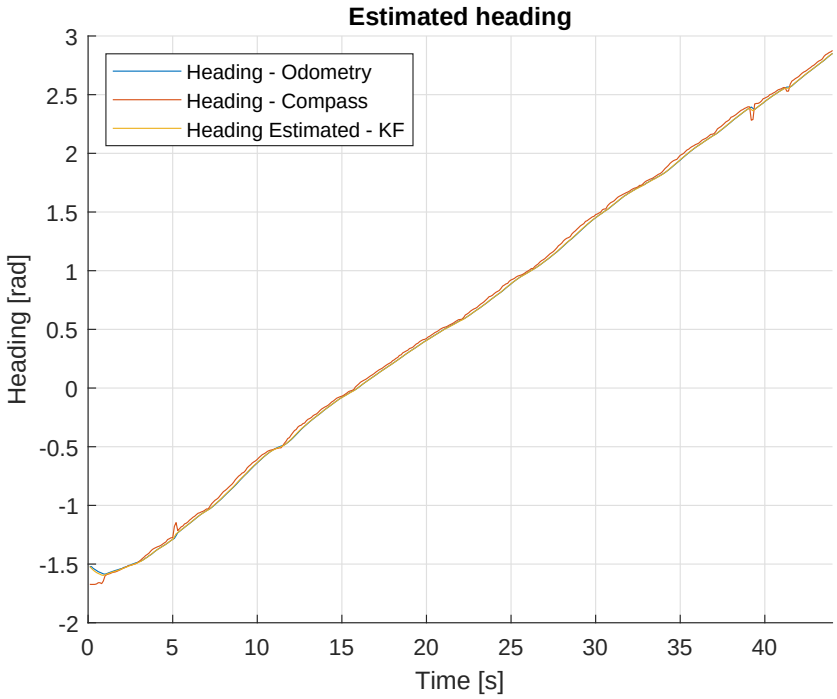


Figure 7.2: A test of the KF estimating the heading. As in figure 7.1, the robot was programmed to run in a circle. At the beginning of the test there is a deviation, but after less than two seconds, the filter has converged to the sensor value.

Regarding the sensor fusion in the Kalman filter, it must be noted that the GPS value is an absolute value, which is correct within the boundaries of the drifting and variance of the signal. Opposed to this, the odometry will drift over time, resulting in a divergence between the two signals. A correction value for the odometry must therefore be maintained in the loop. This is done by subtracting the odometry values from the estimated position of the Kalman filter, which is the "best guess" at any given moment.

The filter was tested by programming the robot with a steady velocity and angular velocity. This made the robot run in a circle. Meanwhile the sensor signals were logged together with the estimated values from the EKF. The result can be seen in figure 7.1 and 7.2 showing position and heading respectively. The filter works very well, and is capable of merging and follow the sensor signals.

This chapter has now described how the sensor signals have been fusioned and filtered. Before the robot can be controlled in the field, a trajectory must be build that it can follow. This will be discussed in chapter 8.

8 Trajectory planning

As described in section 2.6 the robot needs a trajectory to follow. This trajectory must be used by the controller and contain reference points that the robot should visit. The overall planning and control of a mobile agricultural robot can be split in four levels:

1. **Strategic level** - Can be seen as "mission planning", where the farm manager develops a plan for what the robot should do in the field and how this work should be done.

2. **Global level** - Based on the strategic planning, the global planner must generate trajectory points that the robot must visit in the correct order.

3. **Local level** - The local planner is responsible of translating the reference points generated in the global planner to as desired velocity and angular velocity of the robot

4. **Hardware level** - At the hardware level, the desired velocity and angular velocity of the robot must be transformed to an actual voltage and current on each motor.

The strategic and global levels can be done off-line before the robot starts to drive. These are discussed in this chapter. The local and hardware levels must be handled on-line in the robot controller, while the robot is driving. This will be discussed during development of the controller in chapter 9.

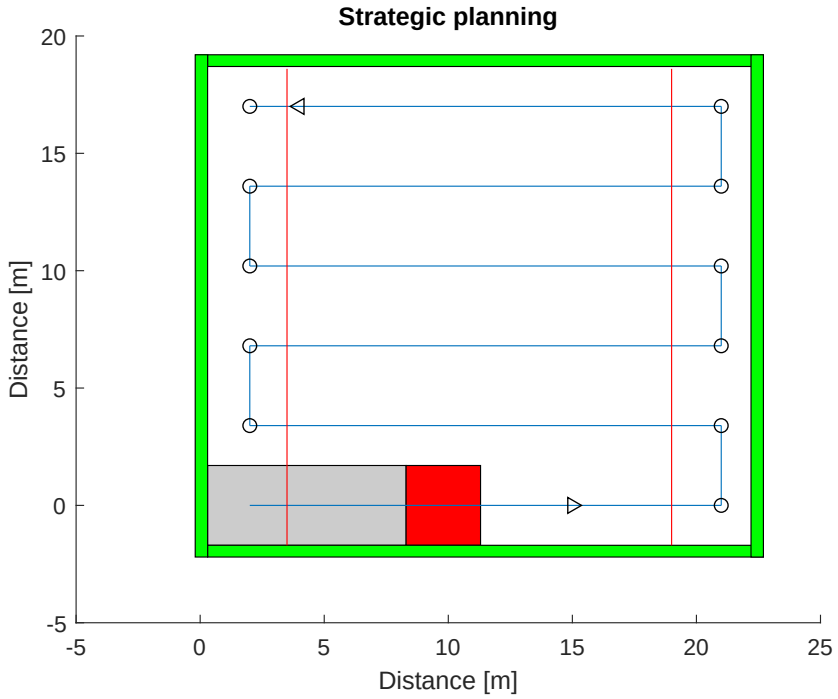


Figure 8.1: A simplified representation of strategic planning. The field is bounded by the green lines. The red square represents the robot with attached machinery, while the grey represents already treated area. The red lines represents the boundary between the headland and the main area. According to desired direction, working width, headlands etc. the robot must follow a certain trajectory to ensure full treatment of the field.

8.1 Strategic planner

All though a strategic plan for a field robot is not the main objective of this report we will just briefly visit some of the considerations that must be taken. Based on the job at hand, the strategic plan should describe which kind of machinery or manipulators that should be attached to the robot. The attached machinery will define the working width, which is an important parameter when a global plan should be developed.

The working direction in the field must also be determined. It is not always desirable to treat a field in the same direction or order every time, since this could potentially create areas of compacted soil, which makes the field uneven. Also fields with previous row crops can have an uneven and rough surface. A switching between directions will make the field more even and easy to prepare for the next crop.

Headlands will also have to be considered. When the robot reaches the end of the field it must turn around and go back in the direction parallel to the previous direction. In most cases the attached machinery will not be able to make a sharp turn, which means that it must be lifted from the ground while the robot makes the turn. When the robot again is situated at the correct position and orientation, the machinery can be lowered to the ground and the work can continue. This leads to the use of headlands, that are left untreated at the first operation. When the main area of the field is treated, the headlands can then be treated in a direction perpendicular to the main direction.

A theoretical and simplified representation of these aspects, can be seen in figure 8.1.

8.2 Global planner

For this project a global planner was developed that can generate a trajectory, which can be used by the robot controller. The approach is inspired by how the GPS field maps for modern farm tractors are build. these maps are generated by defining a line between two points, denoted A and B, in the field. The tractor computer then generates lines, parallel to the first line, that the tractor must follow. The line following and coverage of the machinery, is then shown on a monitor in the tractor cabin. (see figure 8.2)

The global planner build for this project, works by using code-blocks that generates straight and circular trajectory's. All though this is not a very high level approach, it fills the need for trajectory building in this project.

In the beginning of the script, the A and B points, are set. These points marks the beginning and end of the first straight line to follow. This approach means that when the first line is set, the field will be treated in that direction. The user can now build up a desired trajectory by using the predefined code-blocks. These blocks are put together to form one big array of data points, that contain information. Each point can then act as a reference to the controller. In this project, a reference point contains x and y-position in S, denoted x_r and y_r along with a desired orientation of the robot, denoted θ_r. The trajectory is build such that the distance between each data-point in the field is 1 cm. This ensures a very precise movement of the robot.

Every code-block is build such that it passes the generated information onto the following block. This information contains the index n of the trajectory array along with the generated x_r, y_r and θ_r values. The next block can then read this information and expand the array with the next set of values.

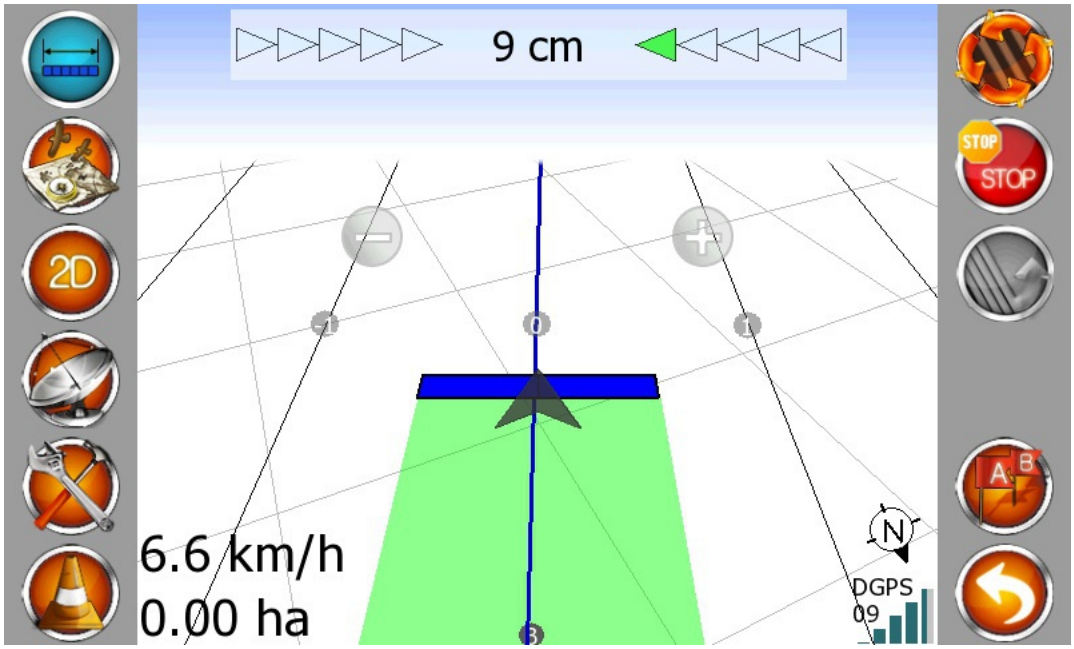


Figure 8.2: A typical screen-shot of a tractor monitor. The big arrow in the middle shows where the tractor is situated according to a predefined line. Parallel line are also shown, which the tractor will use in the following pass.

As mentioned earlier, the code blocks are split in two types defining straight and circular movements: In the straight line blocks, the user sets a desired length of the line. The code-block then simply uses a for-loop, that, according to the direction given by the previous block, generates points for each cm. along that line. These blocks are build for both forward and backward movement.

The circular blocks are split in four different types: left and right forward and left and right backwards. The circular blocks defines 90 degrees turns and works by defining a circle with a radius of 1 meter. The script then uses that circle and generates a trajectory along $\frac{\pi}{2}$ of the perimeter.

A sequence of a generated trajectory example can be seen in table 8.1 The trajectory could also have contained information of desired velocity and angular velocity. Also information of headland boundaries could be included, which could be used by the robot to define when attached machinery should be taken out or in to the ground. This is left for future development. A plot of a generated trajectory example can be seen in figure 8.3.

This chapter has now described how trajectory's can be generated using a strategic

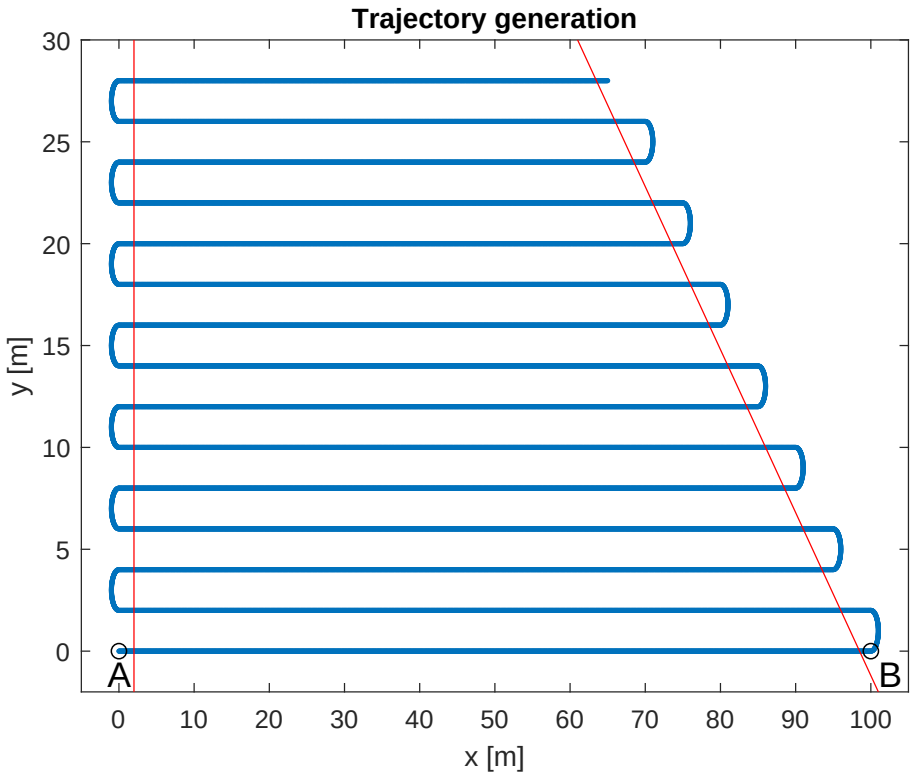


Figure 8.3: A trajectory generated by the script, on an imaginary example of a field. The trajectory starts at the point A. The first line is defined by the A-B line. Then two 90 degrees left turns are added, where after a new straight line is generated, followed by two 90 degrees right turns. This process must be continued until the field is covered. The red lines represents the boundary between the headland and the main area.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
x_r	1.56	1.57	1.58	1.59	1.60	1.61	1.61	1.62	1.63	1.64	1.64	1.65	1.66	1.67	1.67	1.68	1.69	1.70	1.70	1.71
y_r	0.17	0.18	0.19	0.19	0.20	0.20	0.21	0.22	0.22	0.23	0.24	0.24	0.25	0.26	0.26	0.27	0.28	0.28	0.29	0.30
θ_r	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.71	0.72	0.73	0.74	0.75	0.76	0.77	0.78	0.79

Table 8.1: A sequence of a generated trajectory, during a turn in the counter clockwise direction. The four rows contains the index n along with $x_r[m]$, $y_r[m]$ and $\theta_r[rad]$.

and global approach. A local level planner (controller) must now be developed that can make the robot follow the predefined reference point in the trajectory. An investigation of these topics are done in chapter 9.

9 Controller

As described in chapter 8, the robot needs a local planner (controller), that can make the robot follow the reference points generated from the global planner. Furthermore a hardware controller must translate the signals to an actual current on the motors. The development of these concepts are done in this chapter and inspired by [3], [7] and [21].

9.1 Local Controller

As in chapter 5, figure 5.1, the robot is situated in the global coordinate system S with x and y denoting the position and θ denoting the heading.

The robot must follow a trajectory, described in chapter 8, which is an array of reference points, where x_r and y_r defines reference points in the x and y-direction of S, while θ_r denotes the reference heading.

The robot must now track these reference points one by one, as time progresses forward. The controller shall minimize the error, denoted e_f, between the reference and actual position of the robot. Increasing the index n in the trajectory, will make the reference points jump forward. As the reference points are moving forward the robot will start to move forward as the controller minimizes e_f.

Next step is to decide how the reference index n should increase. This can either be triggered when e_f is below some predefined threshold, or it can be done with a predefined time interval. In this project it was decided to use the latter approach. This has the advantage that this time interval can be used to control the velocity of the robot. If the interval is low the robot will move faster and visa versa. It is therefore not neccesary to include a reference velocity in the trajectory.

We now need to define the robot position and orientation with respect to a desired reference point in the trajectory. An overview of this approach can be seen in figure

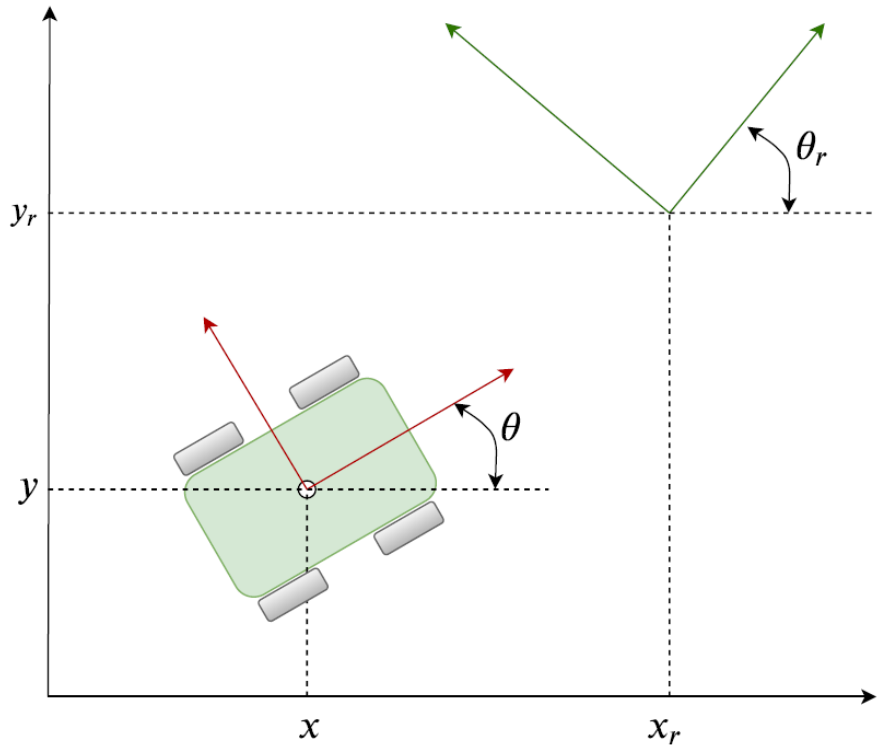


Figure 9.1: The robot is situated in the global frame S with an x and y-position and θ defined as the angle between the forward longitudinal direction of the robot (red frame) and the global x-axis . In the same way the reference frame (green) represents position and orientation by x_r, y_r and θ_r. The task for the controller is now to minimize the error e_f in position and orientation between the red and green frames.

9.1.

We now describe position by the vector p and reference position by the vector p_r:

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \tag{9.1}$$

$$p_r = \begin{bmatrix} x_r \\ y_r \end{bmatrix} \tag{9.2}$$

In the same way we use a direction vector d and a reference direction vector d_r,

which both describes the direction as unit vectors.

$$d = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \tag{9.3}$$

$$d_r = \begin{bmatrix} \cos(\theta_r) \\ \sin(\theta_r) \end{bmatrix} \tag{9.4}$$

It is now clear that if e_f goes to zero, the differences $p_r - p$ and $d_r - d$ also converges to zero, which will make the robot track the reference frame. With v describing linear velocity, while ω describes angular velocity, the kinematic model can be described as:

$$\dot{p} = dv \tag{9.5}$$

$$\dot{d} = Rd\omega \tag{9.6}$$

where R is a 90 degrees counter-clockwise rotational matrix given by:

$$R = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{9.7}$$

With $\dot{p}_r$ denoting the derivative of p_r and $\dot{d}_r$ denoting the derivative of d_r, we can now calculate the difference as:

$$\dot{p} - \dot{p}_r = dv - d_r v_r \tag{9.8}$$

$$\dot{d} - \dot{d}_r = R(d\omega - d_r\omega_r) \tag{9.9}$$

Where v_r and ω_r express reference velocity and angular velocity. For control purpose it is desirable to have a linear system. We therefore use a Taylor approximation and linearise around the reference point:

$$\begin{aligned} \dot{p} - \dot{p}_r = f(d, v) &\approx f(d_r, v_r) + (d - d_r)\frac{\partial f(d, v)}{\partial d}\bigg|_{\substack{d=d_r \\ v=v_r}} + (v - v_r)\frac{\partial f(d, v)}{\partial v}\bigg|_{\substack{d=d_r \\ v=v_r}} \\ &= (d - d_r)v_r + (v - v_r)d_r \end{aligned} \tag{9.10}$$

$$\begin{aligned} \dot{d} - \dot{d}_r = f(d, \omega) &\approx f(d_r, \omega_r) + (d - d_r)\frac{\partial f(d, \omega)}{\partial d}\bigg|_{\substack{d=d_r \\ \omega=\omega_r}} + (\omega - \omega_r)\frac{\partial f(d, v)}{\partial v}\bigg|_{\substack{d=d_r \\ \omega=\omega_r}} \\ &= R(d - d_r)\omega_r + Rd_r(\omega - \omega_r) \end{aligned} \tag{9.11}$$

The errors in position and orientation, denoted with subscript e, are now calculated and projected on to the reference frame.

$$p_{ex} = (p - p_r)^T d_r \tag{9.12}$$
$$p_{ey} = (p - p_r)^T R d_r \tag{9.13}$$
$$d_{ex} = (d - d_r)^T d_r \tag{9.14}$$
$$d_{ey} = (d - d_r)^T R d_r \tag{9.15}$$

To see how the errors evolve, we differentiate with respect to time, using equation 9.10 and 9.11:

$$\begin{aligned} \dot{p}_{ex} &= (\dot{p} - \dot{p}_r)^T d_r + (p - p_r)^T \dot{d}_r \\ &= v_r (d - d_r)^T d_r + (v - v_r) d_r^T d_r + (p - p_r)^T R d_r \omega_r \\ &= v_r d_{ex} + v - v_r + P_{ey} \omega_r \end{aligned} \tag{9.16}$$

$$\begin{aligned} \dot{p}_{ey} &= (\dot{p} - \dot{p}_r)^T R d_r + (p - p_r)^T R \dot{d}_r \\ &= v_r (d - d_r)^T R d_r + (v - v_r) d_r^T R d_r + (p - p_r)^T R R d_r \omega_r \\ &= v_r (d - d_r)^T R d_r + (v - v_r) - (p - p_r)^T d_r \omega_r \\ &= v_r d_{ey} - P_{ex} \omega_r \end{aligned} \tag{9.17}$$

$$\begin{aligned} \dot{d}_{ex} &= (\dot{d} - \dot{d}_r)^T d_r + (d - d_r)^T \dot{d}_r \\ &= (R(d - d_r))^T d_r \omega_r + (R d_r)^T d_r (\omega - \omega_r) + (d - d_r)^T R d_r \omega_r \\ &= (RR(d - d_r))^T R d_r \omega_r + (d - d_r)^T R d_r \omega_r \\ &= -(d - d_r)^T R d_r \omega_r + (d - d_r)^T R d_r \omega_r \\ &= 0 \end{aligned} \tag{9.18}$$

$$\begin{aligned} \dot{d}_{ey} &= (\dot{d} - \dot{d}_r)^T R d_r + (d - d_r)^T R \dot{d}_r \\ &= (R(d - d_r))^T R d_r \omega_r + (R d_r)^T R d_r (\omega - \omega_r) + (d - d_r)^T R R d_r \omega_r \\ &= (RR(d - d_r))^T R R d_r \omega_r + (\omega - \omega_r) - (d - d_r)^T d_r \omega_r \\ &= (d - d_r)^T d_r \omega_r + (\omega - \omega_r) - (d - d_r)^T d_r \omega_r \\ &= \omega - \omega_r \end{aligned} \tag{9.19}$$

which means that we now have a linear time variant system, that in state-space form looks like

$$\begin{bmatrix} \dot{p}_{ex} \\ \dot{p}_{ey} \\ \dot{d}_{ex} \\ \dot{d}_{ey} \end{bmatrix} = \begin{bmatrix} 0 & \omega_r & v_r & 0 \\ -\omega_r & 0 & 0 & v_r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{ex} \\ p_{ey} \\ d_{ex} \\ d_{ey} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v - v_r \\ \omega - \omega_r \end{bmatrix} \tag{9.20}$$

From this state-space form it is seen that the state d_{ex} cannot be manipulated and thereby driven to zero. However this will go to zero when d_{ey} goes to zero. With

the system reduced to three states, the state vector denoted Φ is defined as:

$$\Phi = \begin{bmatrix} p_{ex} \\ p_{ey} \\ d_{ey} \end{bmatrix} \tag{9.21}$$

and the input vector u, defined as:

$$u = \begin{bmatrix} v - v_r \\ \omega - \omega_r \end{bmatrix} \tag{9.22}$$

The state feedback, $u = -K\Phi$, with $K_1 - K_3$ denoting feedback gains, can be written as:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} -K_1 & 0 & 0 \\ 0 & -K_2 & -K_3 \end{bmatrix} \begin{bmatrix} p_{ex} \\ p_{ey} \\ d_{ey} \end{bmatrix} + \begin{bmatrix} v_r \\ \omega_r \end{bmatrix} \tag{9.23}$$

Now we need to calculate the gains. This can be done using a Linear Qaudratic Regulator (LQR).

9.2 LQR

LQR is an optimal controller that solves the optimization problem by minimizing the performance function J:

$$J = \int_0^{\inf} (\Phi^T Q \Phi + u^T R u) dt \tag{9.24}$$

Where $Q \in \mathbb{R}^{3\times3}$ and $R \in \mathbb{R}^{2\times2}$ are weight matrices relating to state and input respectively. These matrices penalize the deviation from zero and can be seen as tuning parameters for the controller. During the initial design, it can be difficult to know how to set the values in these matrices correctly. One approach is to use Brysons rule, which state that the weights can be calculated as:

$$Q_{ii} = \frac{1}{Q_{m,i}^2} \tag{9.25}$$

$$R_{ii} = \frac{1}{R_{m,i}^2} \tag{9.26}$$

where $Q_{m,i}$ and $R_{m,i}$ represents the maximum acceptable value of the i^{th} state of Q and R respectively.

Now we must decide what maximum values can be accepted at the state and input. If we want the robot to move precisely to the reference position, the position errors p_{ex} and p_{ey} must be low. We will set the maximum acceptable error to 0.05 m in both x and y-direction. The same goes for the direction, where we set the maximum acceptable error to sin(0.01 rad). For the input it is hard to know how strong it should be to reach the desired reference, but a low value should make the robot move smoothly. We will therefor initially set it to a low value of 0.1 for both entrances and see how the system reacts.

After these considerations Q and R are defined as:

$$Q = \begin{bmatrix} 400 & 0 & 0 \\ 0 & 400 & 0 \\ 0 & 0 & 100 \end{bmatrix} \tag{9.27}$$

$$R = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \tag{9.28}$$

Calculating the gains are now done using the lqr() fuction in Matlab. With the state and input vectors defined along with the weight matrices, the gain matrix K is calculated as:

$$K = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 1 \end{bmatrix} \tag{9.29}$$

With these gains we have now solved the local control problem. The last step is now to use equation 5.8 and 5.9 to map v and ω to v_l and v_r, which was our desired velocity on left and right wheel-pair.

9.3 Hardware controller

The hardware controller was already build in to the drives previous to this project. This controller is closing the inner control loop, meaning that it takes the output from the local controller and uses this as a reference, to calculate the current on each motor of the robot. This works by calculating an error between the desired and actual velocity for each motor (v_l and v_r in this case). This error is now related to a PI-gain. Tests showed that this loop was already proper tuned, which mean that further investigation in to these gain levels, are not crucial for this project. It is however possible to adjust these gains and tune the loop in the DriveWare software package. A schematic overview of the overall control system, can be seen in figure 9.2.

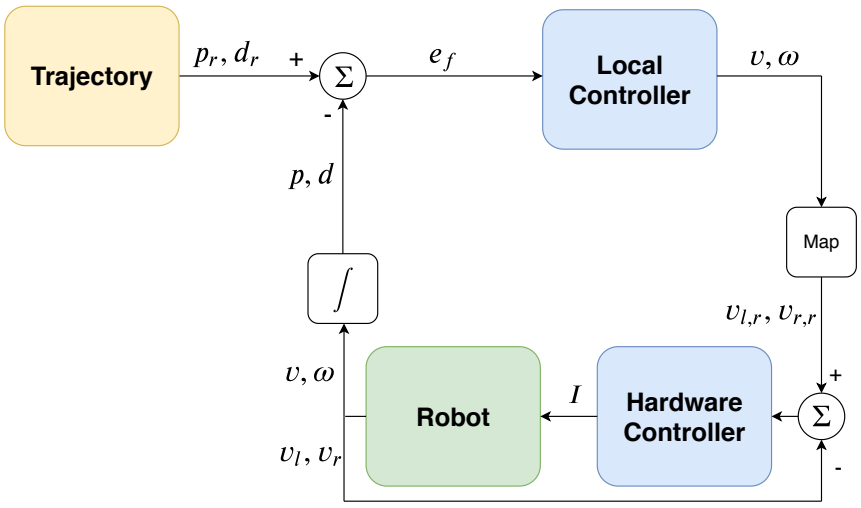


Figure 9.2: The block diagram represents the signal flow in the control systems. The local controller outputs the desired v and ω, which through a mapping is converted to $v_{l,r}$ and $v_{r,r}$ which again is a reference signal to the hardware controller. The hardware controller uses a PI-loop to calculate a current for the motors. This will give a velocity v_l and v_r of the wheels which are used as feedback. These velocity's will also result in movement of the robot, producing an overall velocity v and angular velocity ω. Through a natural integration of the system, the robot will also have a position p and a direction d, which are subtracted from the reference points p_r and d_r. The error e_f are now used as a feedback signal to the local controller.

The controller was tested while the robot was situated in the field. This showed some challenges for the robot, especially during sharp turns. These tests are described in chapter 11. Before moving to the tests, the implementation will be described in chapter 10.

10 Implementation

The implementation of the control system is done on a pc. This pc is what was referred to as the "central control unit" in section 3.1. The software used to control the robot is built in Matlab, which has a lot of pre-build toolboxes and functions that makes implementation easier. The most important functions, along with a brief explanation on how the software is built is now presented. The first section describes the setup procedure and the second the main loop.

10.1 Setup

First, the pc must be connected to the hardware. As described in chapter 4, the drives are connected via CAN-bus using a CAN/USB adapter from Peak Systems. The sensors are connected to the pc via a USB hub.

Matlab has a "Vehicle Network Toolbox" that supports sending and receiving messages via Can-bus. First, a CAN-bus channel must be set up, that makes Matlab communicate with the CAN/USB adapter. This is done using the function "canChannel()" which take the specified adapter and bus-speed as input arguments. See appendix C.1.1. Be aware that the functions in Matlab use base-10 numbering system, while the documentations manual for the drives use hexadecimal numbers.

At start-up, the drives must be reset. To do this, two NMT messages are sent to each drive: The first one resets the drive and the second starts the communication, as explained in section 4.3. An example is shown in table 4.2. The function "transmit()" are then used to put these messages on to the CAN-bus. See the code in appendix C.1.2. The drives are now ready to receive information from the CAN-bus.

During runtime three different PDO-messages must be sent to each drive. These messages contains no data, but the RTR bit must be set high. This will trigger the drives to put a live signal, the actual position and the actual current on the CAN-bus. The PDO id's are set in the drives (see figure A.3 in appendix A.1). The pc

must now send the PDO-messages to these id's periodically. This ensures that the drives will transmit the information in a regular manner. The function "transmitPeriodic()" was used for these messages. This function can run in the background, without obstructing the main loop used for control.

The last message sent, is the actual command message. This message must be packed with the command velocity for each drive, calculated by the controller. An example can be seen in table 4.3. Also this message is sent periodically. The code for all four message types are found in appendix C.1.3. Note that the least significant byte in the command message must contain the value 15. This is not documented in the communication manual, but was discovered during the initial testing of the system.

The serial connections for the magnetometer and GPS sensors are set-up using the function "serial()". This function takes the COM-port, specified by the pc, and the Baud-rate as input. The Baud-rate is 115200 bit/sec for the GPS and 19200 bit/sec for the magnetometer. The function "fopen()" opens the connections. The code is in appendix C.1.4.

Before running the main loop, a trajectory must be defined. This is explained in chapter 8. The code is built with a small initial script that defines the A - B line. It then calculates the heading and length of that line and build trajectory points along it, with a distance of 1 cm (see appendix C.1.5).

After the initial trajectory-script, small code-blocks can be added, which generates 6 different trajectory segments:

1. Left turn forward
2. Right turn forward
3. Line segment forward
4. Left turn backward
5. Right turn backward
6. Line segment backward

These code-blocks makes it possible to generate a trajectory, by connecting them in a long "chain". All the code blocks carry four different values: x and y-position, heading and the index. These values must be available for the next code-block in the chain, so that it can calculate the next segment in the trajectory. The code for the "Right turn forward" and "Line segment forward", is in appendix C.1.6 and

C.1.7.

Now the initial measurements from the GPS and magnetomer are retrieved. As described in chapter 7, the initial state of the Kalman filter is calculated by using measurements from the GPS-sensor and the magnetometer. This is done by running a script that recieves and calculates these values, which then are used as a best initial guess for the Kalmamn filter. How these scripts are built is described in the next section.

This section has now described the most important set-up topics. The next section will describe how the main loop is build.

10.2 Main loop

To run the main loop at a fixed frequency, the function "robotics.rate()" is used. This function is contained in the "Robotics System Toolbox". It was decided to run the main loop at 10 Hz.

The first code snippet in the main loop, uses the function "receive()" to read the input buffer of the CAN-bus connection. The "extractRecent()" function is then called to extract the most resent messages from the drives. Each drive is sending two messages: The actual position and actual current. Since four drives are connected, this gives a total of eight messages. The "unpack()" function is then used for unpacking the data, so it can be used in the control loop. The code for this part is in appendix C.1.8.

The GPS signal is now received by using the function "fscanf()" to read the serial buffer. This will produce the NMEA-sentence from the GPS-sensor. The length of the sentence received, must be either 115 or 116 characters. Furthermore the first 10 characters must be "$PASHR,POS". This is controlled, using an if/else statement. If the sentence is deemed ok, the function "regexp()" is used to determine the comma positions. The latitude, longitude and heading, can now be subtracted using "extractBetween()" to subtract the data between two distinct comma positions. These values are now transformed to Cartesian coordinates using the function "cassinifwd()", which is contained in the "GeographicLib" toolbox. When using this function a reference value must be given for the longitude and latitude. This reference will mark the origin of the generated Cartesian space.

Since the GPS-sensor is mounted with an offset to the centre of the vehicle this must also be considered. This is done by subtracting an offset value. The code for the GPS sensor is found in appendix C.1.9

Next, the readings from the magnetometer is received in the same way as for the GPS-sensor. The values extracted, have an offset and must be corrected, as described in section 6.2. After this correction, the function "atan2()" calculates the four-quadrant inverse tangent, which defines the heading between $-\pi$ and π. Note that the magnetometer calculates heading as positive in the clockwise direction opposed to the GPS. The heading is therefore negated to align with the GPS. Furthermore the magnetometer calculates heading zero pointing North. This is corrected such that heading zero is pointing East. The code for the magnetometer is seen in appendix C.1.10.

The position values from the drives are now used to calculate the odometry, as explained in chapter 6.3. See code in appendix C.1.11.

Now, the Kalman filter is used as described in chapter 7. After running the Kalman filter, the correction value for the odometry must be maintained. This is done by subtracting the estimated state from the odometry signal. This correction value is then subtracted from the odometry before it is fed back in to the Kalman filter in next iteration.

The controller now calculates the error between the reference state from the trajectory and the estimated state from the Kalman filter. These errors is then multiplied with the gains. This is described in chapter 9 After mapping the velocity and the angular velocity in to a velocity for the left and right wheels, the values are packed in the command CAN-messages using the "pack()" function and transmitted on the CAN-bus. The code for the main loop is in appendix C.1.12

After implementing all the aforementioned topics, the system is now ready for test. The robot should now be ready to follow a given trajectory in a controlled manner. This will be tested in chapter 11.

11 Testing

Different tests, done with the robot will now be described. First an overall test of the systen is performed, which tests the ability of the controller to make the robot follow a trajectory.

11.1 Testing the controller

To test the controller, the robot was taken to the field (see figure 11.1). A simple trajectory was generated and the robot was then commanded to start at trajectory index 1. It was decided to drive the robot with a velocity of 0.4 m/s, which gives a smooth drive. The control loop runs with a frequency of 10 Hz. By increasing the trajectory index with 4 steps per iteration, the index number increases with 40 Hz. Since the distance in between each point in the trajectory is 1 cm, this will give the desired velocity.

As seen in figure 11.2 the robot follows the trajectory to the end. The robot follows the straight lines reasonably, however during turning it has severe problems. It was not able to follow the trajectory turns as good as expected. It was unclear what caused the problems in the turns but it seemed as the motors sometimes had problems driving the wheels fast enough. It was therefore decided to test the current use during turning operations.

11.2 Determining use of current

During tests with the robot it is interesting to see how much current the motors use. The drives put information of the current on to the CAN bus by setting up a transmit PDO that reads the value of object 6077h. A test was done, where the robot was driving with a steady velocity on a flat surface, while the current drawn in each motor was logged. A result of this test can be seen in figure 11.3. The raw current values have a very high variance, while the robot is driving. This is probably due to the fact that each drive is controlling the individual velocity of a motor. To make more sense of the values, the current use of each motor can be

Figure 11.1: The robot at the field where the controller test was done.

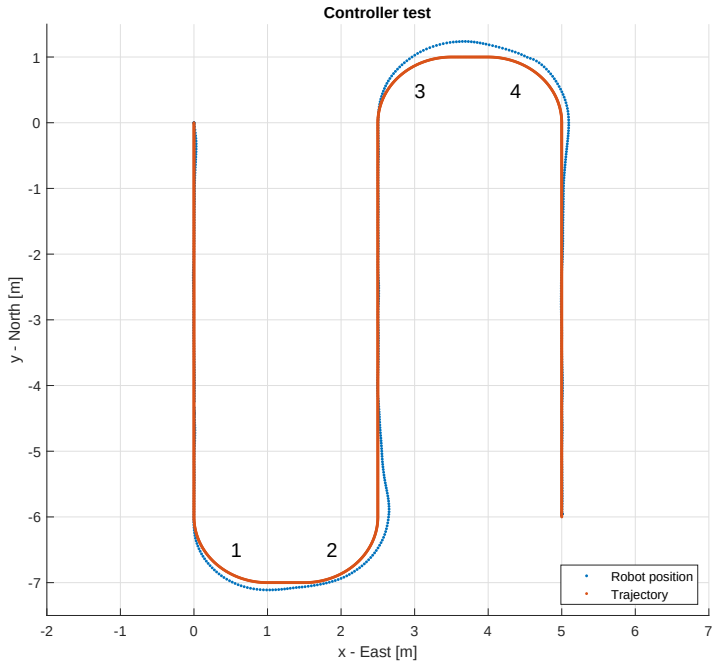


Figure 11.2: The figure shows a field test with the robot. A simple trajectory was generated. The robot follows the trajectory quite well on the straight lines, but when turning it has big problems. The biggest error occurs between turn 3 and 4, where the robot is positioned 0.23 m wrong.

summed. This value can then be smoothed by a moving average. The result can be seen in figure 11.4

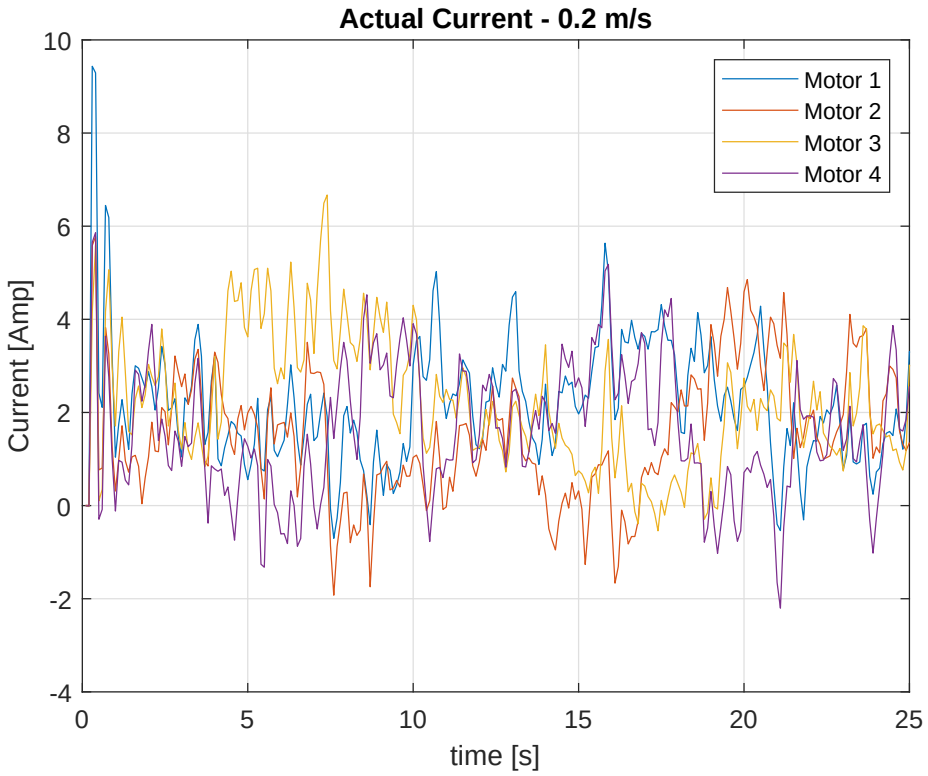


Figure 11.3: The actual current used in each motor, when the robot drives with a velocity of 0.2 m/s. This test was done on a flat surface of gravel. As seen the current is varying quite a lot, which probably is a consequence of having four drives in the robot, which each runs a closed loop velocity control.

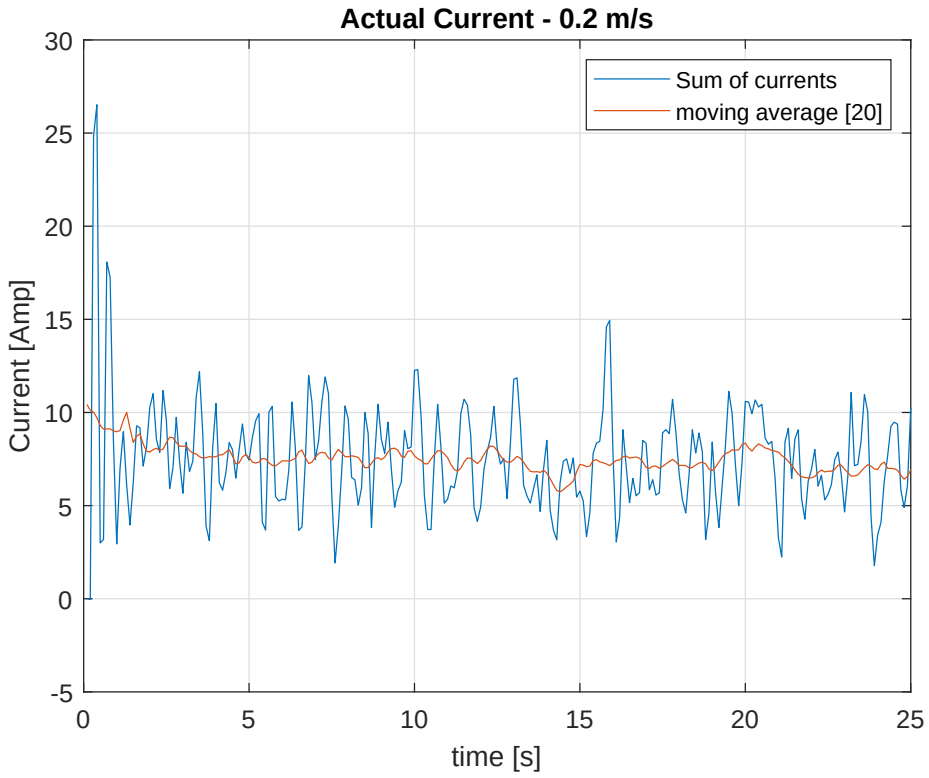


Figure 11.4: Sum of current for all four motors. The values are smoothed with a 2 seconds moving average (red line). The current draw at the beginning is significant higher due to acceleration, but already after approx 1 sec. the robot reaches a steady velocity.

11.3 Maximum current

During tests of the robot, a strange behaviour of the robot was sometimes observed. Especially on a terrain with grass, the robot have difficulties of turning with a small turning radius. If the robot is commanded to turn with this small radius, some or all of the wheels will stop turning after a few seconds. It was discovered, that the reason for this, is that the maximum current of the motor drives have been reached and they will start to turn down the current.

The drives are configured such that they have a maximum continuous current rating (20 Amp.) and a peak current rating (40 Amp.). In the drives a fold-back period is defined, meaning the time it takes for the drive to decrease the current from peak to continuous level. If the peak current is reached the drive will only maintain this level for a few seconds and then turn down the current over the fold-back period until the max. continuous level is reached. After this period, the drive will only deliver the continuous level of current. If the current, needed to turn the robot,

exceeds 20 Amp., the result is that the robot is not capable of driving further, or perhaps it turns in a wider radius than expected.

To test this behaviour, one of the wheels were blocked and the current level logged. The result can be seen in figure 11.5, where the peak level, fold-back region and continuous levels are clearly shown. The maximum, peak and continuous levels together with the fold-back period can all be configured in the drive. This is documented in the communication manual.

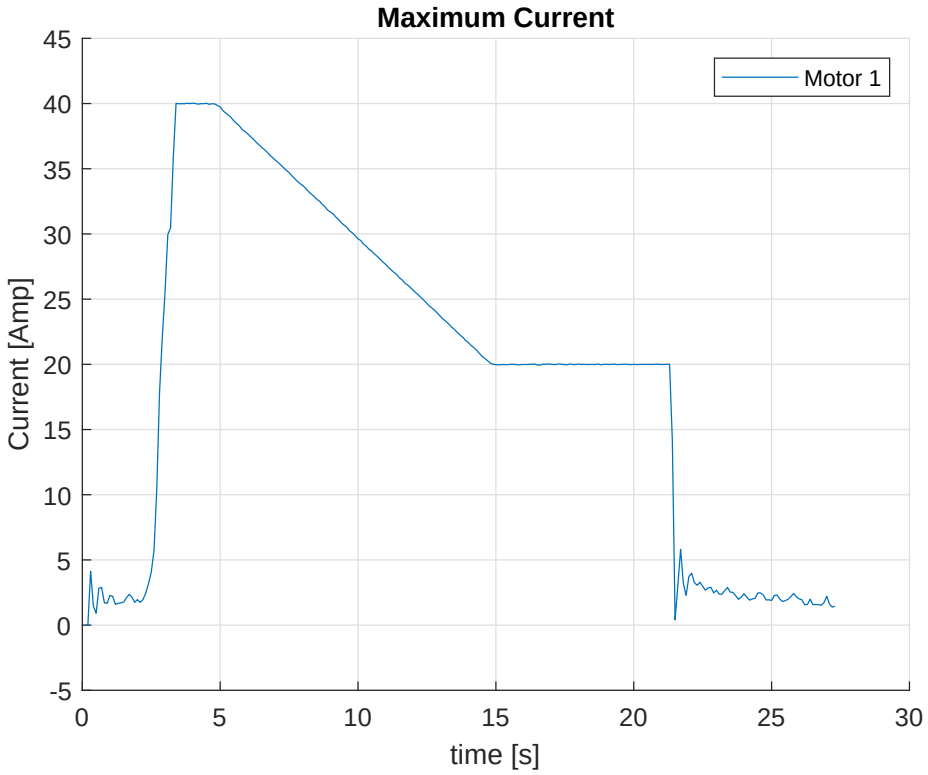


Figure 11.5: The figure shows how the drive reacts when a wheel is blocked. At approx. 2.5 seconds wheel 1 is blocked. The drive increases the current until it peaks at 40 Amp. The drive maintains the peak current for 2 seconds and then starts to decrease the current down to 20 Amp. over the fold-back period. After the fold-back period the drive maintains a maximum level of 20 Amp. At approx 22 seconds the wheel is released again.

To test the outcome of these limitations, the robot was programmed to turn around its own axes. This test was performed on a terrain with grass. The result can be seen in figure 11.6. The robot turns for a few second, where after it stops completely and is not capable of performing the task. In the same way the robot was programmed to perform a turn with a bigger turning radius of 4 meter (see figure

11.7). In this test current draw on the wheel motors on the outer perimeter is restricted, while the inner motors stay below the continuous level.

The same tests was performed on a terrain of dirt, where the robot is capable of turning more freely due to a lower friction between the wheels and the ground. These results can be seen in figure 11.8 and 11.9. The conclusion to these tests is that these current restrictions must be taken into consideration when developing control algorithms for the robot.

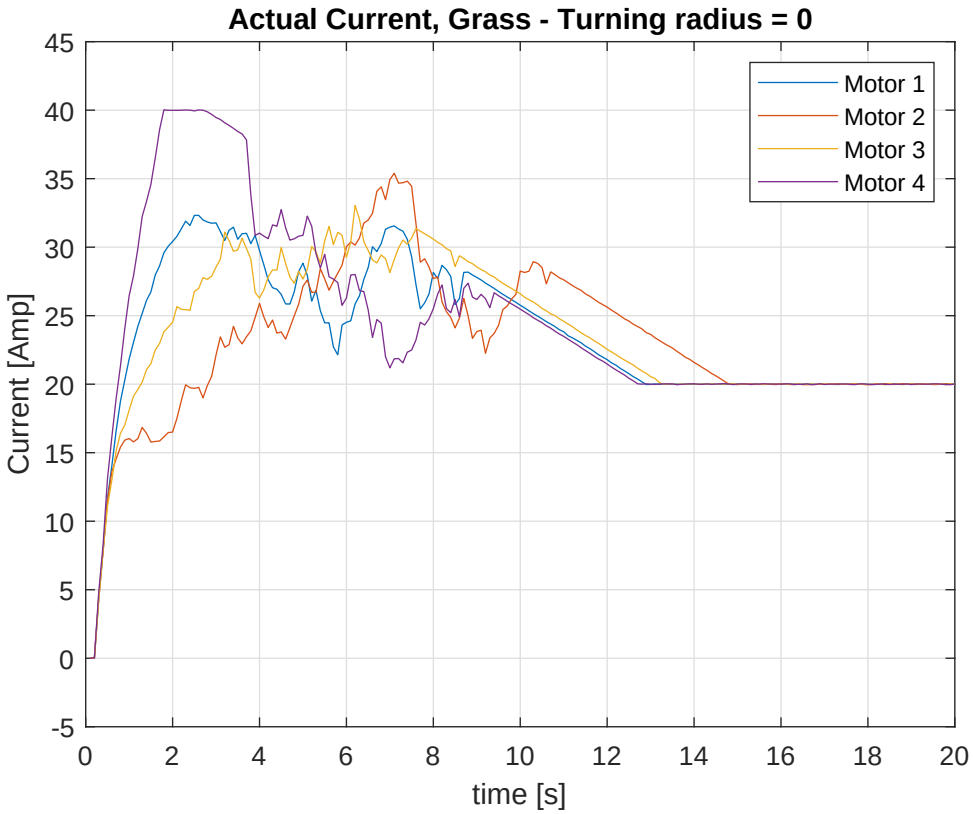


Figure 11.6: The figure shows current drawn from each motor during a test on an terrain with grass. The robot was programmed to turn around its own axes. The current grows above the continuous maximum level and the drives turn down the current to a maximum of 20 A. This is not enough to drive the wheels, and the robot is not capable of moving any further.

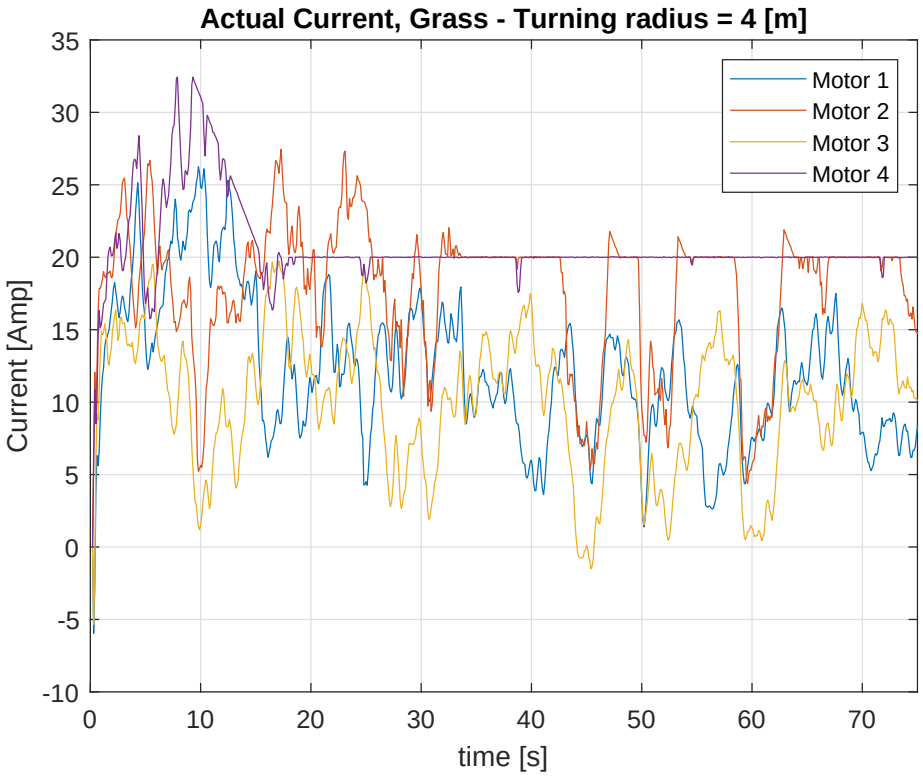


Figure 11.7: The figure shows current drawn from each motor during a test on a terrain with grass. The robot was programmed to drive in a circle with a radius of approx 4 meters. Occasionally the current of motor 2 and 4 grows above the continuous maximum level and the drives turn down the current to a maximum of 20 A. The result is a bigger turning radius.

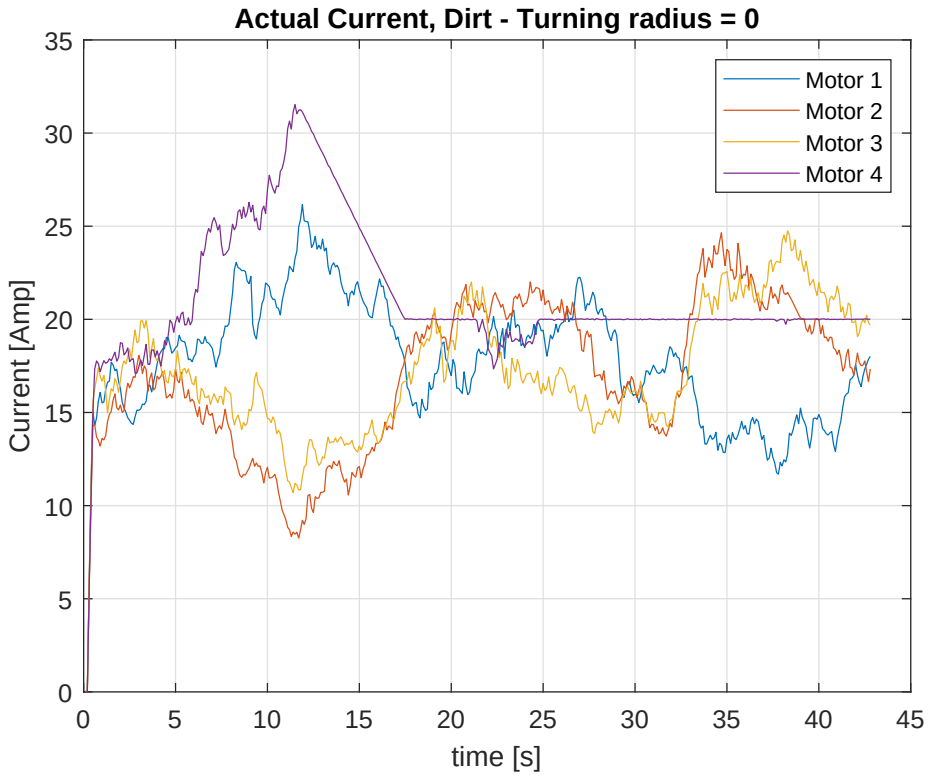


Figure 11.8: The figure shows current drawn from each motor during a test on a terrain of dirt. The robot was programmed to turn around its own axes. The current in motor 4 is restricted by the drive and turned down to a max of 20 A. This makes the robot slide a little and it does not stay exactly at the same spot as commanded.

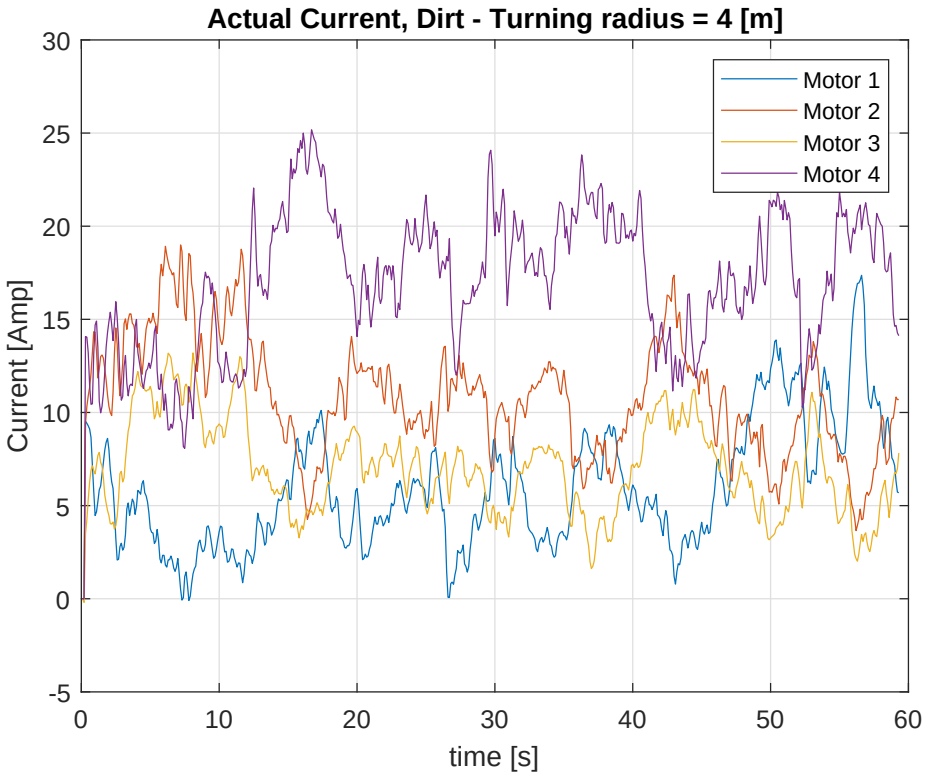


Figure 11.9: The figure shows current drawn from each motor during a test on a terrain of dirt. The robot was programmed to drive in a circle with a radius of approx. 4 meters. Non of the motors are restricted by the drives and the robot is able to perform the task.

11.4 Tuning weights

As mentioned in chapter 9, the weight matrices Q and R was estimated using Bryson's rule. To optimize the controller performance and gain some understanding of how the change in weights would affect the performance of the controller, some tests was performed where these weights was changed, while the robot performance was logged. To avoid the peak current problems, these tests was done in the laboratory on a hard even surface, where the robot was programmed to run straight for 1.5 m and then perform a 90 degrees left turn, followed by a straight line of 1 m.

Weight test 1

In the first test the weight on the states Q was unchanged and the weight on the input R was set one magnitude lower:

$$Q = \begin{bmatrix} 400 & 0 & 0 \\ 0 & 400 & 0 \\ 0 & 0 & 100 \end{bmatrix}, R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, K = \begin{bmatrix} 6.3 & 0 & 0 \\ 0 & 6.3 & 3.2 \end{bmatrix} \tag{11.1}$$

This test actually showed a very good performance. The robot follows the trajectory almost perfectly. See figure 11.10.

Weight test 2

In the next test R is set one magnitude higher than the original:

$$Q = \begin{bmatrix} 400 & 0 & 0 \\ 0 & 400 & 0 \\ 0 & 0 & 100 \end{bmatrix}, R = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}, K = \begin{bmatrix} 0.6 & 0 & 0 \\ 0 & 0.6 & 0.3 \end{bmatrix} \tag{11.2}$$

This produced weaker results, the robot turns too slow. See figure 11.11.

Weight test 3

Since test 1 showed good results, R was set back to this value. Now it was investigated if a changed relationship between the position errors and direction errors, in the state vector, would affect the performance:

$$Q = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 200 \end{bmatrix}, R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, K = \begin{bmatrix} 4.5 & 0 & 0 \\ 0 & 4.5 & 4.5 \end{bmatrix} \tag{11.3}$$

Also this result was bad. The robot seems more aggressive and turns too fast, without following the trajectory particularly well. See figure 11.12.

Weight test 4

In the last test the weight on the direction error are set 4 times higher than the weight on the position error:

$$Q = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 800 \end{bmatrix}, R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, K = \begin{bmatrix} 4.5 & 0 & 0 \\ 0 & 4.5 & 8.9 \end{bmatrix} \tag{11.4}$$

This is really bad. After the turn the robot becomes unstable and is not able to come to a steady state. See figure 11.13. The overall conclusion to these tests is that a low weight on R and a relationship in Q, where the weight on the direction error are kept lower than the weights on the position errors, seems to produce the best results. It must be noted that these tests were done on a solid and hard floor in the laboratory. There is a high possibility that these results will change under different terrains, being it grass or dirt. A deeper testing, could perhaps even reveal the need for some kind of gain scheduling, related to different terrains.

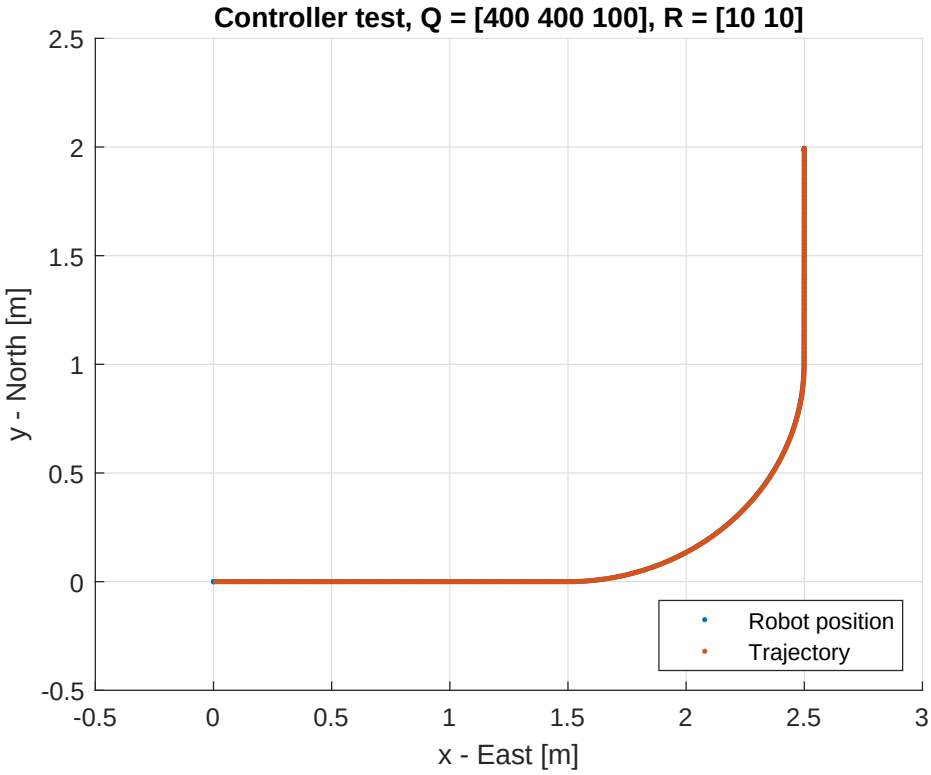


Figure 11.10: An almost perfect result. The robot follows the trajectory with very little deviation.

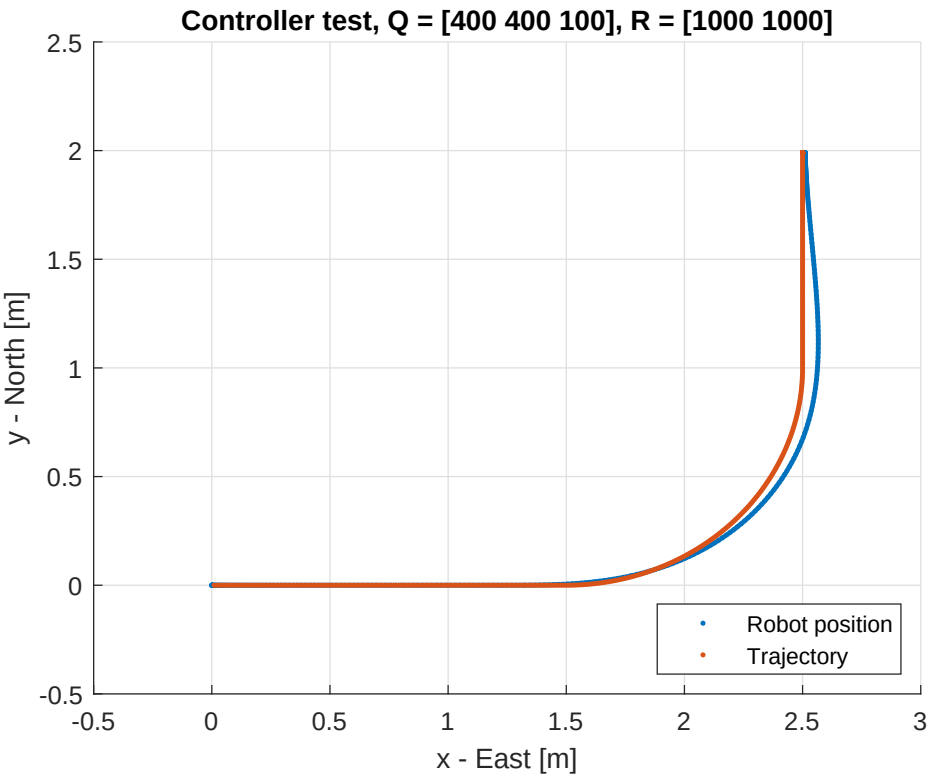


Figure 11.11: In this test the controller reacts too slow on the direction change.

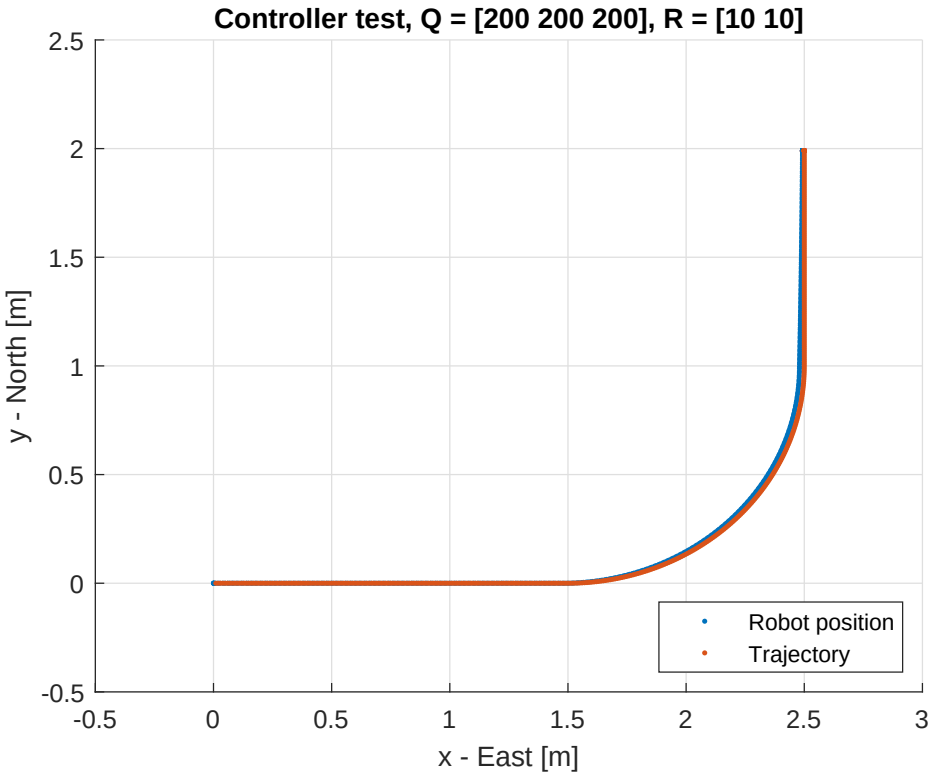


Figure 11.12: In this test the controller reacts too aggressive to change in the direction and does not follow the trajectory very well.

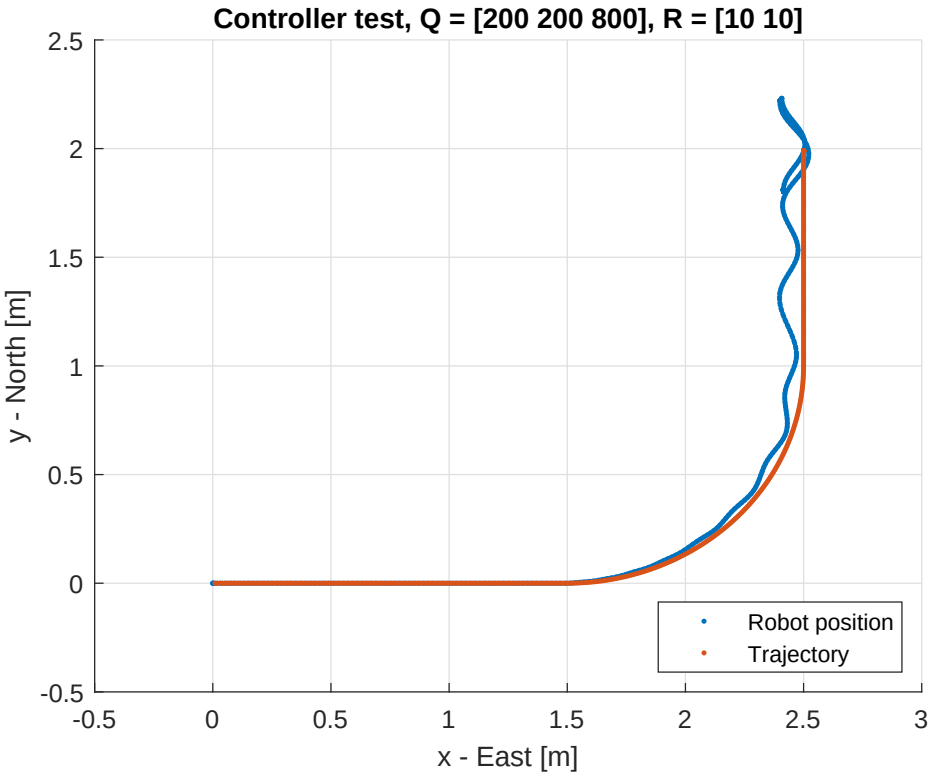


Figure 11.13: A very bad result. After the turn, the robot becomes unstable and are not able to come back to the reference heading and follow the trajectory.

11.5 Estimating forces

To determine the coefficients k_i and k_p, a test was performed, where the robot drives with different velocity's, while the current use is logged. As seen in table 11.1, the mean current of all the motors (denoted μ_{all}) increases with an increasing velocity. The values are logged after the acceleration has finished and the robot has reached a steady velocity. Although the data have a high variance, it is possible to use linear regression and fit an affine function to the measured values, as seen in figure 11.14. The function was estimated to:

$$I = 1.26v + 1.8 \tag{11.5}$$

This function can now be used to estimate k_i and k_p, however it must be emphasised that this estimation are done on a low amount of data. The outcome of this test can probably vary a lot under different circumstances, especially different terrain.

velocity $[m/s]$	0.01	0.05	0.1	0.2	0.3	0.4	0.6
$\mu_{motor1}[A]$	0.83	1.18	2.44	2.12	2.93	2.7	2.83
$\mu_{motor2}[A]$	1.83	2.21	2.32	1.51	2.68	2.47	2.71
$\mu_{motor3}[A]$	1.72	2.89	3.07	2.19	3.83	2.35	2.33
$\mu_{motor4}[A]$	1.15	1.09	1.40	1.60	1.74	1.21	1.50
$\mu_{all}[A]$	1.38	1.84	2.29	1.85	2.80	2.18	2.34

Table 11.1: This table shows the current drawn from the individual motors on under seven different velocity's. μ expresses the mean of the data. As seen motor 4 has a tendency to draw less current than the other three motors.

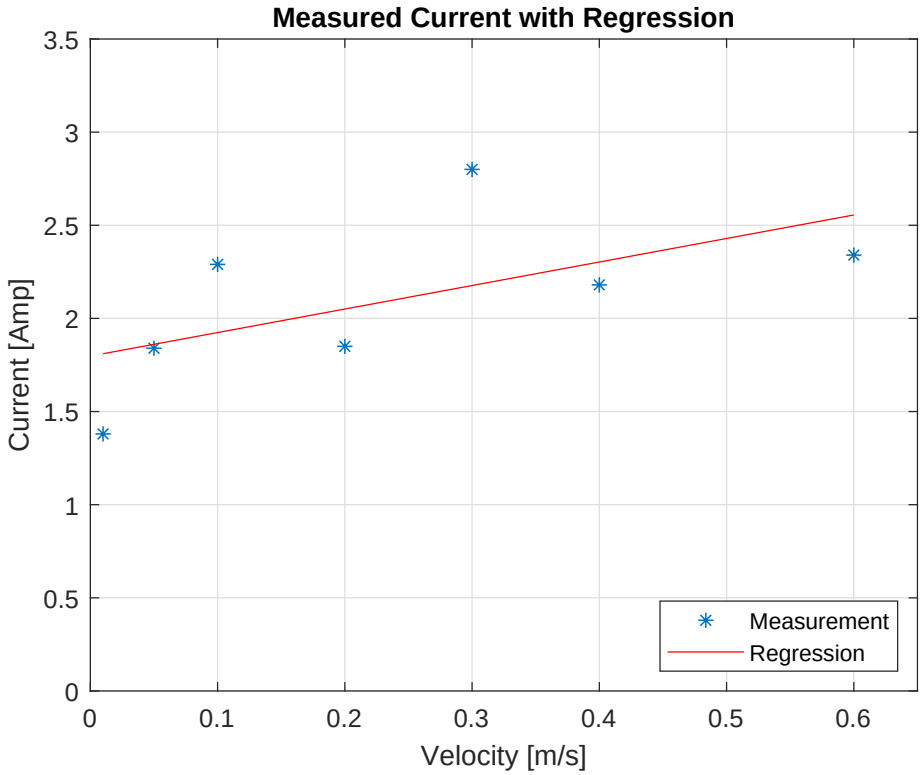


Figure 11.14: The blue stars shows current drawn during seven different steady velocity tests. By using linear regression, an affine function is estimated to fit the data (red line).

11.6 GPS test

Testing the variance of the GPS-signal was done by leaving the robot in the same location for an hour and logging the data. The result is seen in figure 11.15. As seen the signals drift over time and varies with a little more than 1 meter in both directions. A vector with difference values was calculated by, in each timestep,

calculating the difference between the current value and the previous value. The variance is then calculated as the mean of this vector. The result was a variance of $9.3 \cdot 10^{-4}$.

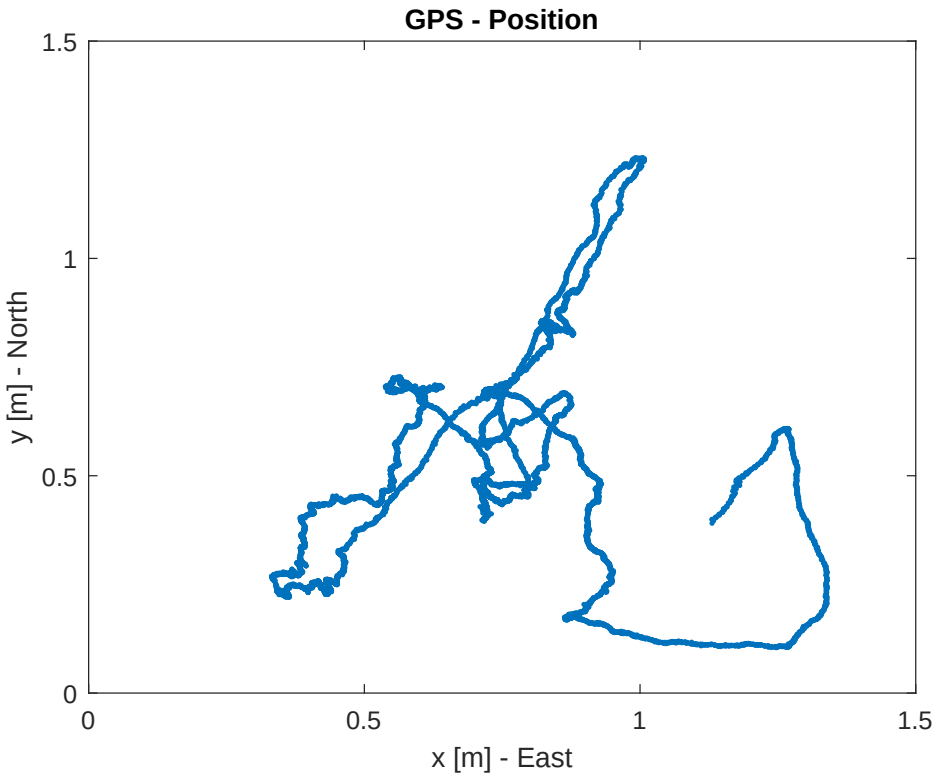


Figure 11.15: A static test of the GPS signal. The robot was standing in the same spot for an hour while logging the GPS-data. As seen the position slides within a little more than a m^2.

This marks the ending of the test chapter and the end of the development phase of the project as a whole. To round up the report a conclusion is needed, which will be followed by a discussion of some topics for future work on the robot. This will be done in chapter 12 and 13

12 Conclusion

This project set out to develop, model and control an agricultural robot. This lead to initial analyses of the problems at hand, regarding the use of heavy machinery in modern agriculture. Furthermore, some design thoughts were presented along with the introduction of the prototype, used for the project. This lead to the problem formulation, which asked if it was possible to model and control an agricultural robot prototype, based on the RobuROC4, that can run autonomously and follow a given trajectory.

After establishing the communication, building the model, analysing and filtering the sensor signals, defining trajectory's and designing the controller, the answer to this question is positive. It was possible to make the robot drive along a predefined trajectory. The project did not set out to give any requirements for how precise the trajectory must be followed, since this will very much depend on the application. However the tests showed that the controller can be tuned and adjusted in a way that will produce satisfactory results.

The development process also showed limitations and challenges regarding the design. As examples can be mentioned the uncertainties in the dynamic modelling and the problems with the current restrictions in the drives. These topics must be investigated deeper if a future development of the robot, shall reach higher levels.

Despite these limitations, the conclusion is that the robot could be controlled properly in the field and the author hereby claims the problem solved.

13 Discussion

This chapter will discuss some of the topics of the robot design, that have not been treated throughout the report, or needs deeper investigation. The chapter is not meant as a completely fulfilling list of problems at hand, but more as inspiration to what could be done in future iterations of development.

- The robot has a very rigid frame. This means that it sometimes in uneven environments can come in the situation where e.g only the right front wheel and the left back wheel caries the weight of the robot. This leads to a very bad traction on the two other wheels. As claimed in the beginning of the project, a tracked robot would have been a better solution. This would make the robot in better contact with the ground and also produce lower soil compaction.
- The motor drives are restricted to 20/40 amps. This is one of the very important design limitations on the robot. In future developments, this must be taken in to consideration when developing control algorithms. In an uneven field it will be very hard to model when these limitations will take effect. A more practical solution could be using stronger motors or a higher gearing, so that the robot always is capable of turning when commanded.
- During field tests, it was seen that sometimes one of the wheels were spinning freely, without having contact with the ground. This is due to the rigid frame of the robot. This has two major disadvantages, first the robot is of course not moving as desired, because of the slippage of the wheel. Secondly the odometry produces very poor results. This type of failure could perhaps be avoided by some kind of virtual alignment of the wheels on each side, making sure that both wheels on a side is spinning with the same angular velocity.
- The mathematical model of the robot was simplified quite a lot. Especially the friction coefficients are highly non-linear by nature. If the wheels dig deep in to the ground during turning the resistance will increase very much as the wheels have to move a lot of dirt when the robot is turning. The dynamic

model requires that the robot is situated on a plain solid surface, where the normal force on the wheels is evenly distributed en each wheel. In the real world, where the robot drives in the field this is far from the truth.

- Determining the parameters k_p and k_i was done by driving the robot and measuring the actual current used in each motor. This was only done in a few runs, which could be expanded a lot if more precise data is needed.

- Besides the bumpers there is not, at the given time, used any sensors on the robot to avoid collision with the surrounding environment. This means that the robot is not capable of detecting and avoiding any obstacles appearing. Since avoiding obstacles was not a research topic in this project these limitations did not give any problems. It can be argued that avoiding obstacle does not make much sense for agricultural robot. All obstacles in a field e.g trees and waterholes should be mapped beforehand and considered when planning the trajectory. If any unforeseen obstacles enters the field after start-up e.g. animals or humans, the robot should just stop and wait until the path is clear again.

- The global planner uses code-blocks that must be put together by the user, to generate a trajectory. In future developments this could be automated further. As an example, algorithms have been proposed that uses Dubin curves to solve this problem [15].

- The GPS system was not able to establish a connection to the RTK base station. If higher precision is desired this issue must be resolved.

- The robot prototype, used in the project, did not have any attached machinery. If that had been the case, an algorithm must be developed that raise/lower the machinery when the robot approches/leaves the headland. This could be done by adding an extra row of data to the trajectory, defining points where the headland starts/ends. The robot controller could then use these points as reference of when to raise/lower the machinery.

Bibliography

[1] Agrointelli. *Robotti*. http://www.agrointelli.com/robotti. 2019.

[2] Mathias N Andersen, Lars J Munkholm, and A Lisbeth Nielsen. "Soil compaction limits root development, radiation-use efficiency and yield of three winter wheat (Triticum aestivum L.) cultivars". In: *Acta Agriculturae Scandinavica, Section B–Soil and Plant Science* 63.5 (2013), pp. 409–419.

[3] Nollet Slove Andersen Madsen. "Autonomous Road Striper". In: (2012).

[4] CAN in Automation. *CAN in Automation*. https://www.can-cia.org. 2019.

[5] Waldemar Bangert et al. "Field-robot-based agriculture:"RemoteFarming. 1" and "BoniRob-Apps"". In: *VDI-Berichte* 2193.439-446 (2013), pp. 2–1.

[6] Bomford-Turner. *Flailbot*. https://www.bomford-turner.com. 2019.

[7] Ricardo Carona, A Pedro Aguiar, and Jose Gaspar. "Control of unicycle type robots tracking, path following and point stabilization". In: (2008).

[8] John Deere. *JD 9620*. 2019.

[9] engineersedge.com. *coeffients of friction*. https://www.engineersedge.com. 2019.

[10] Andrew English et al. "Vision based guidance for robot navigation in agriculture". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1693–1698.

[11] FARMDROID. www.farmdroid.dk. 2019.

[12] Fendt. *1050 Vario*. https://www.fendt.com/dk/7873.html. 2019.

[13] Mohinder S Grewal, Angus P Andrews, and RW Bass. "Kalman filtering: Theory and practice". In: *IEEE Transactions on Automatic Control* 40.11 (1995), p. 1983.

[14] Lise Gustavson. "Koersel paa jord og jordhaandtering". In: (2013).

[15] Karl D Hansen and Anders La Cour-Harbo. "Waypoint planning with Dubins curves using genetic algorithms". In: *2016 European Control Conference (ECC)*. IEEE. 2016, pp. 2240–2246.

[16] HP Harrison and WB Reed. "An analysis of draft, depth and speed of tillage equipment". In: *Canadian Agricultural Engineering* 4.1 (1962), pp. 20–23.

[17] Case IH. *Magnum 380*. `https://www.caseih.com`. 2019.

[18] Wajahat Kazmi et al. "Adaptive Surveying and Early Treatment of Crops with a Team of Autonomous Vehicles." In: *ECMR*. 2011, pp. 253–258.

[19] Mathieu Lamand and Per Schjoenning. "Trykkets forplantning i jorden: effekt af bl. a. daekstoerrelse og hjullast". In: *Trykkets forplantning i jorden: effekt af bl. a. daekstoerrelse og hjullast*. Aarhus Universitet, Det Jordbrugsvidenskabelige Fakultet. 2007, pp. 138–139.

[20] Naio. *Oz*. https://www.naio-technologies.com/en/. 2019.

[21] Henrik Schioler and Trung Dung Ngo. "Trophallaxis in robotic swarms-beyond energy autonomy". In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. IEEE. 2008, pp. 1526–1533.

[22] Per Schjoenning. "Strukturskader-hvordan og hvorfor er det vigtigt?: Kan vi selv klare det, eller skal vi vente paa EUs Jordrammedirektiv?" In: *Plantekongres 2008*. Aarhus Universitet, Det Jordbrugsvidenskabelige Fakultet. 2008, pp. 296–298.

[23] BS Shivaprasad, MN Ravishankara, and BN Shoba. "Design and implementation of seeding and fertilizing agriculture robot". In: *International Journal of Application or Innovation in Engineering and Management (IJAIEM)* 3.6 (2014), pp. 251–255.

A Appendix

A.1 Quick start guide to DriveWare

This appendix is meant as a quickstart guide, that describes how to connect to the drives and use the software DriveWare for configuration. DriveWare can be downloaded from `https://www.a-m-c.com/downloads/`.

A thorough manual for DriveWare is also found on the webpage.

Figure A.1: When using DriweWare, it is necessary to connect to the auxiliary port on the drive. Use a USB-RS232 cable with a Phoenix 1881338 plug mounted. Remember to turn on the drive!

After starting DriveWare, a pop-up box appears. Select **Open** to work on a project off-line or **Connect** to contact the drive. If everything works as it should the system should be visible in the system browser.

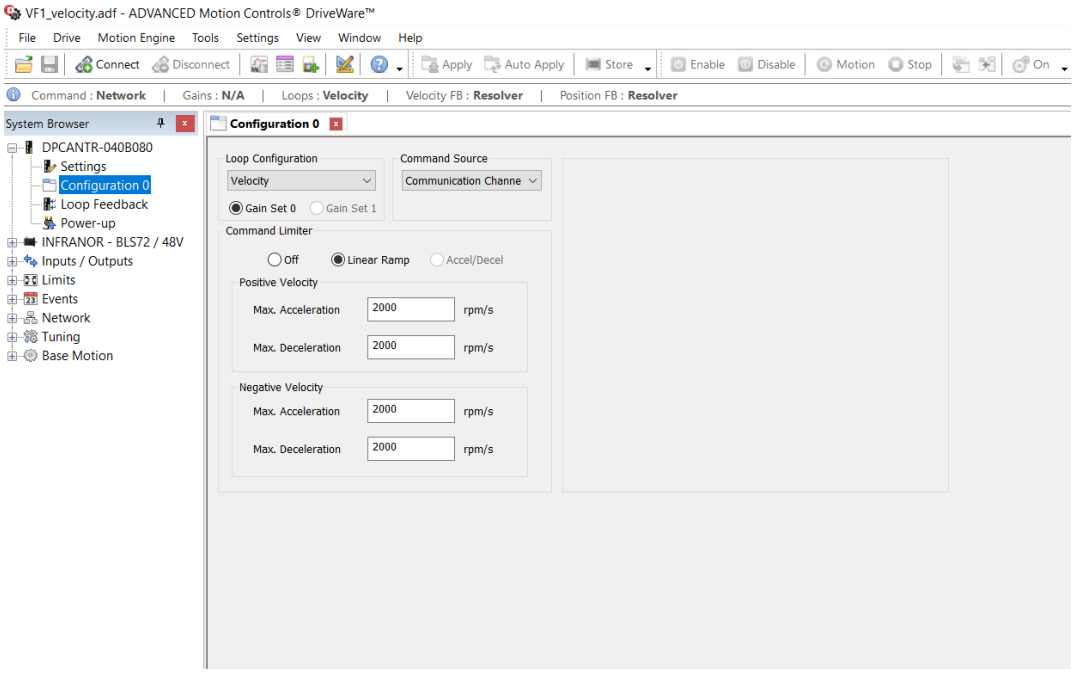


Figure A.2: In the Configuration tab, it is possible to define how the drive should operate. Chosse between position, velocity or current.

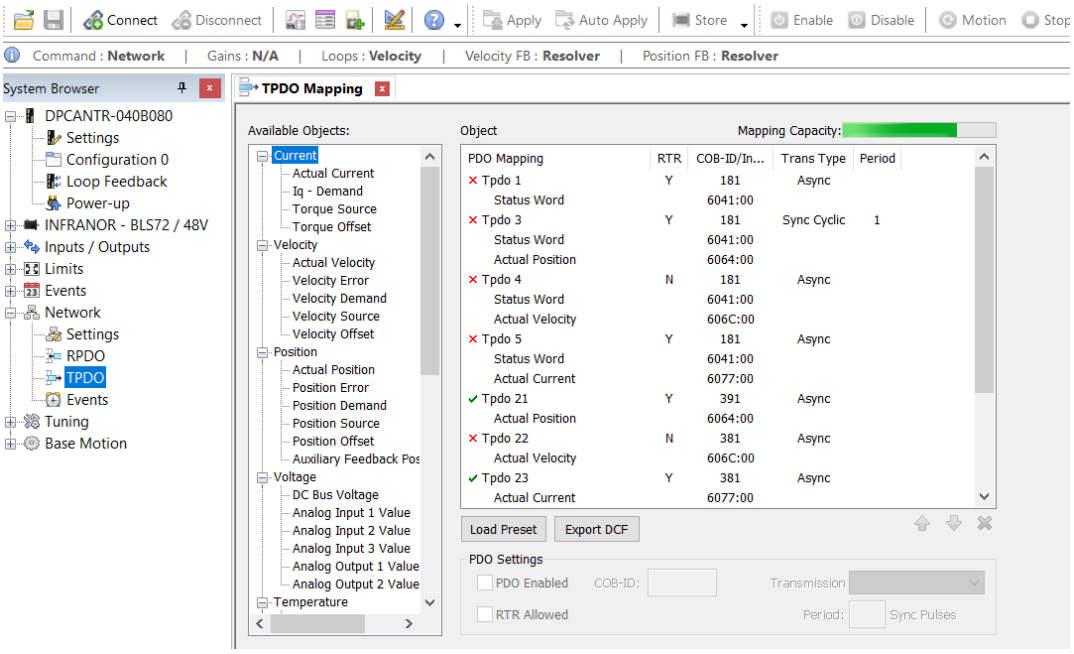


Figure A.3: In the network settings the TPDO's are set. These can be set up to read the value of a specific object and then transmit them to the CAN-bus. Click on the different TPDO's to make them active, and it is now possible to enable the TPDO and allow RTR. In this specific case TPDO 21 and 23 are active and set up to handle information on actual position and actual current.

Like the Transmit Process Data Object (TPDO) shown in figure A.3, a Receive Process Data Object (RPDO) must be set up to receive the position, velocity or current commands from the CAN-network. This is done in the same way as for the TPDO.

Be aware that there is three different ways of saving the configuration.

1. **Save** - Saves the work in a file on the pc
2. **Apply** - Lets the drive work with the changes while it still is connected to the pc.
3. **Store** - Saves the work on the drive, also after disconnecting the drive.

It is therefore very important to press the **Store** button before disconnecting, if the work should be saved on the drive.

After configuration, press **Disconnect**. The drive is now ready to run with the new configuration.

B Appendix

B.1 Wheels vs. Tracks

During the introduction it was claimed that an agricultural robot should be equipped with tracks. The testing of the wheeled robot in this project, shoved that the robot uses a lot of current when turning. Based on these thoughts it could be interesting to calculate the force needed to turn a skid steered wheeled robot compared to a robot equipped with tracks.

Wheels

On a skid steered vehicle, the weight is distributed on four wheels. When the vehicle turns the wheels must slide according to the angle the wheels are mounted compared to a central line through the vehicle, denoted L_c (see figure B.1). Let ϕ_1 denote the angle of the wheel according to that central line. The friction force for the robot with wheels, can now be calculated as:

$$F_w = \sin(\phi_1) F_n \mu_d \tag{B.1}$$

where F_n was the normal force and μ_d was a dry friction coefficient. In case of the robot, where ϕ_1 was calculated to 0.71 rad, the summed friction force of the robot with wheels is:

$$F_w = \sin(0.71) \cdot 1375 \cdot 0.36 = 322[N] \tag{B.2}$$

Tracks

If the robot instead of wheels, was mounted with tracks, the weight would be distributed on all the area of the tracks. Some parts of the tracks that are lined up with the central line does not have to slide, since they are perpendicular to the central line. (See figure B.2). We can, as an example, use five points along the tracks, denoted $t_1 - t_5$ to approximate the weight distribution. Let t_1 and t_5 be calculated with respect to the angle $\phi 1$. These two points are the same as the ones in the

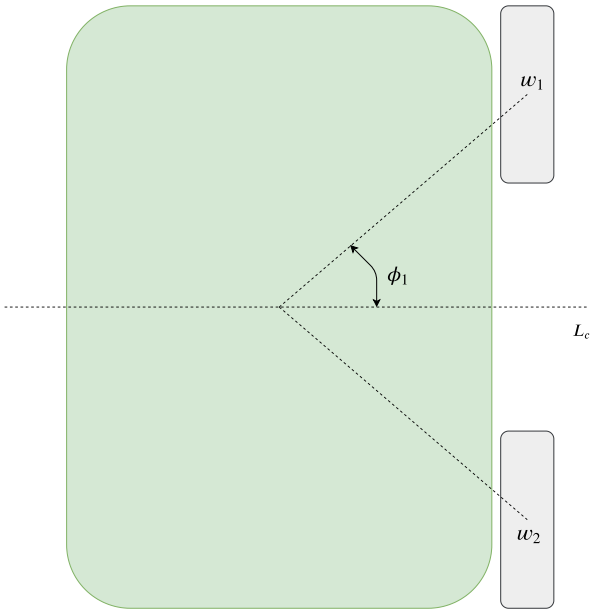


Figure B.1: The weight on the wheeled robot is distributed on 4 wheels, of which two is shown on the figure. All wheels are situated at the same angle with respect to the central line L_c.

wheeled example. t_2 and t_4 are then calculated with respect to ϕ_2. In the point t_3, no sliding occurs.

In this example ϕ_1 was calculated to 0.71 and ϕ_2 was calculated to 0.41. The total slide force for the robot with tracks can therefore be calculated as:

$$F_t = (\sin(\phi_1) + \sin(\phi_2))\frac{2}{5}F_n * \mu_d \tag{B.3}$$

$$= (\sin(0.71) + \sin(0.41)) \cdot \frac{2}{5}1375 \cdot 0.36 = 207[N] \tag{B.4}$$

As seen the tracked version only needs approx 65 % of the force needed in the wheeled version. A conclusion to these calculations is that a tracked solution produce less friction force, when the robot is turning.

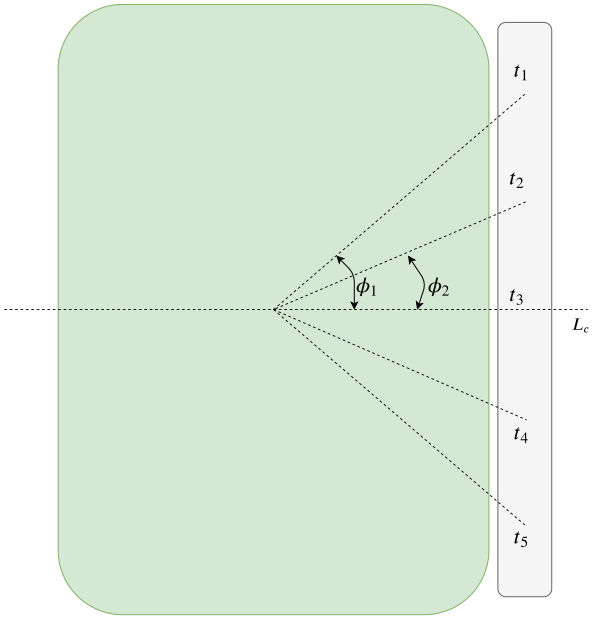


Figure B.2: On a tracked robot, the weight is distributed on two tracks, of which one is shown on the figure. If we approximate the tracks as a collection of five points (t_1-t_5) These points are situated with an angle according to the central line L_c. In the point t_3, the tracks does not experience slipping during turning since they are situated perpendicular to L_c.

C Appendix

C.1 Code

C.1.1 Setup CAN channel

```
ch1 = canChannel('PEAK-System','PCAN_USBBUS1');
configBusSpeed(ch1,1000000);

%Start the channel
start(ch1);
```

C.1.2 Reset drives

```
function resetDrives(ch1)

msg17 = canMessage(0,false,2);
msg17.Data = ([129 1]);
msg18 = canMessage(0,false,2);
msg18.Data = ([129 2]);
msg19 = canMessage(0,false,2);
msg19.Data = ([129 3]);
msg20 = canMessage(0,false,2);
msg20.Data = ([129 4]);

try
transmit(ch1,[msg17,msg18,msg19,msg20]);
catch
transmit(ch1,[msg17,msg18,msg19,msg20]);
end

pause(1);

msg5 = canMessage(0,false,2);
```

```
msg5.Data = ([1 1]);
msg6 = canMessage(0,false,2);
msg6.Data = ([1 2]);
msg7 = canMessage(0,false,2);
msg7.Data = ([1 3]);
msg8 = canMessage(0,false,2);
msg8.Data = ([1 4]);
try
transmit(ch1,[msg5,msg6,msg7,msg8]);
catch
transmit(ch1,[msg5,msg6,msg7,msg8]);
end

pause(0.25);
end
```

C.1.3 Setup periodic messages

```
%Setup_periodic_messages

% RTR - live signal drives
msg13 = canMessage(1793,false,8);
msg13.Remote = true;
msg14 = canMessage(1794,false,8);
msg14.Remote = true;
msg15 = canMessage(1795,false,8);
msg15.Remote = true;
msg16 = canMessage(1796,false,8);
msg16.Remote = true;

% RTR - actual current
msg1_V = canMessage(897,false,8);
msg1_V.Remote = true;
msg2_V = canMessage(898,false,8);
msg2_V.Remote = true;
msg3_V = canMessage(899,false,8);
msg3_V.Remote = true;
msg4_V = canMessage(900,false,8);
msg4_V.Remote = true;

% RTR - actual position
msg1_P = canMessage(913,false,8);
```

```
msg1_P.Remote = true;
msg2_P = canMessage(914,false,8);
msg2_P.Remote = true;
msg3_P = canMessage(915,false,8);
msg3_P.Remote = true;
msg4_P = canMessage(916,false,8);
msg4_P.Remote = true;

% Velocity commands
msg1Vcom = canMessage(769,false,6); %Velocity command drive 1
msg1Vcom.Data = ([15 0 0 0 0 0]);
msg2Vcom = canMessage(770,false,6); %Velocity command drive 2
msg2Vcom.Data = ([15 0 0 0 0 0]);
msg3Vcom = canMessage(771,false,6); %Velocity command drive 3
msg3Vcom.Data = ([15 0 0 0 0 0]);
msg4Vcom = canMessage(772,false,6); %Velocity command drive 4
msg4Vcom.Data = ([15 0 0 0 0 0]);
```

C.1.4 Serial setup

```
serial_connection = serial('COM9','BaudRate',115200);
fopen(serial_connection)
serial_con_mag = serial('COM6','BaudRate',19200);
fopen(serial_con_mag)
```

C.1.5 Trajectory - First line

```
%First_line is only defined in forward direction
p_start = [0;0]; % A-punkt
p_slut = [1.5;0]; % B-punkt

trajectory = 0;
j=1;
A = p_slut-p_start;
A_length = sqrt(A(1)^2 + A(2)^2);

A_unit = A/A_length; %unit vector that point to p_slut
nn = (A_length * 100) + 1;
B = [1;0]; % x-axis unit vector in global frame
V = acos(A_unit' * B);
if p_slut(2) < 0
V = -V ;
end
```

```
trajectory(1,j) = p_start(1);        % x-position
trajectory(2,j) = p_start(2);    % y-position
trajectory(3,j) = V;    % heading

for j=2:nn
trajectory(1,j) = trajectory(1,j-1) + 0.01*A_unit(1);% x-pos
trajectory(2,j) = trajectory(2,j-1) + 0.01*A_unit(2);% y-pos
trajectory(3,j) = V;    % heading x
end
n = length(trajectory);
x_pos = trajectory(1,end);
y_pos = trajectory(2,end);

Right_turn_forward
line_length = 0.5;
Line_segment_forward
Left_turn_forward
```

C.1.6 Trajectory - Right turn forward

```
turn_radius = 1;
R_neg = [0 1;-1 0];

cirkel_centrum_vektor = R_neg*[cos(V);sin(V)];
cirkel_centrum = [x_pos;y_pos] + cirkel_centrum_vektor;

kvart_omkreds = turn_radius * 2 * pi/4;
c1 = kvart_omkreds * 100;
points = round(c1);
increment = kvart_omkreds/points;

for j=1:points
trajectory(1,n+j)=cirkel_centrum(1)+cos(V-j*increment+pi/2);
trajectory(2,n+j)=cirkel_centrum(2)+sin(V-j*increment + pi/2);
trajectory(3,n+j)=V - j*increment;
end

n = length(trajectory);
V = trajectory(3,end);
x_pos = trajectory(1,end);
y_pos = trajectory(2,end);
```

C.1.7 Trajectory - Line segment forward

```
for j=1:line_length*100
trajectory(1,n+j) = trajectory(1,j+n-1) + 0.01*cos(V);
trajectory(2,n+j) = trajectory(2,j+n-1) + 0.01*sin(V);
trajectory(3,n+j) = V;    % heading
end

n = length(trajectory);
V = trajectory(3,end);
x_pos = trajectory(1,end);
y_pos = trajectory(2,end);
```

C.1.8 Receive messages

```
%Recieve_messages
modtaget = receive(ch1,Inf); %Recieve messages
msgIn1 = extractRecent(modtaget,897,false);
msgIn2 = extractRecent(modtaget,898,false);
msgIn3 = extractRecent(modtaget,899,false);
msgIn4 = extractRecent(modtaget,900,false);
msgIn5 = extractRecent(modtaget,913,false);
msgIn6 = extractRecent(modtaget,914,false);
msgIn7 = extractRecent(modtaget,915,false);
msgIn8 = extractRecent(modtaget,916,false);

% Extract current
if msgIn1 ~= 0;
d1 = unpack(msgIn1,0,16,'LittleEndian','int16');
end
if msgIn2 ~= 0;
d2 = -unpack(msgIn2,0,16,'LittleEndian','int16');
end
if msgIn3 ~= 0;
d3 = unpack(msgIn3,0,16,'LittleEndian','int16');
end
if msgIn4 ~= 0;
d4 = -unpack(msgIn4,0,16,'LittleEndian','int16');
end

% Extract position
if msgIn5 ~= 0;
```

```
d5 = unpack(msgIn5,0,32,'LittleEndian','int32');
end
if msgIn6 ~= 0;
d6 = -unpack(msgIn6,0,32,'LittleEndian','int32');
end
if msgIn7 ~= 0;
d7 = unpack(msgIn7,0,32,'LittleEndian','int32');
end
if msgIn8 ~= 0;
d8 = -unpack(msgIn8,0,32,'LittleEndian','int32');
end

value1 = double(d1); %Change int to double
value2 = double(d2);
value3 = double(d3);
value4 = double(d4);
value5 = double(d5);
value6 = double(d6);
value7 = double(d7);
value8 = double(d8);
```

C.1.9 GPS sensor

```
gps_data = fscanf(serial_connection); % Hent GPS data

lgd = length(gps_data);
if (lgd==115||lgd==116) &&...
strcmp(extractBetween(gps_data,1,10),'$PASHR,POS');
komma_pos  = regexp(gps_data,',');
latd = extractBetween(gps_data,komma_pos(5)+1,komma_pos(5)+2);
latm = extractBetween(gps_data,komma_pos(5)+3,komma_pos(6)-1);
lond = extractBetween(gps_data,komma_pos(7)+1,komma_pos(7)+3);
lonm = extractBetween(gps_data,komma_pos(7)+4,komma_pos(8)-1);
GP_data=extractBetween(gps_data,komma_pos(11)+1,komma_pos(12)-1);

gps_data = 0;

latd = str2double(latd);
latm = str2double(latm);
lond = str2double(lond);
lonm = str2double(lonm);
GP_head = str2double(GP_data);
```

```
if GP_head ~= 0;
GP_head=atan2(sin(-GP_head*2*pi/360+pi/2),cos(-GP_head*2*pi/360+pi/2))
end

lat_dm = [latd latm];     %Latitude [d:m]
lon_dm = [lond lonm];

lat_d = dm2degrees(lat_dm); %Convert from degree:minute to degree
lon_d = dm2degrees(lon_dm);

lat0 = 57.253306;
lon0 = 10.051168;

[GPx, GPy] = cassini_fwd(lat0, lon0, lat_d, lon_d)

GPx = GPx - 0.64*cos(2.5621 + head);% Offset from GPS antenna
GPy = GPy - 0.64*sin(2.5621 + head);

else
disp('Sorry, no satellite fix')

end

flushinput(serial_connection)
```

C.1.10 Magnetometer

```
mag_collect = fgetl(serial_con_mag);

if length(mag_collect) > 7 &&...
strcmp(extractBetween(mag_collect,1,5),'$OHPR') ;

komma_pos  = regexp(mag_collect,',');
subtrct_val=extractBetween(mag_collect,komma_pos(1)+1,komma_pos(2)-1);
Mx = str2double(subtrct_val);
subtrct_val=extractBetween(mag_collect,komma_pos(2)+1,komma_pos(3)-1);
My = str2double(subtrct_val);

%Calibrate magnetometer
Mx = Mx - ((max_x + min_x)/2);
Mx = 2*Mx/(max_x - min_x);
```

```
My = My - ((max_y + min_y)/2);
My = 2*My/(max_y - min_y);

%Calculate heading
mag_head = -atan2(My,Mx);negated for positiv heading

%Subtracting pi/2 for heading 0 = East
mag_head = atan2(sin(mag_head - pi/2),cos(mag_head - pi/2))

end
flushinput(serial_con_mag);
```

C.1.11 Odometry

```
if i==1
odo_sync1 = value5; %Startvalue from drives
odo_sync2 = value6;
odo_sync3 = value7;
odo_sync4 = value8;
end

odom(1) = ((value5 - odo_sync1)/TICKSPERREVOL)* CIRCUMFERENCE;
odom(2) = ((value6 - odo_sync2)/TICKSPERREVOL)* CIRCUMFERENCE;
odom(3) = ((value7 - odo_sync3)/TICKSPERREVOL)* CIRCUMFERENCE;
odom(4) = ((value8 - odo_sync4)/TICKSPERREVOL)* CIRCUMFERENCE;

odo_left = (odom(1) + odom(3))/2;
odo_right = (odom(2) + odom(4))/2;

odo_left_diff = odo_left - odo_left_old;
odo_right_diff = odo_right - odo_right_old;

slip_faktor = 0.54;

odo_theta_diff = slip_faktor...
*((odo_right_diff - odo_left_diff)/WheelTrack);
odo_theta = odo_theta + odo_theta_diff-o_corr;

Ox = cos(odo_theta);
Oy = sin(odo_theta);
odo_theta = atan2(Oy,Ox); %Heading
```

```
odo_x = odo_x - x_corr +((odo_left_diff...
+ odo_right_diff)/2)*cos(odo_theta);
odo_y = odo_y - y_corr +((odo_left_diff...
+ odo_right_diff)/2)*sin(odo_theta);

odo_left_old = odo_left;
odo_right_old = odo_right;
```

C.1.12 Main loop

```
while run == 1
Recieve_messages
calc_current
Do_GPS
Do_magnetometer
Calc_odom
Do_Kalman
%===== Regulator ==================
p_ref=[trajectory(1,traj_point);trajectory(2,traj_point)];
d_ref=[cos(trajectory(3,traj_point));sin(trajectory(3,traj_point))];

pos = [Xest(1,2);Xest(2,2)];
dir = [cos(Xest(3,2));sin(Xest(3,2))];

p_ex = (pos-p_ref)'*d_ref;
p_ey = (pos-p_ref)'*R_pos*d_ref;

d_ex = (dir-d_ref)'*d_ref;
d_ey = (dir-d_ref)'*R_pos*d_ref;

K1 = K(1,1); %Gain  positionsfejl i x-retning
K2 = K(2,2); %Gain  positionsfejl i y-retning
K3 = K(2,3); %Gain  heading fejl i y-retning

v = -K1*p_ex;
omega = -K2*p_ey - K3*d_ey;

traj_point = traj_point+1;

if (traj_point > n-1) %Er vi ved enden af trajektoriet?
traj_point = traj_point-1;
disp('Trajektoriet stopper her')
```

```
end

v_l = (2*v - omega*WheelTrack)/diam;
v_r = (2*v + omega*WheelTrack)/diam;

data_l = v_l * 2048888; %Scale factor to the drives
data_r = v_r * 2048888;

pack(msg1Vcom,int32(data_l),16,32,'LittleEndian');
pack(msg2Vcom,int32(-data_r),16,32,'LittleEndian');
pack(msg3Vcom,int32(data_l),16,32,'LittleEndian');
pack(msg4Vcom,int32(-data_r),16,32,'LittleEndian');

waitfor(loop_rate);
i = i+1;
end
```