Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Plzeň, 2024

Filip Duda

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Filip DUDA**
Osobní číslo:	**A21N0111P**
Studijní program:	**N3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Téma práce:	**Návrh řídicího systému mobilního robotu použitelného v oblasti precizního zemědělství**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1) Seznámení s problémem obsluhy skleníku s využitím mobilní robotické platformy.
2) Návrh řídicího systému mobilní robotické platformy pro obsluhu skleníku.
3) Praktické ověření navrženého řídicího systému.

Rozsah diplomové práce: **40-50 stránek A4**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

Dodá vedoucí diplomové práce.

Vedoucí diplomové práce: **Ing. Miroslav Flídr, Ph.D.**
Katedra kybernetiky

Datum zadání diplomové práce: **2. října 2023**
Termín odevzdání diplomové práce: **20. května 2024**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Dr. Ing. Vlasta Radová
vedoucí katedry

V Plzni dne 2. října 2023

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 20.5.2024 .....

Vlastnoruční podpis

Declaration

I hereby submit this diploma thesis for review and defence, prepared at the end of my studies at the Faculty of Applied Sciences of the University of West Bohemia in Pilsen.

I declare that I have worked on this diploma thesis independently and exclusively using the specialized literature and sources, the complete list of which is included.

In Pilsen on 20.5.2024 .....

Signature

Poděkování

Rád bych poděkoval Ing. Miroslavu Flídrovi Ph.D. za vedení práce, cenné rady a trpělivost při konzultacích. Dále Ing. Tomáši Svobodovi ze společnosti Siemens za podporu během vývoje. Na závěr celému týmu Fravebot, zejména pak Ing. Martinu Juříčkovi a Ing. Vratislavu Benešovi.

Acknowledgement

I would like to thank Ing. Miroslav Flídr, Ph.D., for supervising the work, his valuable advice, and his patience during consultations. Additionally, I would like to thank Ing. Tomáš Svoboda from Siemens for his support during the development. Finally, I would like to thank the entire Fravebot team, especially Ing. Martin Juříček and Ing. Vratislav Beneš.

Abstrakt

Tato práce se zabývá vývojem řídících algoritmů pro mobilního zemědělského robota. Práce poskytuje přehled precizního zemědělství a jeho etap, následuje popis vývoje komunikačního systému mezi komponenty robota. Dále vysvětluje řízení motorů a inverzní kinematiku. Práce se dále zabývá vytvořením algoritmu generujícího řídicí sekvenci pro motory pomocí metody gradeint descent. Tento algoritmus je následně testován pomocí NX MCD. Kromě toho práce popisuje dvě techniky lokalizace: jedna je založena na měření a odhadu rychlostí kol a druhá na inerciální měřící jednotce.

Klíčová slova: zemědělství, robotika, řídící systémy, lokalizace

Abstract

This work deals with the development of control algorithms for a mobile agriculture robot. The thesis provides an overview of precision agriculture and its stages, followed by a description of the development of a communication system between the robot's components. After that, it explains the motor control and inverse kinematics. The work further involves the creation of an algorithm generating a control sequence for the motors using gradient descent. This algorithm is then tested using NX MCD. In addition, the thesis describes two techniques for localisation: one based on the measurement and estimation of wheel velocities, and the other based on an inertial measurement unit.

Keywords: agriculture, robotics, control systems, localisation

Contents

1 Introduction **8**

2 Description of Robot and Work Environment **11**
2.1 Fravebot Scout Robot . . . 11
2.1.1 Technical properties . . . 12
2.2 Robot work environment . . . 13

3 Communication architecture **16**
3.1 ROS . . . 16
3.2 MQTT . . . 17
3.3 IPC - PLC bridge . . . 17
3.4 TIA portal library . . . 19

4 Motor control **21**
4.1 Inverse velocity kinematics . . . 21
4.2 PLC Implementation . . . 24

5 Planing of control sequence using gradient descent **26**
5.1 Single step optimisation . . . 27
5.1.1 Simulation verification of the algorithm . . . 30
5.2 Receding horizon control . . . 31
5.3 Cost function adjustment . . . 34
5.4 Initialisation data . . . 37
5.5 Automatic increase of step count . . . 37
5.6 End of the rail . . . 37

6 Test of the AGV control system using software in the loop **40**
6.1 Implementation of the simulation loop . . . 41

7 Encoder-based Dead Reconing **44**
7.1 Feedforward positioning . . . 44
7.2 Left wheel encoder . . . 45
7.3 Right wheel velocity estimation . . . 46
7.3.1 Proportional estimation . . . 47
7.3.2 Drive model identification . . . 47

8 Rotation estimation using IMU **51**
8.1 Data acquisition . . . 51

9 Conclusion **58**

Acronyms

AGV Automated Guided Vehicle. 21–23, 26, 27, 29, 30, 32, 35–38, 40–45, 47, 58

GPU Graphics Processing Unit. 44

ICC Instantaneous Center of Curvature. 23

IMU Inertial measurement unit. 10, 13, 51–53, 57, 58

IPC Industrial PC. 13, 16, 17, 58

MCD Mechatronic concept desinger. 40–42

MQTT Message Queuing Telemetry Transport. 16–18, 58

MSE Mean Squared Error. 46

PLC Programmable Logic Controller. 13, 16–19, 21, 24, 40–42, 45, 58

ROS Robot Operating System. 16–19, 30, 41, 58

ROS2 Robot Operating System 2. 16–19, 30

RPM Revolutions per minute. 45

SIL Software-in-the-loop. 40, 41, 44

STL Standard Triangle Language. 41

TCP/IP Transmission Control Protocol/Internet Protocol. 41

TIA Totally Integrated Automation. 16, 19, 24, 41, 58

UAV Unmanned Aerial Vehicle. 8, 9

Chapter 1

Introduction

Agriculture has been the largest employment sector for a significant part of human history. According to Our World in Data [6] it employed more than half of the total workforce in European countries until the 17th to 19th century and in many countries it still represents a major sector. The beginning of reliance on agriculture started with the Neolithic Revolution when humans transitioned from hunting and gathering to farming. Since then every new agricultural invention that increased the efficiency of farming had a large impact on society. Innovations in agriculture changed food production and reshaped social, economic, and cultural aspects of human life. An example of early automation is the cotton gin, a very impactful machine. It was patented in 1794 by Eli Whitney [11] as a machine, that can pull cotton fibres from the seed. It was capable of replacing multiple workers as it could process more than 20kg of cotton per day, which was equivalent to 100 man-hours of work. While it replaced some workers, it also made cotton farming more profitable and ended up increasing the total number of workers in cotton farming. While the first machines required a human as a power source, the introduction of motors replaced the need for human work entirely. Even the modern ginning machines are very similar to the one made by Eli Whitney.

According to Fadhaeel [7], the earliest development of agricultural robots began in the 1920s with research on autonomous vehicle steering. The first self-driving agricultural vehicles emerged in the 1950s, however, these vehicles required a cabling system to work. Between 1950 and 1980, the number of patents in the field of agricultural robotics was relatively small under 30 per year. Between 1981 and 1990, the number of patent applications gradually increased to almost 70 per year. The development of positioning and navigation allowed the research of different types of agricultural robots, such as picking, milking, or harvesting. Papers on robotic shearing and milking were published in the UK and Australia. During the time between 1991 and 2005, the agricultural robotics further developed. Patents addressed mainly working conditions, growing techniques, and physical characteristics of the crops. In 1995, almost 200 patents were filled in this field globally. After 2006, the patent count increased sharply to hundreds per year. Modern sensors and actuators allow the production of intelligent, reliable, fast, and cost-effective robots. Institutions such as the European Union, U.S. Department of Agriculture, and Japanese Council for Science, Technology and Innovation [30] invest heavily in the research of agriculture innovation.

The spread of cheap and reliable unmanned aerial vehicles allows their use in agriculture. They are used in areas such as remote sensing, mapping, monitoring, and pest control. As it can take a lot of time to cover a large area, it can be beneficial to use multiple cooperating Unmanned Aerial Vehicle (UAV)s, however, this requires the development of a cooperation strategy. Examples of this can be RHEA Project [3], which developed a central management system for pest control in maize fields. Or research from Centre for Automation and Robotics in Madrid [1], that used three UAVs for mapping a vineyard.

While aerial vehicles are generally fast, their carrying capacity is limited and for many operations, ground vehicle is a more suitable option. Higher payload allows ground vehicles to transport required

materials or to be equipped with heavier effectors, such as robotic manipulators. As UGVs need to avoid obstacles, the path-planning can be more complicated, than with UAVs travelling directly. This will be a part of this thesis. Similarly, as with UAVs, there is often a need for multiple vehicles, which need to be coordinated. The strategy for coordination of multiple vehicles can be either centralised or decentralised. When these problems are solved with a sufficient level of safety and robustness, many labor-demanding tasks can be automated. These include planting, seeding, transplanting and harvesting [17], weed management [5], or even spraying a ploughing [12]. One of the fields, that specialise in increasing the efficiency of farming is so-called precision agriculture.

According to the International Society of Precision Agriculture [20], Precision Agriculture is defined as follows: "Precision Agriculture is a management strategy that gathers, processes and analyzes temporal, spatial and individual data and combines it with other information to support management decisions according to estimated variability for improved resource use efficiency, productivity, quality, profitability and sustainability of agricultural production." In other words, the main goal of precision agriculture is to conserve resources by applying them efficiently in the right place at the right time.

Research group at the University of Lleida [21] describes four main stages of precision agriculture:

1. First stage is data acquisition, in this stage data related to crops, soil, terrain or climate are acquired using different kinds of sensors.

2. The Second stage is extracting valuable information from the gathered data. This could be storing the data for later use, processing it into information or even using the location-based data to create maps.

3. The third stage involves making a decision based on the information gathered in the previous stages. This is usually done by an experienced specialist who uses the information to make the right decision. Although some aspects of decision-making, such as irrigating dehydrated plants, can be automated, the goal is not to eliminate human decision-making but to aid and support it.

4. The last stage is performing the decided action. This could be irrigation, application of fertilizer or herbicide, weeding or picking etc. This is also a very workforce-demanding stage, so a lot of systems aim to automate this part.

This thesis was created in cooperation with FraveBot company, which aims to automate stages 1, 2 and 4 and aid stage 3. FraveBot stands for Fruit and Vegetable Robot and is a startup based in Brno, Czech Republic. They focus on developing agricultural robots for monitoring, pruning and harvesting of the plants such as the one in Figure 1.1. According to their official webpage [28], the company uses deep learning and cameras on robotic arms to monitor fruit and vegetable plants. The artificial intelligence can evaluate flower pollination, fruit maturity, plant diseases and pests. Early intervention against pests reduces the need to use chemical treatments, making the process more ecological and contributing to the development of sustainable agriculture of tomorrow. The operation uses the LOTYLDA DL platform for data collection and analysis.

Figure 1.1: Fravebot strawberry picking robot [28]

The main goal of this thesis is to develop a control system for a mobile agriculture robot. At the beginning of the work on this thesis, the robot lacked high and low-level motion control. This thesis will describe the development of necessary software to move the robot to a desired destination efficiently. This should consist of the control of the motors as well as path planning, localisation of the robot and communication between all the mentioned parts. The robot was under development for the whole duration of this thesis and it is expected to continue afterwards.

After this introduction, the thesis will continue with a second chapter consisting of a description of the robot itself and the environment it will work in. The third chapter will cover the communication between hardware components of the robot. The next chapter will explain low-level control of the stepper motors and inverse kinematics. Chapter five illustrates the control strategy used for the movement in the environment. The sixth chapter will clarify the methods used for testing the developed algorithms in software simulation. Chapters seven and eight present two techniques used to obtain the position and rotation of the robot. Chapter seven describes localisation using real and virtual encoder. Chapter number eight explains rotation estimation using an Inertial measurement unit (IMU). The last chapter concludes the work and recapitulates the results.

The robot is under development and the information noted here may not reflect the current state of the robot.

Chapter 2

Description of Robot and Work Environment

This chapter will introduce the Fravebot Scout robot that will be used as the main development platform for the rest of the thesis. It will describe its functionality and technical properties and it will end with a brief overview of the environment it will work in.

2.1 Fravebot Scout Robot

Figure 2.1: Fravebot Scout robot

Fravebot Scout robot (Figure. 2.1) is a mobile robot designed to detect disease and pests as well as to predict the yield of plants. At the time of writing it could be adjusted for three types of plants, cucumbers, tomatoes and strawberries. The robot used in this work was configured for strawberries, however, for the purposes of this thesis, the robot's behaviour remains the same. The main difference in configurations is in the machine vision part. In the strawberry configuration, it can detect infestations

of angular leaf spots, anthracnose fruit rot, blossom and leaf blight or even insects such as spider mites or thrips. It can also determine the hydration or ripeness of the plants as can be seen in Figure 2.2.

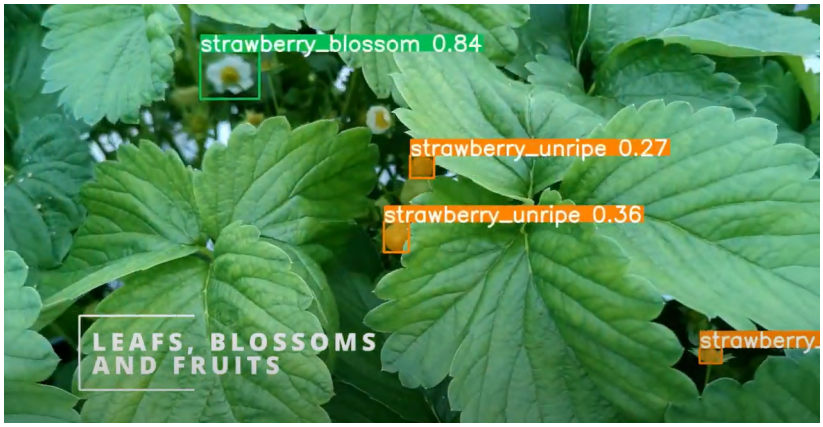
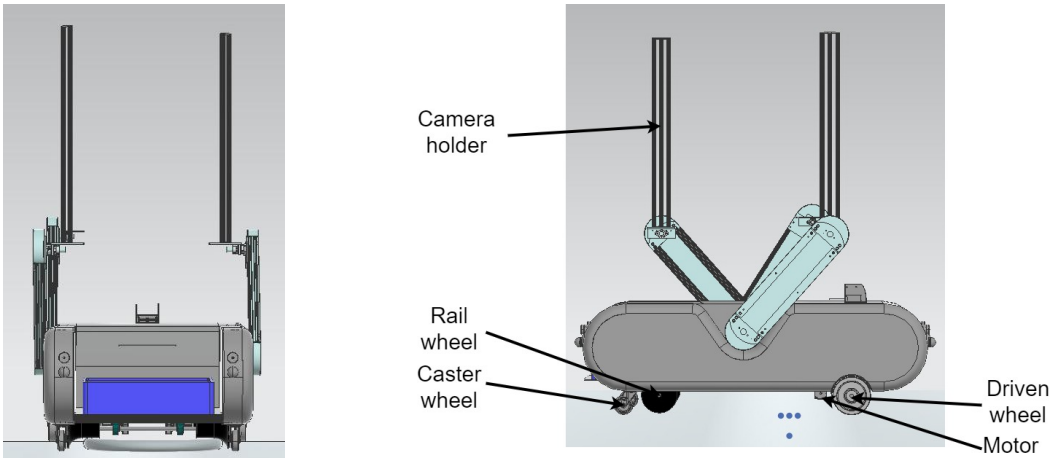


Figure 2.2: Classification example

2.1.1 Technical properties

The robot has four wheels, two driven and two passive caster wheels (visible in Figure 2.3). The back wheels are driven by two RAVEO stepper motors. Both wheels contain a 3:8 gearbox to increase the torque of the wheels as the robot weight is in hundreds of kilograms based on configuration. The left wheel includes an encoder for wheel velocity feedback. There are two sets of driven wheels, one for the transfer between rails, with a 10cm diameter and a conical rail wheel. The track width of the transfer wheels is 90cm. This drive system should allow the robot to scan 1ha of strawberries in 12h at a scanning speed of 0.35m/s and transport speed up to 2m/s. At the time of writing this thesis, the battery was replaced manually, but plans were to do it automatically in the future [28].



(a) Front view

(b) Side view

Figure 2.3: 3D model of the robot

In terms of hardware, we can decompose the robot into four main subsystems as presented in Diagram 2.4.

The first subsystem is the Industrial PC (IPC). Its task is to provide the high-level parts of the control system. The IPC used in the robot is Simatic IPC520A an industrial PC developed for AI-based applications. There are six main components on the IPC. The first part is the path planner, generating Twist control messages. Second is the encoder-to-wheel speed converter, which converts the pulses of the encoder into the angular velocity of the wheel. The third part is the velocity estimator of the wheel without the encoder. The fourth part is the differential drive kinematics, which takes the velocities of both wheels and estimates the robot's position. The fifth part is the IMU positioning which estimates the robot rotation based on data from a gyroscope. The last part is the communicator, which transfers important data between IPC and Programmable Logic Controller (PLC).

The next subsystem is the PLC, namely ET200 SP, which runs the inverse kinematics that converts the requested forward and angular velocity into the wheel velocity. The second part is responsible for communication with IPC. The last part implemented on PLC is the wheel drive controller, which is responsible for the low-level motor control.

The remaining two subsystems are the subsystems of the wheels, both of them contain a stepper motor and drive card controlling it, left wheel system includes an encoder to obtain feedback of the real velocity of the wheel.

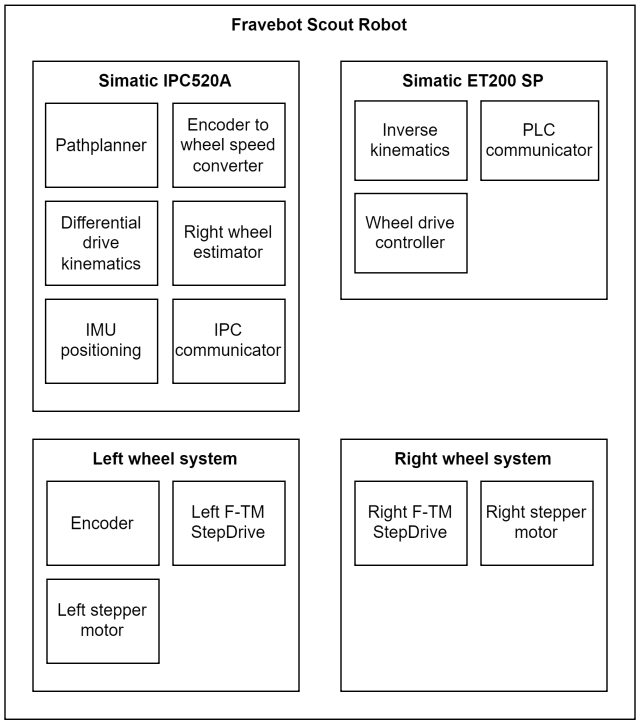


Figure 2.4: Function structure diagram

2.2 Robot work environment

The robot is designed to work in heated greenhouses. Many greenhouses use artificial heating systems to prolong the growing season. These systems can use different mediums for heat distribution, however, the one important for this work is the use of tubing filled with hot water as can be seen in Figure 2.5. This is useful as it can be used as a rail system for manual, motorised or even autonomous carts. The U-shaped tubing is laid on a concrete platform a few centimetres above the ground of the

greenhouse. The tubing continues in a horizontal position through the whole greenhouse supported by small pillars.

Figure 2.5: Photo of greenhouse heating system [9]

The concrete platform holds the endings of the tubing and acts as an intersection that trolleys can use to change their rail. It is clearly visible in Figure 2.6.

Figure 2.6: Greenhouse concrete platform

Unfortunately, the diameter of the tubes and the distance between them aren't standardised and can have different size combinations in different greenhouses. Many trolley manufacturers [8] deal with this issue by having changeable wheelbase and thus supporting multiple track widths.

In order to plan a path for the robot and control its movement, the control system needs to know the layout of the greenhouse. Thus, a suitable representation of the work environment was required. For this purpose, a map was created, which contains two main parts. First is the switching pad, the concrete platform, which is used to switch the rails, second is the list of rails. The switching pad is defined by the location of the top left corner, its width and height. In the next chapters, it will be represented by a blue rectangle as in Figure 2.7. Every rail is defined by its id, coordinates of the start

of the rail and coordinates of its end. It will be graphically represented by a grey line as in Figure 2.7.

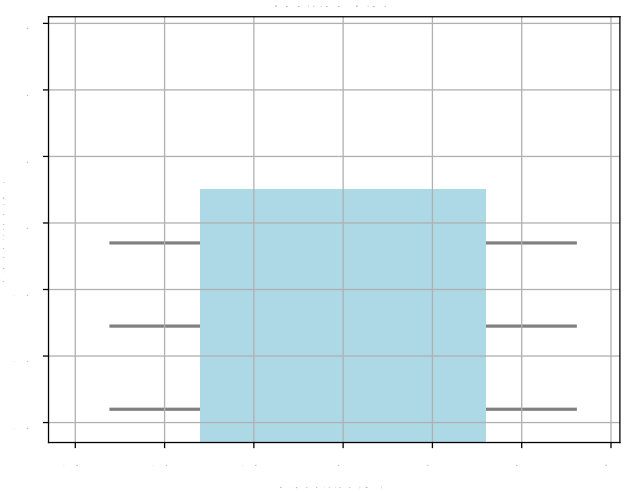


Figure 2.7: Simplified representation of greenhouse

In reality, the rails would be significantly longer, however, the movement along the rail is relatively simple and the motion we are interested in will occur on the concrete platform and at the beginning and end of the rails. For this reason, the size ratio between the platform and rails was changed to better show the functionality.

Chapter 3

Communication architecture

This chapter describes the communication architecture that enables precise movement of the robot. The two main hardware systems used are the IPC, which runs the path planning logic and PLC which is responsible for the motor control. In this chapter, we will focus on the communication between these two parts. The solution requested by Fravebot should support both Robot Operating System 2 (ROS2)[1] and Message Queuing Telemetry Transport (MQTT)[2] on the IPC side. Both of the mentioned communication protocols are widely used and will be further discussed later. There is other communication happening in the system, which can be seen in Figure 3.1 such as the one between ET200 PLC and the StepDrive cards. This communication as well as the generation of driving pulses for the motors is included in the standard F-TM StepDrive library and we don't need to study it in detail. In the same manner, the communication between MQTT or Robot Operating System (ROS) nodes is a standard implementation of basic subscribers and publishers. On the other hand, the PLC used doesn't currently support direct subscription of ROS or MQTT topics and it was necessary to develop a communication bridge. This chapter will focus on this bridge, which is highlighted by orange colour in the Diagram 3.1.

As this project is in development, it is preferable, that the communication is not limited to motor control, so that it is possible to add more communication topics in the future. The architecture consists of two main parts, the bridge translating from either ROS2 or MQTT topic into S7 protocol commands and the structure in the PLC memory, which organises the memory for the bridge to write in. This chapter will first generally describe the protocols that were used, and then it will look into the bridge running on IPC. The last part starts with a general description of the Totally Integrated Automation (TIA) portal and then describes the library that was developed.

3.1 ROS

The ROS, is an open-source framework for developing software for robotic devices. Since its inception in 2007, ROS has become a popular platform with a large number of software packages and support for numerous commercial robots. ROS2 is the current iteration of the Robot Operating System. It was introduced in 2015 with the first full version available in 2017. Even though it uses a different architecture, the use is very similar to ROS 1 [15]. ROS includes a set of drivers that enable reading data from sensors and sending commands to actuators in an abstracted format. In addition to a collection of control and mapping algorithms, ROS also provides infrastructure for data transfer, allowing for component integration within a robotic system and straightforward storage of transmitted data [23].

The communication is performed by so-called publishers and subscribers, which communicate by

[1] https://docs.ros.org/en/rolling/index.html
[2] https://mqtt.org/

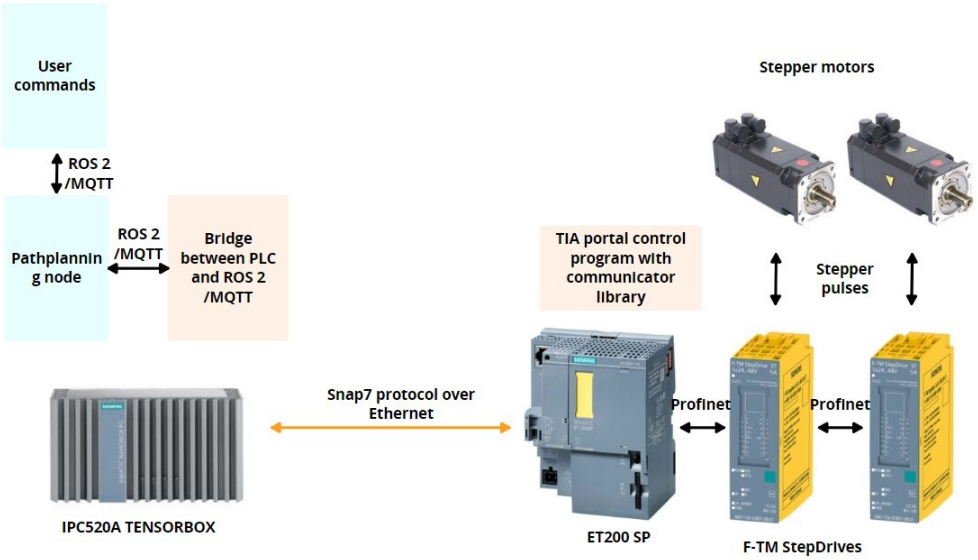


Figure 3.1: Communication between components of the architecture ; Source of pictures: Siemens industry mall [26]

means of so-called topics. Topics allow for the transmission of simple messages as well as more complex data structures between different components of the system.

The topic, that is essential for this thesis is the Twist topic (Figure 3.2). Twist datatype consists of 6 Floats, describing linear and angular velocity in every dimension.

3.2 MQTT

Fravebot has requested to add support for the MQTT protocol to the bridge. MQTT is a messaging protocol that was developed for the Internet of Things. It is designed to be lightweight, reliable and scalable and it uses a publisher-subscriber architecture that is very similar to ROS2 [16]. Just like ROS utilizes a ROS Master for managing communication between nodes, MQTT employs a broker to facilitate message exchange between publishers and subscribers. As a result, the additional support for MQTT did not require many changes in the bridge node and no changes in the PLC part however, since MQTT does not contain many of the functions of ROS2, some features needed to be additionally implemented. An example of this can be the rosbag package, which is a set of tools for recording and playing back ROS topics [24]. This package can be very useful for gathering data for analysis. As we needed to gather data when using MQTT, a tool for saving them needed to be developed. It is a Python node, that takes a list of topics, that need to be recorded and subscribes to all those topics. Afterwards, it creates a text file for each topic and saves all the incoming data with a corresponding time stamp. It can play back the recorded files in a similar way as rosbag. This script proved very useful for model identification mentioned in later chapters.

3.3 IPC - PLC bridge

As mentioned before, S7 PLCs don't support direct connection to ROS or MQTT, therefore a bridge is needed to connect the IPC and PLC. This bridge receives the data from either ROS2 or MQTT and writes it into the memory of PLC using Snap7 protocol. It also functions in the opposite way,

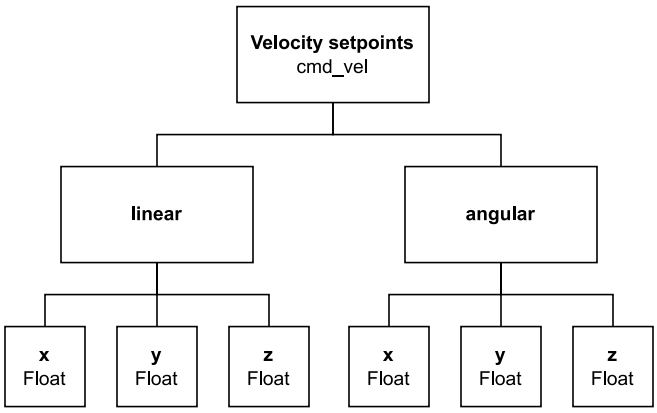


Figure 3.2: Twist datatype

First byte content	Datatype	Struct size [bytes]
00000000	Int16	38
00000001	String256	70
00000010	Float32	40
00000011	Twist	60

Table 3.1: Supported datatypes

it periodically reads from the PLC and publishes it to ROS2 or MQTT topics. The bridge uses two libraries for its functionality. For the receiving part, there is the rclpy library, which provides Python API for communication with ROS2 as well as other useful functionality such as asynchronous timers. The second library used is the snap7 library, which allows us to write data directly to PLC memory. According to official documentation [27] Snap7 is an open-source Ethernet communication suite for interfacing Siemens PLCs. The platform is independent and scalable and it supports both 32 and 64-bit architecture. The library can be used with Pascal, C++, C#, Python, Node.js, Java or ARDUINO. It was used to write the data into prepared data blocks. The algorithm requires 3 arguments to start: the IP address of the PLC and the IDs of the subscriber (Figure 3.3) and publisher (Figure 3.4) data block. The algorithm starts by connecting to the PLC at the selected IP and starts scanning for all subscriber and publisher topics. Since the data blocks are read byte by byte, it's necessary to determine the datatype of each subscriber or publisher. The first byte always contains information about the datatype, which allows us to determine the addresses of the rest of the structure. The list of supported datatypes is visible in Table 3.1. It includes a ROS Twist datatype which will be used as a setpoint for the movement of the AGV.

It would be possible to implement more datatypes, however, the mentioned ones were sufficient for the use case. Next, the algorithm sets up a subscriber or publisher according to the topic and datatype, so that the correct function can be used to transform the message from byte array to ROS2 message. When it receives a message from ROS2, it writes the data to the corresponding memory. The publisher's object includes information about the requested period. The bridge reads the memory at the requested rate and publishes the data to ROS2.

3.4 TIA portal library

TIA Portal is an engineering software platform developed by Siemens. It is used for configuring, programming, testing, and diagnosing the Basic, Advanced and Distributed Controllers of all generations, whether PLC- or PC-based, including software controllers. With TIA Portal, users can design and simulate automation projects, perform diagnostics and troubleshooting, and seamlessly integrate different components of an industrial system [18].

The bridge node is capable of writing in the PLC memory and the main purpose of the TIA portal part is to create a suitable structure in the memory for the data to be written in. To allow general use of this bridge, the TIA portal part was implemented as a library. The library contains publisher and subscriber datatypes. In order to subscribe from ROS2 topics, it is necessary to create a Subscriber data block. This can be done manually, or by editing the example subscriber provided in the library. Subscriber block must only contain subscriber types.

The structure of the datatypes is visible in a Diagram 3.5. Subscriber type consists of three parameters, The first is the datatype, which is selected automatically by selecting the correct type of the subscriber itself. The second is a topic, here it is possible for the user to specify the topic which should be subscribed to. The last parameter is a payload, here the subscribed message will appear and can be further used.

Subscriber1	"ROS_Subscriber_St..	0.0	
Subscriber	Struct	0.0	
Datatype	Byte	0.0	16#1
Topic	String[32]	2.0	'StringSub'
Payload	String[32]	36.0	''

Figure 3.3: Example of a subscriber object in TIA portal

The publisher block is created the same way as the subscriber and can only contain publisher types. Each publisher contains four parameters, datatype and topic work the same as in subscriber. Compared to the subscriber, the publisher contains an extra parameter Period, which selects the period of publishing messages. Payload is the message that will be published to the ROS2 topic.

publisher1	"ROS_Publisher_Int...	...	
Publisher	Struct	...	
Datatype	Byte	...	16#0
Topic	String[32]	...	'topicName'
Period	Real	...	1.0
Payload	Int	...	0

Figure 3.4: Example of a publisher object in TIA portal

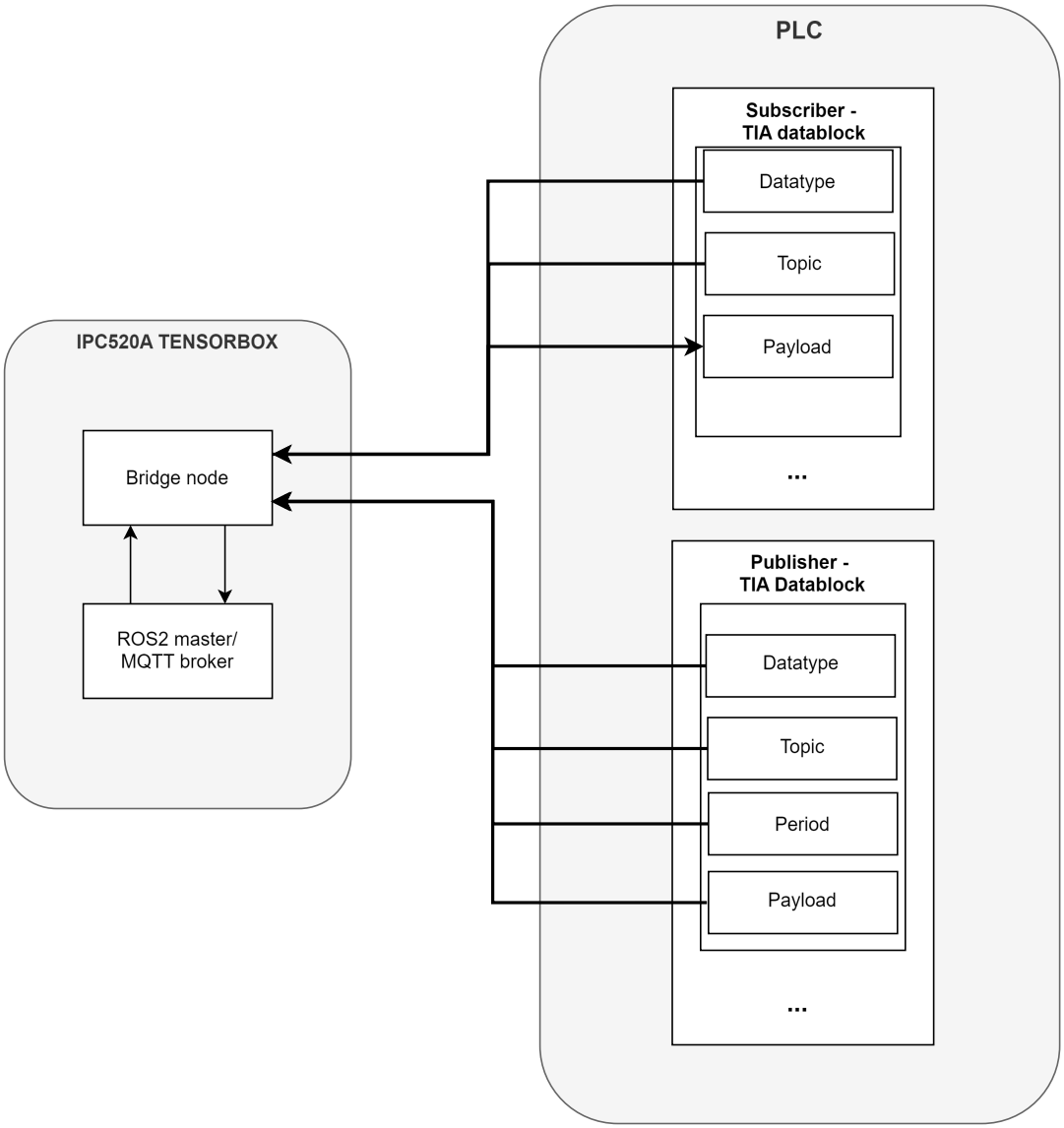


Figure 3.5: Communication diagram

Chapter 4

Motor control

The communication architecture chapter described the transfer of forward and angular velocity as a Twist datatype into the subscriber data block on PLC. This chapter will describe the transformation of this information into the rotation of the wheels. It will start with the equations used to transform the Twist message into wheel velocity setpoints. The second part will discuss the implementation of the inverse kinematics and control of the motors.

4.1 Inverse velocity kinematics

The goal of this section is to convert the forward and angular speed in m/s and rad/s of the robot to individual RPM setpoints of the motor.

This chapter will start with a list of variables (Table 4.1) for better orientation, followed by Figures 4.1 and 4.2 showing the variables on a sketch of the Automated Guided Vehicle (AGV).

Variable	Description	Unit
V	Velocity (linear.x)	m/s
θ	Angular coordinate of the AGV	rad
$\dot{\theta}$	Angular velocity of the robot (angular.z)	rad/s
x	X coordinate of the AGV	m
y	Y coordinate of the AGV	m
l	Track width	m
G_r	Gear ratio	
r	Wheel radius	m
V_l	Linear velocity of left wheel along ground	m/s
V_r	Linear velocity of right wheel along ground	m/s
ω_r	Angular velocity of the right wheel	RPM
ω_l	Angular velocity of the left wheel	RPM
ω_{for}	Forward component of the angular wheel velocity	RPM
ω_{ang}	Angular (AGV rotation) component of the angular velocity	RPM

Table 4.1: Variables used in this chapter

The AGV uses differential drive to move. It has two driven wheels and two passive ones, the distance between the driven wheels is marked as l. The construction of the robot allows movement only on the X-axis and rotation around the Z-axis. The forward velocity of the whole AGV will be noted as V while the forward velocities of the left and right driven wheel will be noted as V_l and

V_r respectively. Those forward velocities correspond with angular velocities of the wheels ω_l and ω_r. These can be split into a forward component ω_{for}, moving the whole AGV forward with velocity V and angular one ω_{ang}, responsible for the rotation θ of the AGV.

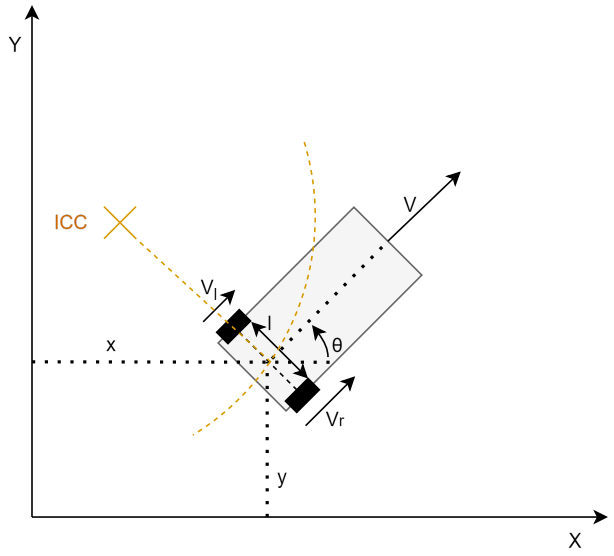


Figure 4.1: Top view of the AGV

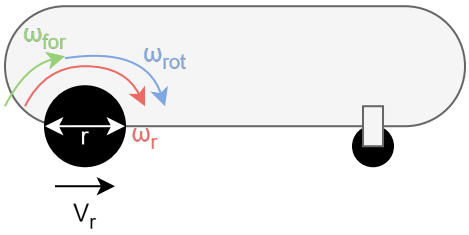


Figure 4.2: Side view of the AGV

Before getting to inverse kinematics, we will start with basic relations between the velocity of the AGV and the angular velocities of the left and right wheels. The forward velocity of the AGV can be calculated as the average forward velocity of the left and right wheels.

$$V = \frac{V_r + V_l}{2} \tag{4.1}$$

The angular velocity of the AGV can be calculated as

$$\dot{\theta} = \frac{V_r - V_l}{2l} \tag{4.2}$$

Now we can start with the inverse kinematics, for now, we will split the problem into two and solve

the forward and angular parts separately.

$$\omega_r = \omega_{for} + \omega_{rot} \tag{4.3}$$
$$\omega_l = \omega_{for} - \omega_{rot} \tag{4.4}$$

Assuming there is no slip on the wheels, we can get the pure forward movement just by rotating the wheels at the same angular velocity ω_{for}.

$$\omega_l = \omega_r = \omega_{for} \tag{4.5}$$

To calculate these angular velocities, we will take the requested velocity V and divide it by the circumference of the wheel. As we are transforming meters per second to rotations per minute, we also need to convert the requested velocity to meters per minute.

$$\omega_{for} = \frac{60 \cdot V}{2\pi \cdot r} \tag{4.6}$$

Where r is the radius of the wheel. To increase torque a 3:8 gearbox was used, which also needs to be accounted for. The gear ratio will be symbolised as G_r

$$\omega_{for} = \frac{60 \cdot V}{2\pi \cdot r \cdot G_r} \tag{4.7}$$

The second part is the rotational component of the AGV motion. In order to rotate, one wheel has to rotate faster than the other. The centre of rotation is a point that lies along the common wheel axis. The specific point that the robot rotates around is called the Instantaneous Center of Curvature (ICC) [4]. The distance between the centre of the wheels and ICC can be calculated using angular velocities of left and right wheel ω_l and ω_r as

$$R = \frac{l}{2}\frac{\omega_l + \omega_r}{\omega_r - \omega_l} \tag{4.8}$$

If $\omega_l = -\omega_r$ then $R = 0$ and the AGV rotates around the center between the wheels. These velocities will be utilized for rotational kinematics simplification.

The angular velocity of the AGV can be calculated using following equation [4]

$$\dot{\theta} = \frac{V_r - V_l}{l} \tag{4.9}$$

where l is the track width and V_r and V_l are linear velocities of wheels along the ground. As mentioned above, we will set $V_l = -V_r$

$$\dot{\theta} = \frac{2V_r}{l} \tag{4.10}$$

The requested linear wheel velocity can be acquired as

$$V_r = \frac{\dot{\theta} \cdot l}{2} \tag{4.11}$$

Now we can convert the linear velocity to the angular velocity of the motor as we did above by multiplying it by $\frac{60}{2\pi r G_r}$

$$\omega_{rot} = \frac{60 \cdot \dot{\theta} \cdot l}{4\pi \cdot G_r} \tag{4.12}$$

The last step is to add the rotational and forward parts together to obtain the final angular setpoint velocities of the wheels as shown in (4.4).

4.2 PLC Implementation

The robot contains two F-TM StepDrive cards that drive the Raveo stepper motors. Those cards connect directly to the PLC and require a separate power supply to run. Motor control was implemented on the PLC using the TIA portal. Siemens provides a TIA portal extension, that adds the stepper cards as local modules. We added a module for each card and set the parameters according to the motor documentation as can be seen in Figure 4.3. As we wanted to control the speed of the motors, we created a TIA portal SpeedAxis technology object for each motor and paired them with the modules.

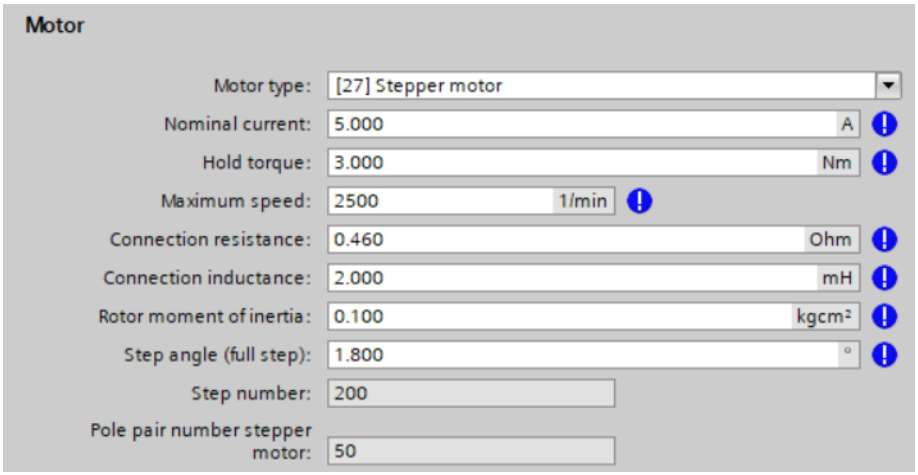


Figure 4.3: Parameters of the motor

The TIA project contains one Main organization block, which calls all blocks inside it every cycle. The project also includes publisher and subscriber blocks from Chapter 3. In every cycle, the data is read from the subscriber block and used as input for the RobotKinematics block, which includes the kinematics described in the previous chapter. The code can be seen below.

```
z_angle = atan(m21 / m11)
#leftForwardVelocity := 60.0*#twist.linear.x/(2*#wheelRadius*#PI);
#rightForwardVelocity := 60.0*#twist.linear.x/(2*#wheelRadius*#PI);
#rotationalComponent := (60.0*#twist.angular.z * #trackWidth)/(4*#wheelRadius*#PI);
#leftWheelVelocity := (#leftForwardVelocity-#rotationalComponent)/#gearRatio;
#rightWheelVelocity := (#rightForwardVelocity + #rotationalComponent)/#gearRatio;
```

There is one DriveControl (Figure 4.4) function block for each motor in the Main block and it takes five variables as input. First is the power, which specifies whether the motor is on or off. The motor can be reset by setting the reset signal to True. The third parameter is velocity, this is the velocity setpoint from the RobotKinematics block. The next parameter is moveOn, this allows us to stop the movement without cutting off the power from the motor. The last parameter is an axis, where the SpeedAxis object is specified. The DriveControl block uses standard library functions MC_POWER, MC_RESET and MC_MOVEJOG for powering, resetting and moving the motors.

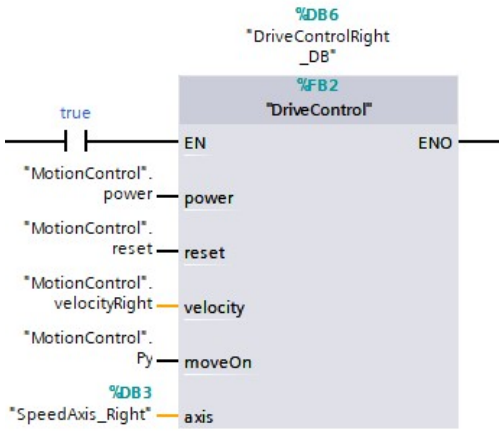


Figure 4.4: Drive control block

Chapter 5

Planing of control sequence using gradient descent

The control loop can be divided into six basic parts as shown in Figure 5.1. Inverse velocity kinematics, driver cards and stepper motors have been mentioned before. The employment of the encoder and localisation will be discussed in one of the following chapters. In this chapter, we will describe the development of algorithms used to plan a control sequence, for the AGV. The goal is to use an estimate of the current location to create a Twist velocity setpoint, that will lead the AGV towards the current target. The chapter will start with the kinematic model used for modelling the AGV motion. Next, the cost function will be defined along with the general algorithm used to find a minimum of this function. After that, a few adjustments to the algorithm will be described, which will improve the quality of the path found as well as the time it costs to find the solution.

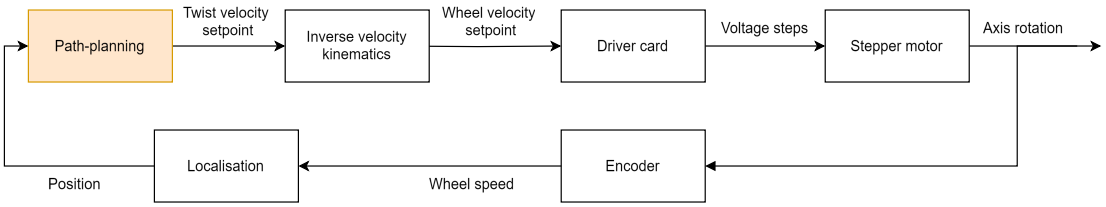


Figure 5.1: Control loop

There are many path-planning algorithms for AGVs. Some of them focus on time efficiency but don't provide optimal solutions. Different algorithms can give better solutions but are computationally expensive. Many path-planning algorithms, such as A* or Floyd-Warshall transform the map into a discrete graph and then solve the simplified version of the problem.

At the time of writing this thesis, the robot used relatively slow motors. This isn't much of an issue as full autonomy allows it to work almost permanently and it has interesting implications for path planning. The small velocity with a combination of high-end hardware necessary for neural network applications gives us a lot of computational capacity for path planning. With this setup, it is preferable to use a more demanding algorithm, that would decrease the necessary travel distance. On the other hand, the environment isn't very complicated and the planning usually consists of efficiently transferring between rails and avoiding simple obstacles. This allows us to combine the path-planning and generation of control sequences. A benefit of this approach is high flexibility as the AGV would react to an obstacle in a single control step after the obstacle was detected.

5.1 Single step optimisation

Before getting to the control, we need to create an estimate of the pose the AGV will move to in one control step. The movement of the AGV can be described by the following differential drive kinematic equations [4]. These equations describe the evolution of the pose of the robot based on forward (v_k) and angular (ω) velocity. The variables used are visible in 4.1.

$$\dot{x} = cos(\theta) \cdot V \tag{5.1}$$
$$\dot{y} = sin(\theta) \cdot V \tag{5.2}$$
$$\dot{\theta} = \omega \tag{5.3}$$

We will use the discrete variant of this model:

$$x_{k+1} = x_k + cos(\theta_k) V_k dt \tag{5.4}$$
$$y_{k+1} = y_k + sin(\theta_k) V_k dt \tag{5.5}$$
$$\theta_{k+1} = \theta_k + dt \cdot \omega_k \tag{5.6}$$

In these equations, the x_{k+1} and y_{k+1} represent the updated x and y coordinates. The orientation of the AGV is represented by θ and the timestep by dt.

This model was used as a part of the planning algorithm to create predictions of the movement and also as a simulated feedback for testing the solution.

The model gives a position and rotation estimation based on the control variables V_k and ω_k. The two variables are the input to the motor control described in the previous chapter. We can define a vector u_k that combines the two control variables together.

$$u_k = \begin{bmatrix} V_k & \omega_k \end{bmatrix}$$

If we note the pose of the AGV as Ψ and the set of kinematic equations (5.6) as f, we can estimate a pose in $k+1$ as

$$\Psi_{k+1} = f(\Psi_k, u_k) \tag{5.7}$$

Both the V_k and ω_k are limited by the maximum velocity of the motors as well as by user-defined restrictions. This gives us an interval of possible vectors u_k. Equation (5.7) transforms the interval of all possible u_k into a set of all reachable poses Ψ_{k+1}. Next, we need to choose the best pose from the set in a metric, that will help the AGV get to the desired destination. Therefore we need to create a cost function that will evaluate all the possible poses. The cost function J was chosen as a three-dimensional Euclidean distance with the x and y axis as two of the dimensions and the third dimension γ which represents the angular distance and will be further discussed later.

Cost J is calculated using formula

$$J(\Psi_k) = \sqrt{(x_k - x_g)^2 + (y_k - y_g)^2 + \gamma_k^2} \tag{5.8}$$

Where x_{k+1}, y_{k+1} are the x and y coordinates of the expected position in timestep $k+1$, x_g, y_g are the x and y coordinates of the requested goal position and γ is the angular distance. If we have an obstacle or a zone we want to avoid, we need to increase the cost function by a value representing how much we want to avoid this zone. Figure 5.2 shows a visualisation of the cost J with $\gamma = 0$.

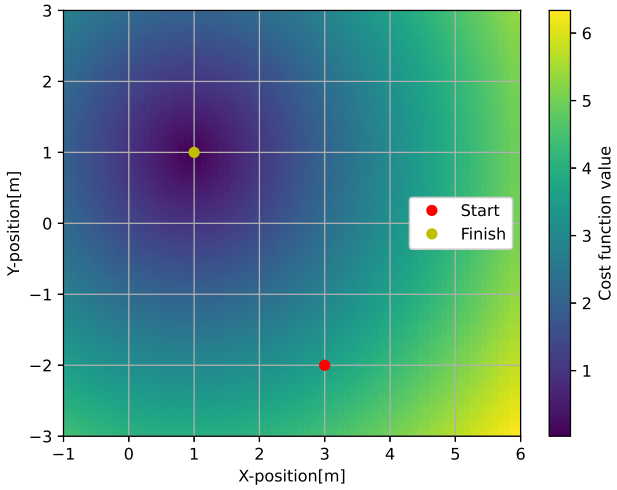


Figure 5.2: Visualisation of a cost function J without an angular distance

Distance between two angles is important for evaluating different poses or for feedback control. The first thing that needs to be considered is the range of rotation. For our purposes, angles 0 rad and 2π rad can be considered equal as there is no need to count the total number of rotations. The difference between the two angles needs to be modulated by 2π. Both clockwise and counterclockwise distances are calculated and the lower one is selected as visible in Figure 5.3. The final angular distance between angles α and β is acquired as:

$$\gamma_1 = |(\alpha - \beta) \mod (2\pi)| \tag{5.9}$$

$$\gamma_2 = 2\pi - |(\alpha - \beta) \mod (2\pi)| \tag{5.10}$$

$$\gamma = \min(\gamma_1, \gamma_2) \tag{5.11}$$

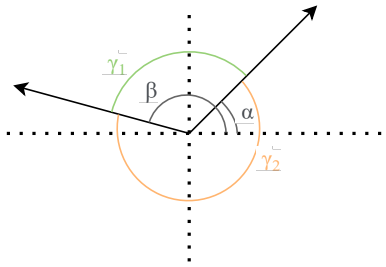


Figure 5.3: Distances between two angles

These equations have been implemented in the following way

```
    def angular_distance(A, B):
        angle_diff = (A - B) % (2 * math.pi)
        dir1_dist = abs(angle_diff)
        dir2_dist = 2 * math.pi - abs(angle_diff)
        return min(dir1_dist, dir2_dist)
```

```
    def cost_function(A, B):
        ang_dist = angular_distance(A[2],B[2])
        return math.sqrt((A[0]-B[0])^2+(A[1]-B[1])^2+ang_dist^2)
```

```
    def differential_drive(control, current_pose):
        omega = control[0]
        velocity = control[1]
        new_pose = [0,0,0]
        new_pose[0]=current_pose[0]+ math.cos(current_pose[2])*T*velocity
        new_pose[1]=current_pose[1]+ math.sin(current_pose[2])*T*velocity
        new_pose[2]=current_pose[2]+ T*omega
        return new_pose
```

Now if we have a target pose, we can assign a cost to any possible pose of the robot. We will take all the poses that are reachable in $k+1$ and choose the one with the lowest value of $J(\Psi_{k+1})$. To get the control u_k that drives the AGV to the pose with a minimum value of J, we will use a gradient descent algorithm.

Gradient descent is one of the most popular numerical algorithms to perform static optimisation. It is an algorithm to minimize an objective function parameterized by the model's parameters. It works by updating the parameters in the opposite direction of the gradient of the objective function. The update of the parameter is composed of the gradient of the objective function multiplied by step length η [25].

$$u = u - \eta \cdot \nabla_u J(f(\Psi, u)) \tag{5.12}$$

The correct choice of step length is important so that the algorithm doesn't move too fast or too slow. In our case, the step size was constant and heuristically tuned.

The gradient will be estimated numerically using the central difference formula where h is a finite difference step size.

$$\frac{J(f(\Psi_k, u+h)) - J(f(\Psi_k, u-h))}{2h} \tag{5.13}$$

The final algorithm will be following:

1. We start by initialising the control variables. At this point, we will initialise them as a random value in an interval reasonable to the robot's limitations. The robot's current pose Ψ_k is obtained from the localisation node.

```
omega = random()
velocity = random()
```

2. Next, we can use a central difference formula to numerically approximate the gradient of $J(f(\Psi_k, u_k))$. We will use the differential drive model (5.6) to estimate the pose of the robot in the next time step for $\omega \pm h$ and $V \pm h$.

```
grad_omega = (cost_function(differential_drive(current_pose, [omega+
    finite_difference_step_size, velocity]))-cost_function(differential_drive(
    current_pose, [omega-finite_difference_step_size, velocity])))/(2*
    finite_difference_step_size)
grad_velocity = (cost_function(differential_drive(current_pose, [omega, velocity+
    finite_difference_step_size]))-cost_function(differential_drive(current_pose,
    [omega, velocity-finite_difference_step_size])))/(2*
    finite_difference_step_size)
```

3. Next, we update the control variables using the calculated gradient

```
omega = omega - step_length*grad_omega
velocity = velocity - step_length*grad_velocity
```

4. The two steps of estimating the gradient and updating the control variables would repeat until one of the three stopping conditions had been fulfilled.

   - The control variable stops updating and the following condition is triggered:

   ```
   if grad_omega < epsilon and grad<velocity <epsilon:

   ```

   where epsilon is some small constant. We are close to a solution, which leads to the highest decrease of J.

   - The number of iterations exceeded the allowed maximum. This leaves us with a sub-optimal solution, however, there has been a certain number of iterations updating the control vector towards an optimal solution and the result is usually fine to use as long as the allowed maximum isn't too low. If epsilon has been chosen as too strict, this condition would keep the algorithm running.
   - The computational time exceeded the allowed maximum. This is the worst option, the algorithm hasn't found an optimal solution and didn't manage to finish the expected amount of iterations. This condition is used to prevent blocking a processor core. Continuous triggering of this condition often indicates, that the parameters of the algorithm are set inappropriately.

5. The final value of $u_k = [V, \omega]$ is sent to motors. The algorithm obtains pose Ψ_{k+1} and returns to step 1 to calculate u_{k+1}.

5.1.1 Simulation verification of the algorithm

The control algorithm was tested using the simulated feedback. This was done by creating a second algorithm, that uses the differential drive model to calculate an estimate of where the robot would move in real time and providing this position back to the path planning node. A separate node was created so that the simulation could be replaced by sensor localization by simply exchanging the nodes. This node subscribes from ROS Twist topic cmd_vel and sums the velocities into position using the mentioned discrete equations of the differential drive model (5.6). After calculation of the pose of the AGV, it publishes it to ROS2 topic.

Its goal was to travel from pose $[3, -2, 0]$ to pose $[1, 1, 3.14]$. Computational demands of the gradient descent algorithm were below 1 ms per step on average. This leaves a lot of margin for the requested 10 Hz control. The main problem however was that it did not always reach the desired position as it often ended in local minimum. This is visible in Figure 5.4. The gradient descent method in its basic variant is susceptible to ending in a local minimum, therefore this behaviour is not unexpected.

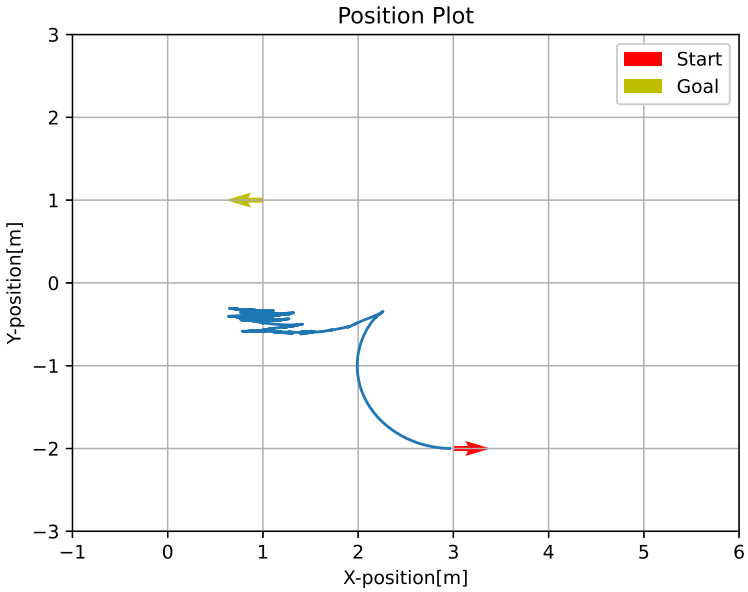


Figure 5.4: Test of the pathfinding algorithm with single-step optimization.

5.2 Receding horizon control

To reach the requested position, which can be considered a global minimum we need to get through the local minimum without stopping there. This can be solved by utilizing a receding horizon technique and optimising a sequence of steps instead of just one step and then using the first step for control. We will define a new control sequence U_k for the whole horizon of control and a kinematic function F, which will be described later. The control sequence U_k is a generalised version of u_k for the horizon of n steps and it will look as

$$U_k = \begin{bmatrix} V_k & \omega_k \\ V_{k+1} & \omega_{k+1} \\ V_{k+2} & \omega_{k+2} \\ \dots & \dots \\ V_{k+n} & \omega_{k+n} \end{bmatrix} \tag{5.14}$$

The cost function remains the same, but instead of calculating the distance to the next point, it uses the distance to the last point of the sequence.

$$J(F(\Psi_k, U_k)) = J(\Psi_{k+n}) \tag{5.15}$$

The position of the last point is obtained by recursively using the prediction model on a set of angular and forward control velocities.

$$\Psi_{k+1} = f(\Psi_k, u_k) \tag{5.16}$$
$$\Psi_{k+2} = f(\Psi_{k+1}, u_{k+1}) \tag{5.17}$$
$$\Psi_{k+3} = f(\Psi_{k+2}, u_{k+2}) \tag{5.18}$$
$$\dots \tag{5.19}$$

For example, for $n = 3$ we can use this equation

$$F(\Psi_k, U_k) = f(f(f(\Psi_k, u_k), u_{k+1}), u_{k+2}) \tag{5.20}$$

If the robot encounters a local minimum, its surrounding is searched for a better point. This allows the planning to overcome local minima, but also significantly increases the computational complexity. This algorithm was tested with multiple settings and we will discuss four of them. The tests are visible in the Figures 5.5, 5.6, 5.7 and 5.8. The task of this test was the same as in the previous chapter, to travel from pose $[3, -2, 0]$ to pose $[1, 1, 3.14]$.

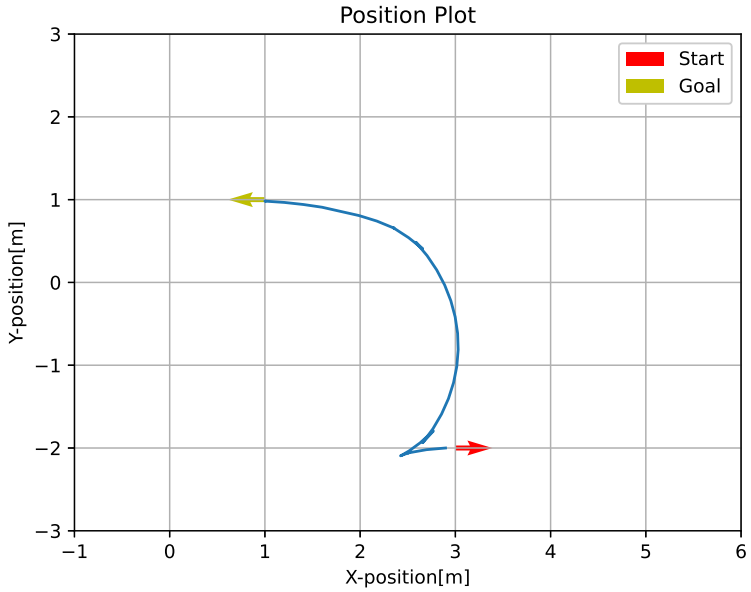


Figure 5.5: Trajectory of the first test

For the first test, the maximum velocity was set to 1m/s and the size of the horizon to 5. As visible in Figure 5.5, the simulated AGV reached the goal point without any issues.

In the next test, we decreased the maximal velocity to 0.1 m/s, which should be more realistic for the AGV used. This caused the algorithm to plan a smaller distance in advance, which resulted in a decrease in the overall quality of control. In this case, it is visible in Figure 5.6 that the algorithm got stuck in the local minimum.

For the next test, the size of the horizon was increased to 15 steps. As you can see in Figure 5.7 the planner uses quite an ineffective way to move along the Y axis, this can be caused by the fact that as it is controlled at 10 Hz at 0.1m/s and 15 steps in future, the algorithm only plans for $15 \cdot 0.1 \cdot 0.1 = 15cm$.

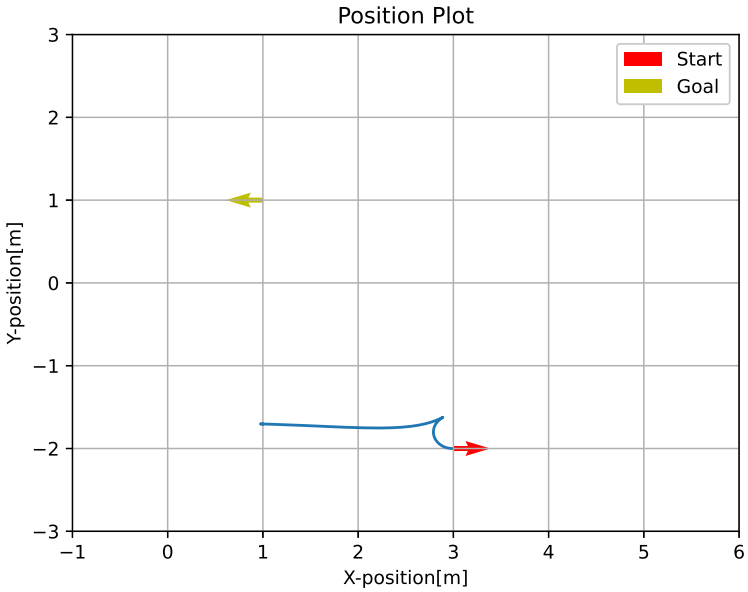


Figure 5.6: Trajectory of the second test

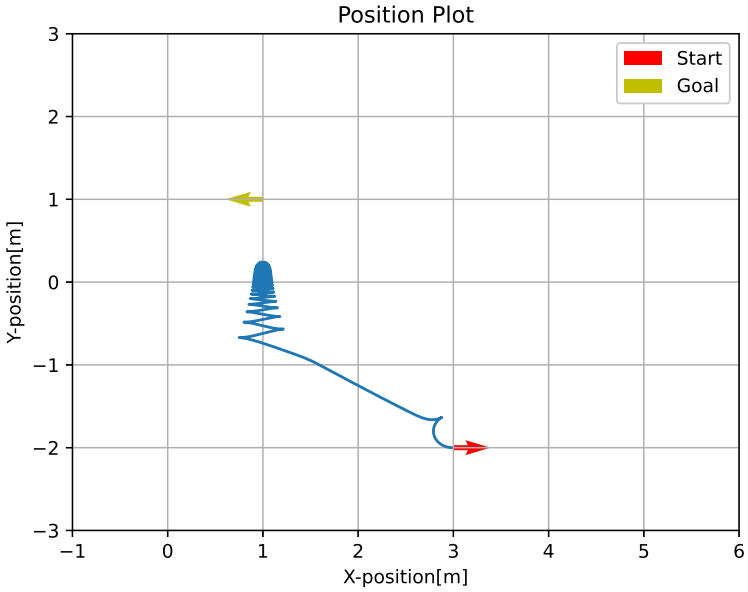


Figure 5.7: Trajectory of the third test

In the next test we doubled the size of the horizon to 30 steps, which significantly improved the quality and made the trajectory more straightforward, but it also nearly quadrupled the processing time to more than 60ms. Further increase in the number of steps could make the algorithm too slow

Test number	Horizon size	Average time	Velocity
1	5	1.483ms	1m/s
2	5	1.779ms	0.1m/s
3	15	15.598ms	0.1m/s
4	30	60.717ms	0.1m/s

Table 5.1: Parameters of the tests

for control at 10 Hz. Figure 5.8 shows quite an effective path.

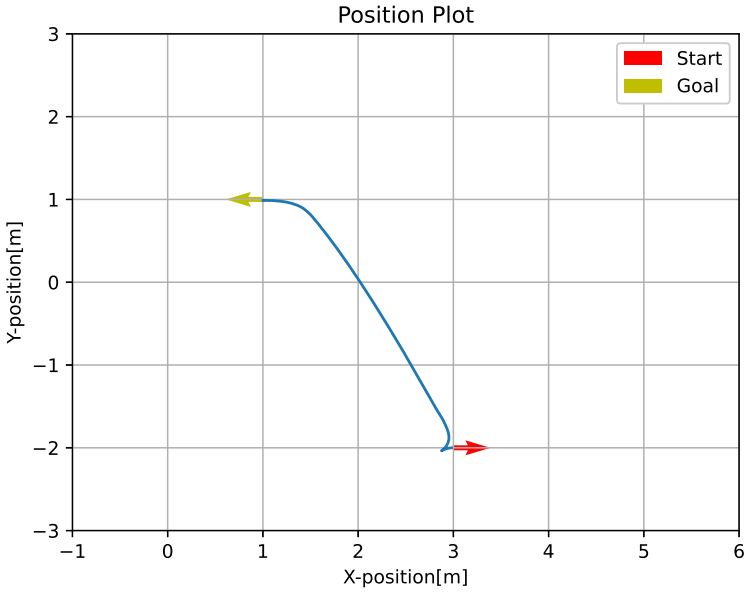


Figure 5.8: Trajectory of the fourth test

Table 5.1 shows the fast growth of computational demands with the increase of the horizon size.

5.3 Cost function adjustment

The path planning is based on minimising a cost function. This cost function is defined by an Euclidean distance between the current pose and the target pose. However, the pose vector contains the X and Y positions in metres and the rotation in radians. A normalisation factor needs to be found to compare the cost of positional displacement and an angular one. We will create a normalisation factor C_p by adjusting the cost function J to

$$J(\Psi_{k+1}) = \sqrt{(x_{k+1} - x_g)^2 + (y_{k+1} - y_g)^2 + C_p \cdot \gamma^2} \tag{5.21}$$

When this factor is chosen too big, the algorithm will tend to prefer correct rotation before the correct position as shown in Figure 5.9.

Decreasing this factor will make the algorithm prefer the correct position and decrease the chance, that it will run into local minimum, but having the value too low may have a negative impact on

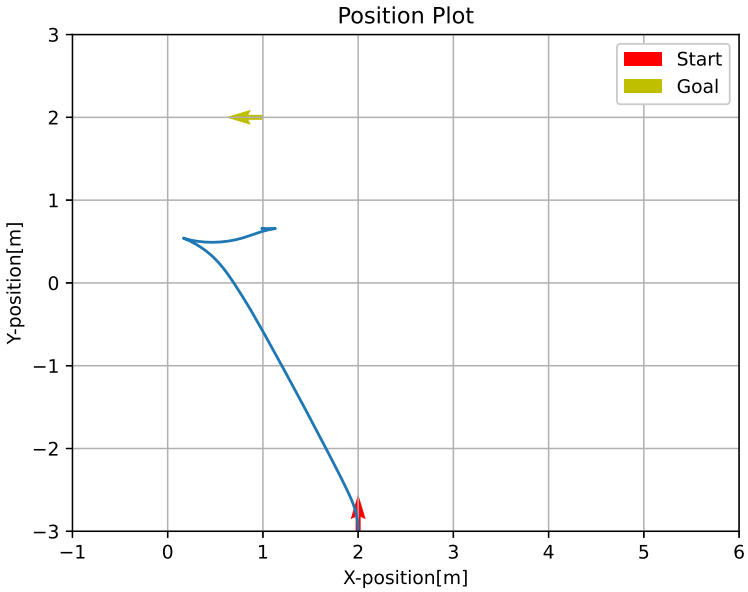


Figure 5.9: AGV running in local minima with rotation price constant equal to 1

the travel time and movement efficiency. When the AGV is approaching the target point, the cost of positional displacement decreases, while the cost of rotational displacement remains the same as long as the AGV doesn't change its direction. At some point, the cost of position displacement decreases below the price of rotation and the AGV starts rotating even though it didn't reach the target. For example, in Figure 5.9, the AGV is approaching its target at approximately 90 degrees - approximately 1.6 radians. The rotation price constant is set to 1, so the price of 1 meter is equal to 1 rad. As can be seen in Figure 5.9, the first undesired rotation ends at approximately 1.6 metres from the target point. We can consider the positional part J_p and the angular part J_a of the cost function separately.

$$J = \sqrt{J_p + C_p \cdot J_a} \tag{5.22}$$

$$J_p = (A_x - B_x)^2 + (A_y - B_y)^2 \tag{5.23}$$

$$J_a = \gamma^2 \tag{5.24}$$

The J_a has a minimum at any pose that is rotated in the same direction as the target rotation. In combination with the kinematics of the robot, it creates a line of possible dead ends. Those don't cause many problems as long as the position displacement is high enough, because the algorithm will prefer minimisation of the position and travel towards the target. The unwanted behaviour begins when $J_p < C_p \frac{\pi}{2}$. This creates a circle around the target in which the described attraction to a local minimum could happen. In order for this not to happen the planning algorithm has to calculate enough steps in advance, that it is able to reach the target (the global minimum) in this number of steps. This creates an area around the AGV, that the algorithm can scan for local minimuma. The size of this area is dependent on the distance, the AGV is able to reach in a certain number of steps. This distance is dependent on the velocity, the AGV is moving at, so it is important to consider the lowest

expected velocity. We will simplify the area to a circle around the AGV with a radius defined by the size of horizon n, maximal velocity V_m and sampling period dT.

$$n \cdot V_m \cdot dT \tag{5.25}$$

To avoid the described issues, it is preferable that the circle around the AGV is bigger than the one around the target.

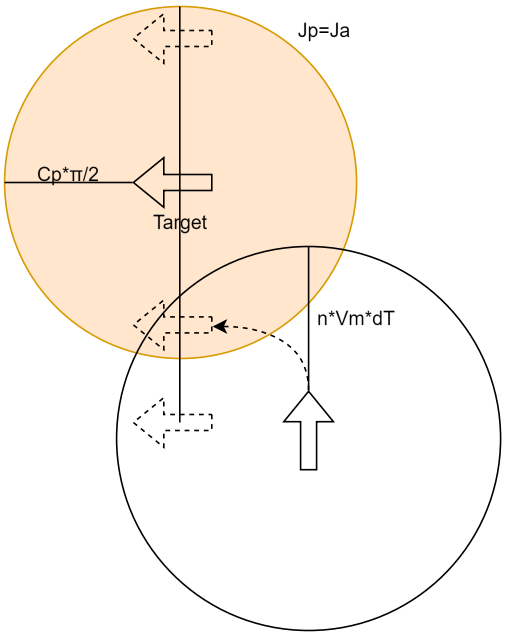


Figure 5.10: Visualisation of scanned and problematic areas

This way the algorithm should find the global minimum before falling into the local one. As the increase of steps highly affects the computational complexity, it may be easier to decrease the size of the target circle than to increase the scanned area. We can do this by simply choosing the angular price constant C_p lower than

$$C_p < \frac{2n \cdot V_m \cdot dT}{\pi} \tag{5.26}$$

where n is a size of the receding horizon, V_m is the minimal expected velocity, and dT is a time of one step. This considers only the minimal distance needed to reach the target, considering the AGV wouldn't travel by a straight line, but rather by a circular section (quarter circle), we should divide it by another $\frac{\pi \cdot \sqrt{0.5}}{2}$ and make it more strict. The maximal suggested value of the rotation price constant is

$$C_p < \frac{4n \cdot V_m \cdot dT}{\sqrt{0.5}\pi^2} \tag{5.27}$$

For example, for a maximum speed of 0.1m/s, 25 steps horizon and control at 10 Hz, we would need a rotation price constant lower than 0.145. Figure 5.11 shows a test with a rotation price constant of 0.1, which significantly improved the control quality. This approach doesn't give an optimal value of C_p, but rather an upper constriction and the value should be further tuned and tested.

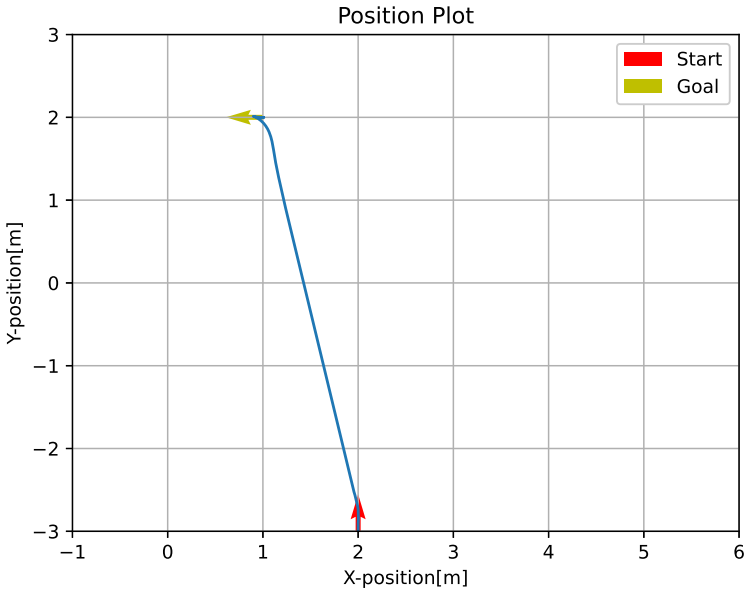


Figure 5.11: AGV running smoothly with rotation price constant equal to 0.1

5.4 Initialisation data

An important problem is, how to select the initial velocity and angular velocity vector. So far, the vector was selected randomly with the limitation of maximal velocity. However the planning algorithm works with the prediction of AGV movement, so we can use the precalculated path from the previous step and adjust it according to the measured position. In the simulated test, this lowered the average computation time by more than half.

5.5 Automatic increase of step count

The experiments showed, that there is a huge impact of the number of the size of horizon on the quality of the chosen route. The AGV can take long amounts of time or even not reach the target at all if this number is too low. That's why the automatic increase of precalculated steps was implemented. First, it was necessary to detect that the AGV is not moving in the right way. At every step, the distance from the target is saved into an array. The control algorithm periodically checks the array and calculates the average velocity at which the AGV moves closer to the target and compares it to the maximum allowed speed of AGV. If this velocity is significantly lower, AGV is probably not moving in the right direction and therefore the number of steps needs to be increased.

5.6 End of the rail

So far, the control algorithm considers only the travel on the concrete platform, where the AGV can move freely. The movement along the rail is much simpler as the robot can move only forwards or backwards. The only issue encountered was the safe detection of the end of the rail. The control algorithm keeps the AGV in motion until it reaches the desired destination. There are multiple ways to check if the AGV reached its target. The simplest one would be to calculate the distance from the

target with every new location message and to stop the AGV if a certain threshold is reached as in Figure 5.12. However, this approach can be problematic when there is too much noise or when the period between localisation steps gets too long. Instead, the limitation to a single dimension can be used to define a half-plane where the AGV is supposed to stop. This is possible thanks to the fact that the AGV is moving on rails. The halfplane is shown in Figure 5.13.

To get the half-plane, we start by getting the vector of the rail from the rail start and end coordinates. A is the point of start, B is the point of end and C is the position of AGV.

$$\vec{AB} = [x_{\vec{AB}}, y_{\vec{AB}}] = [x_B - x_A, y_B - y_A] \tag{5.28}$$

Next, we get the perpendicular vector AB_n from the coordinates x and y of the vector AB

$$\vec{AB}_n = [x_{\vec{AB}_n}, y_{\vec{AB}_n}] = [-y_{AB}, x_{AB}] \tag{5.29}$$

Now the relative position of the AGV to this line can be checked using the following equation.

$$p = x_{\vec{AB}_n} \cdot (y_k - y_B) - y_{\vec{AB}_n} \cdot (x_k - x_B) \tag{5.30}$$

If the value of p is negative, the AGV reached the target half plane and should be stopped.

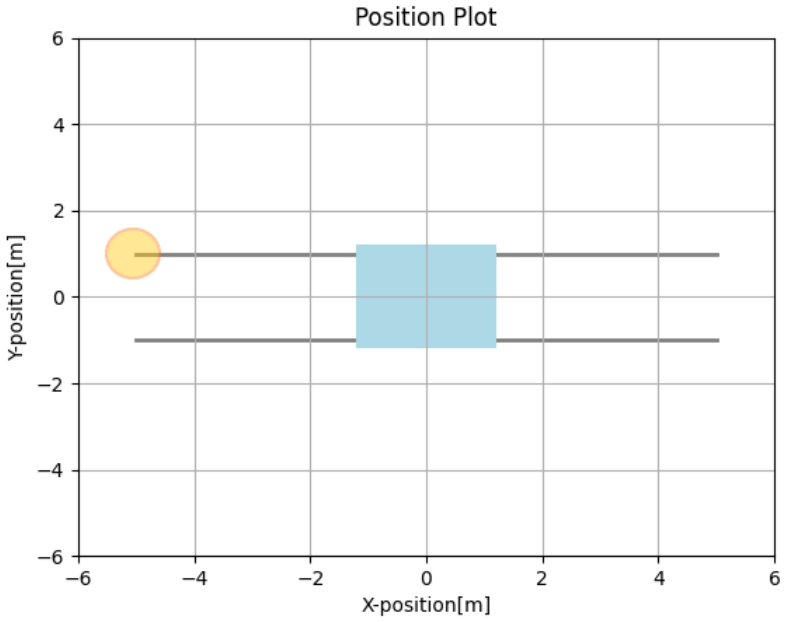


Figure 5.12: Target defined by distance threshold

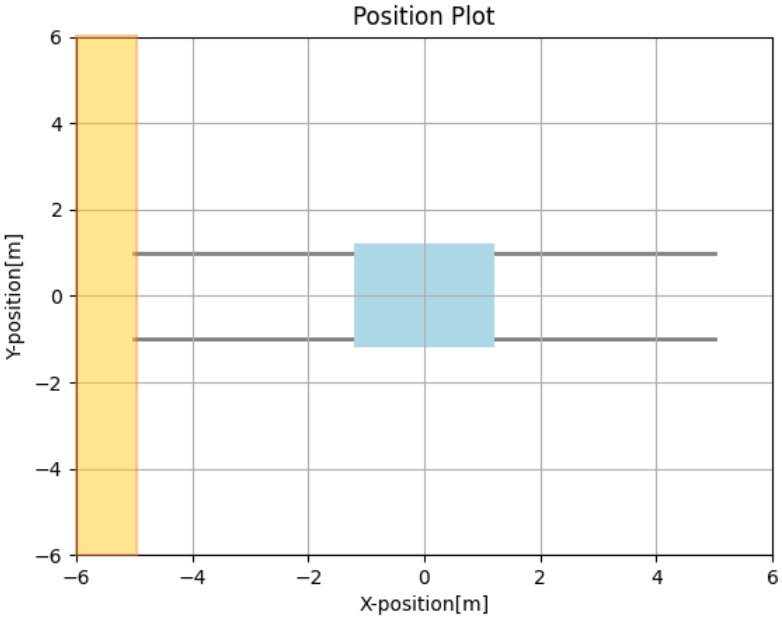


Figure 5.13: Target defined by half-plane

Chapter 6

Test of the AGV control system using software in the loop

During the development of the robot control system, the Software-in-the-loop (SIL) was utilized. This approach resulted in a more reliable and efficient system without requiring physical access to the robot or risking any harm to the people or hardware. The controller was used without any change, the PLC was simulated using PLCSIM Advanced and the AGV was simulated in NX Mechatronic concept desinger (MCD). This chapter will start with an introduction to SIL in general, next it will explain the implementation using PLCSIM advanced and NX MCD. Diagram 6.1 shows the general setup of the control loop, starting from the left with the path-planing controller described in the previous chapter, the setpoints for the wheels are sent to a virtual PLC simulated by PLCSIM Advanced, which resends them to NX MCD simulation. The pose of the AGV is then exported from the simulation back to the virtual PLC, which resends it to the path planner.

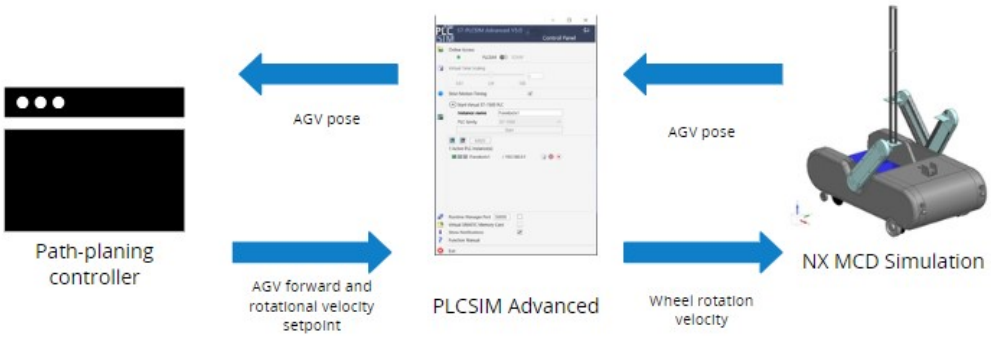


Figure 6.1: Diagram describing software in the loop architecture

Mechanical systems are commonly tested in laboratories to get their characteristics and gather important data. This however brings additional costs and risks to the development of robotic or other systems. SIL testing is an emerging approach that has evolved from conventional computer simulations. In SIL testing, the test environment is composed of both real and virtual components, typically involving a controller, a plant, and a virtual simulation implemented in real-time. This setup allows the virtual part to control the physical plant using the appropriate actuators and sensors.

The concept of SIL testing varies depending on the specific application, offering several potential benefits [19].

- Only parts of the system need to be tested physically, which reduces the cost and complexity
- New system can be tested before manufacturing
- Even conditions that are difficult or impossible to test in the laboratory can be tested in simulation
- Characteristics of the simulated system can be easily and quickly changed
- Testing in simulation can reduce the health risks for testers

6.1 Implementation of the simulation loop

PLCSIM Advanced is a tool from the TIA Portal package. It is a virtual commissioning and simulation environment for testing and validating automation systems. It allows the creation of virtual representations of programmable logic controllers and simulates their behaviour before deployment in production. This enables testing, troubleshooting and optimization without a need for real PLC. It is possible to connect to the simulated PLC using Transmission Control Protocol/Internet Protocol (TCP/IP), so the developed ROS bridge can be used for communication. In the Software-in-the-loop part, it was used to simulate the ET200 used on the AGV.

Siemens NX is a software platform for computer-aided design, manufacturing and engineering. NX provides an integrated environment to create and manage product designs. It supports parametric modeling, that allows users to make easy modifications. One of the notable features of NX MCD is its simulation capabilities. Part of NX is a NX Mechatronic Concept Designer which is a set of tools for creating designs of mechatronic systems. In this work, it will be used as a simulation engine for the creation of a digital twin of the AGV.

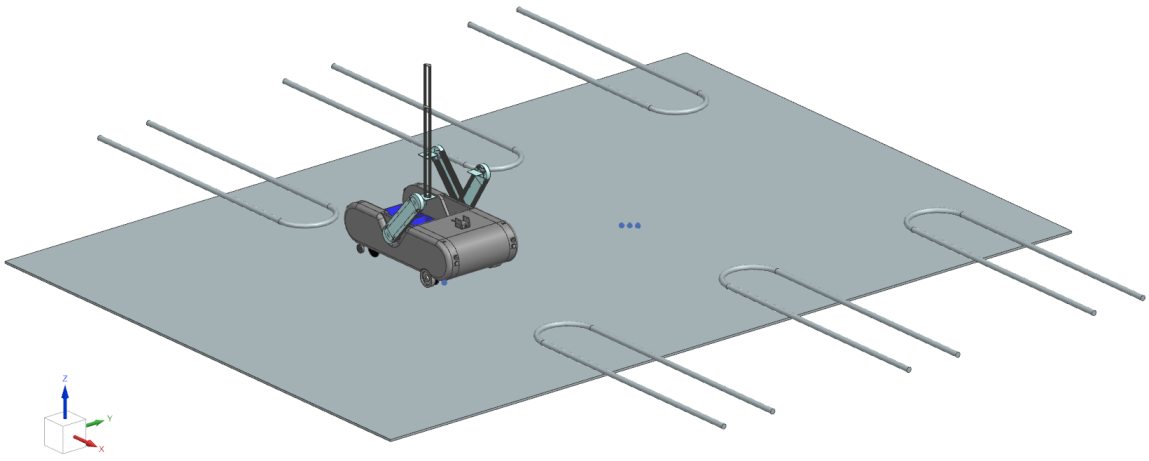


Figure 6.2: NX model of AGV

The implementation started by importing an Standard Triangle Language (STL) model of the robot to NX MCD. This model was developed by Fravebot for manufacturing purposes and is visible

in Figure 6.2. After that, the floor plane was created, which functions as a simplification for the concrete switching platform. Next, the tubing rail 3D model was created as this model was used for testing purposes, it wasn't necessary to use a real length of rail and only the beginning was created. The rail is shown in Figure 6.3 Next, the MCD was activated and it allowed the addition of physical properties to the model components. Rails and the platform were given a collision body, that allows the interaction between multiple objects having this property. The platform was given a simple box collision body. Simulating collisions in real-time can be computationally demanding, so it is usually better to simplify the shapes of collision bodies as much as possible. The collision body of every rail is composed of a box collider and a cylinder collider for the beginning of the rail as this is easier to compute than the true shape of the rails.

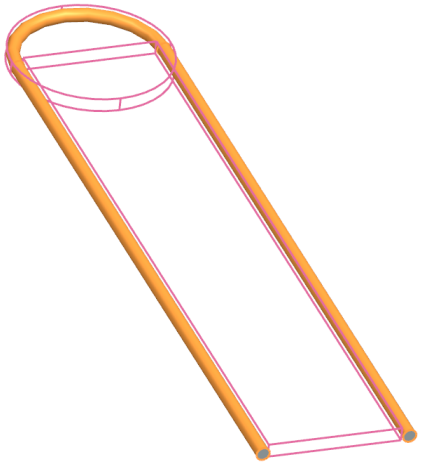


Figure 6.3: Collision bodies of rail

Cylinder colliders were used for the wheels of the AGV, and other parts of the AGV were not given a collider as only wheels are supposed to interact with the environment in this simulation. Apart from the collider wheels and the AGV itself were also given a rigid body, that allows movement in the simulation. The centre of mass was calculated automatically from the model. The rigid bodies were connected using hinge joints so that the wheels could rotate and for each joint, there was a speed controller assigned to control the speed of rotation. NX MCD offers the possibility to connect to real or simulated S7 PLC using various protocols. This allowed us to create a signal mapping connection object, and map the rotational speed of wheels from PLC to the speed controllers in NX MCD.

The signal mapping connection was also used to send the pose of the AGV back to the PLC, simulating the output of a positional sensor. X and Y coordinates were given directly in metres, however in NX MCD, the rotation of rigid bodies is saved in the rotation matrix. A rotation matrix is a square matrix that represents a rotation transformation in three-dimensional space. To get the yaw of the AGV from the rotational matrix to the Euler angle, we can use the following equation:

$$\theta = \text{atan2}(m_{21}, m_{11}) \tag{6.1}$$

Where θ is Euler angle representing the yaw of AGV and m_{21}, m_{11} are elements of the rotational matrix M

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

The AGV movement is based on a two-wheel differential drive. Due to this, the centre of rotation is not in the geometric centre of AGV. This needs to be accounted for in the path planner. We need to measure the distance between the centre of rotation and the geometric centre in simulation or sensor position in reality. Next, we can use the following equations to obtain the global coordinates of the centre of rotation.

$$x_r = x_g - \cos\theta \cdot o_v + \sin\theta \cdot o_h \tag{6.2}$$
$$y_r = y_g - \sin\theta \cdot o_v + \cos\theta \cdot o_h \tag{6.3}$$
$$\tag{6.4}$$

In these equations, x_g and y_g represent the global coordinates of the geometric centre, x_r and y_r represent coordinates of the centre of rotation and o_v and o_h represent the local, vertical and horizontal offset between the mentioned centres. In our case, o_v is 0.3m and o_h is 0. All of the mentioned variables are shown in the Figure 6.4.

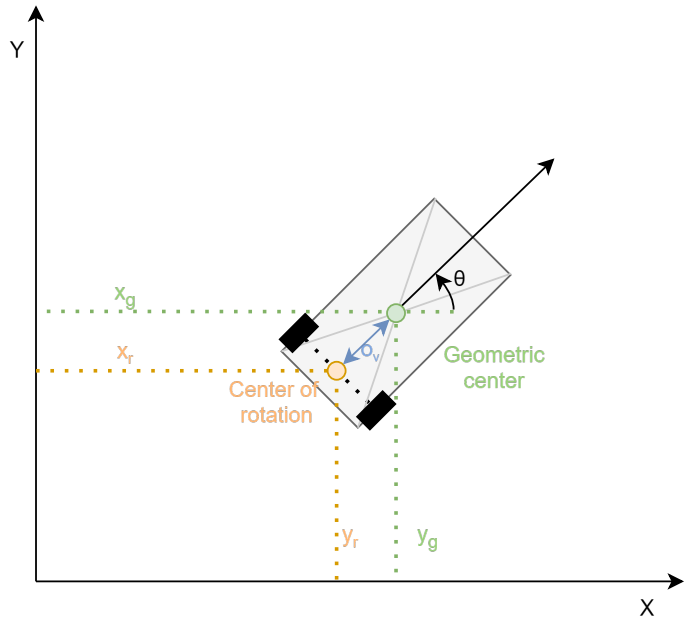


Figure 6.4: Geometric center and center of rotation of the AGV

This provided precise simulated feedback for the control planner. It allowed better tuning of the parameters of the algorithm and it showed the need for some improvements such as the already mentioned rework of the end of the rail detection.

Chapter 7

Encoder-based Dead Reconing

After testing the control system in the SIL simulation, the next step was to test it on a real AGV. The AGV is equipped with an Nvidia Jetson Orin module, with 2048 Nvidia Graphics Processing Unit (GPU) cores, an ARM Cortex-A78AE processor and 64GB of RAM [13]. The main purpose of this module is to run the neural networks used to determine the status of the plants, but it will also be used to run the controlling and positioning algorithms. This chapter will describe the process of development of the AGV localisation algorithms, starting with simple feedforward positioning. Next, the encoder on the left wheel will be utilized, however, there is no encoder on the right wheel, so its velocity needs to be estimated. For this estimation, we will develop and describe three models of the dynamics of the right wheel.

7.1 Feedforward positioning

The first version of the control system implementation used only a feedforward positioning system, based on the time and wheel velocity setpoints. The linear forward velocity V and angular velocity of the AGV can be calculated from the velocities of the left and right wheel V_l and V_r as:

$$V = \frac{V_l + V_r}{2} \tag{7.1}$$

$$\omega = \frac{(-V_l + V_r)}{l} \tag{7.2}$$

where the velocities of the left and right wheels are obtained from the angular setpoints ω_r and ω_l sent to the motors, and the radius of the wheels r

$$V_r = \frac{2\pi \cdot \omega_r \cdot r}{60} \tag{7.3}$$

$$V_l = \frac{2\pi \cdot \omega_l \cdot r}{60} \tag{7.4}$$

The velocities ω and V are then used with the kinematic equations (5.6) to obtain a position.

This approach as well as the following ones were tested by setting the motors to 0.1m/s for 10s. Therefore, the expected distance travelled is 1m. Next, we did a measurement of the true travelled distance by measuring tape. During these tests, we also captured the steps from the encoder for use in the following chapters. The results are visible in Table 7.1.

This test showed, that the feedforward positioning can provide some level of precision for part of the tests, especially the tests 2,3 and 5. However, any unexpected behaviour will lead to high errors as

in tests 1 and 4. The robot uses caster wheels as front wheels and the initial rotation of those wheels has a significant influence on the positioning precision. Thus more complex positioning is needed for a correct navigation.

7.2 Left wheel encoder

The left wheel of the robot was equipped with a high-precision rotational encoder. This encoder provides the feedback necessary to improve the robot's positioning precision.

The encoder used was REM-E-106C15E-9E43B-B3M12 made by Turck GmbH. This is an absolute, multiturn optical encoder with high measuring resolution. In this case, the resolution used was the default 13-bit resolution, however, it is scalable up to 19-bit [22]. Therefore, there are 8192 pulses per revolution.

The encoder was connected to the PLC, which published the current value to a PLC topic using the communication bridge described.

We can use the encoder data with corresponding time stamps to calculate the instantaneous angular velocity in Revolutions per minute (RPM) as:

$$V_l = \frac{60\Delta E}{8192\Delta T} \tag{7.5}$$

Where ΔE is the change in output of the encoder and ΔT is a change in time. ΔT can be variable in time as it is triggered by the pulses of the encoder and is calculated using timestamps corresponding to the measurements.

This data was recorded for the five tests mentioned in the feedforward positioning chapter. The velocity calculated from the encoder data can be seen in Figure 7.1

Here we can see that the acceleration and deceleration of the AGV took approximately 1.5s. The middle section, where the motor controller was supposed to follow a velocity setpoint appears to be quite noisy.

It is visible, that the noise appeared during all of the tests. Compared to the other tests, the AGV accelerated more slowly during the first one, this can be caused by incorrect initial rotation of the caster wheels. The second graph different to the others is test 4, where the deceleration started later compared to the other tests. Tests 1 and 4 were also the ones with the largest error, so while this behaviour is hard to predict, it can be measured by the encoder.

For each test, a total number of steps E was taken and divided by the distance measured d_m by tape measure. This gives us the number of steps per meter S travelled by the AGV.

$$S = \frac{E}{d_m} \tag{7.6}$$

We can average the steps per meter and get a number $\bar{S} = 13013.3$

This number can also be obtained by measuring the wheel radius, we will label it as S_c.

$$S_c = \frac{8192}{2r\pi} \tag{7.7}$$

For the 10cm diameter wheel used, the calculated steps per meter S_c are:

$$S_c = \frac{8192}{20.1\pi} \tag{7.8}$$

$$S_c \approx 13038.0 \tag{7.9}$$

Now we can use the average measured steps per meter to estimate the travelled distance (d_e).

$$d_e = E\bar{S} \tag{7.10}$$

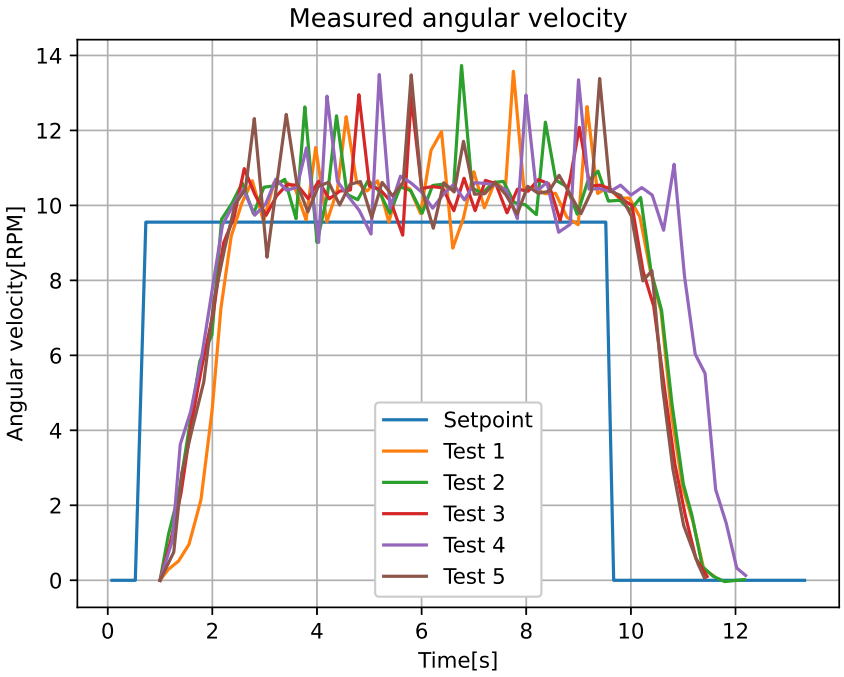


Figure 7.1: Measured angular velocity of the left wheel during tests 1-5

We can compare the two approaches by calculating their mean square error from the measured distance. The setpoint of 100cm was used as an estimation of the feedforward control. The mean square error of the feedforward approach was 20.45, while the Mean Squared Error (MSE) of the encoder estimation was 2.44, indicating that using the encoder significantly improved the precision.

7.3 Right wheel velocity estimation

To estimate the position of the robot, the velocities of both wheels are needed. Since the right wheel didn't have an encoder, the angular velocity of the wheel needs to be estimated. The estimation can be based on the right wheel setpoint and left wheel setpoint and velocity. The structure of this is shown in Diagram 7.2. The mass of the robot's left and right sides is quite balanced and both wheels use the same model of the motor. This allows us to create a model of the observable left wheel system

Test number	Setpoint [cm]	Measured distance [cm]	Amount of steps	Steps per meter	Estimate of traveled distance[cm]
1	100	93	12470	13408.6	95.8
2	100	99	12923	13053.5	99.3
3	100	99	12690	12812.2	97.5
4	100	107	13842	12936.4	106.4
5	100	98	12657	12849.8	97.3

Table 7.1: Results of the tests

and use it for the estimation of the unobservable right wheel.

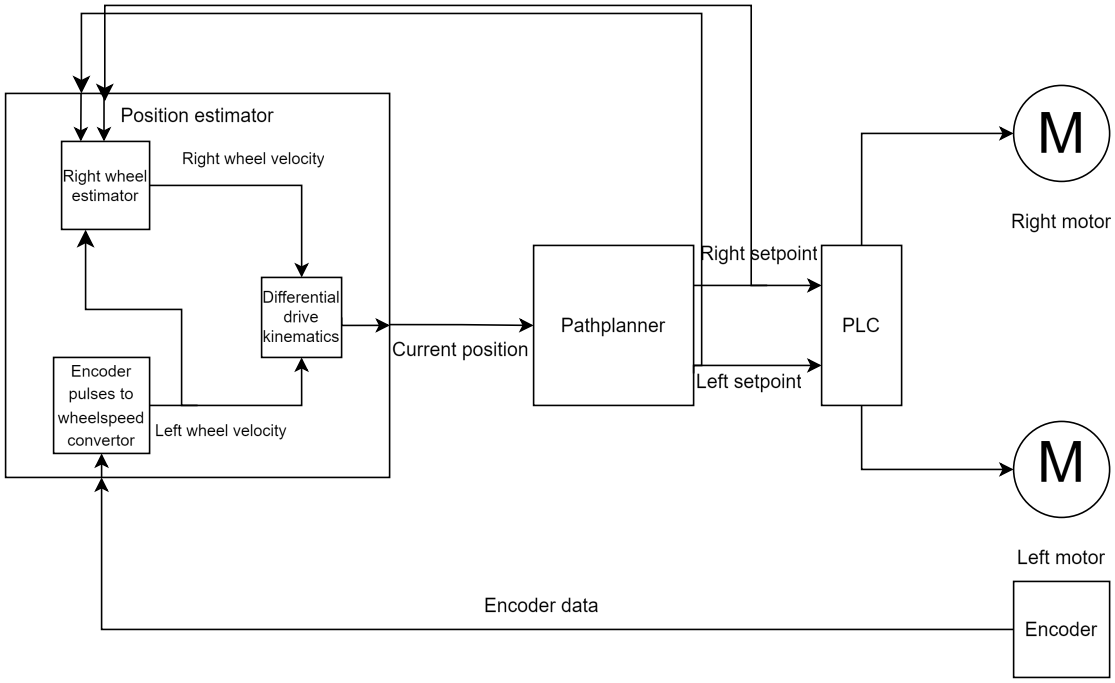


Figure 7.2: Estimation of the right wheel velocity

7.3.1 Proportional estimation

One of the simplest solutions for this is using a proportional system estimator. The output of the modelled system is proportional to the input, so only one parameter is needed to describe this system. The simplification of the real system to zero order one is not detailed enough to model the dynamics of the electrical motor, however, we can use the real-time data to recalculate the model in every control step. The output of the estimator V_r can be calculated from motor setpoints V_{rs} and V_{ls} and velocity of the left wheel V_l

$$V_r = \frac{V_l}{\omega_{ls}} \omega_{rs} \tag{7.11}$$

With a condition setting output to zero when $V_{ls} = 0$ to prevent illegal operation. This approach is very simplified and in some cases can be very inaccurate. For example, accelerating the right wheel while the left wheel is following constant speed could result in unrealistic acceleration. Nevertheless, this approach is very simple to implement and provides very reliable results in two specific cases. The first case is the straightforward (backward) movement, where the motors follow the same setpoint, so the estimator uses just the velocity of the left wheel. The second case is stationary rotation when the setpoints for the two wheels have the same absolute speed but in opposite directions. These two cases, however, are enough to move the AGV into any desired location.

7.3.2 Drive model identification

Proportional estimation can give good results in specific cases, however, to implement the control algorithms from previous chapters, the estimation needs to be more general. In order to model the

dynamics of the drive system, we will use a model with multiple parameters. First, we must select a model structure and find the most suitable set of parameters. The first model chosen is a discrete time-invariant state-space model described by equations

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \tag{7.12}$$

$$\hat{\mathbf{y}}(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}(k) \tag{7.13}$$

In these equations $\mathbf{x}$ symbolises a state vector, $\hat{\mathbf{y}}$ an output of the model, $\mathbf{u}$ an input, $\mathbf{A}$ represent a state matrix and $\mathbf{B}$ an input vector. The matrices $\mathbf{A}$ and $\mathbf{B}$ will be estimated based on measurement. Figure 7.1 shows the measurement we try to model based on the shape of the measurement, we will skip the first-order model and start with a second-order model. Therefore $\mathbf{A}$ will be a 2x2 matrix and $\mathbf{B}$ will be a 2x1 vector.

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \tag{7.14}$$

$$\mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{7.15}$$

To compare the quality of the models, we will need a test that will evaluate the model's ability to describe the observed data. The following method was described by Lennart Ljungs *Theory for the user* [14]

We will start by defining the error ε given by a certain model with a set of parameters $\Phi* = [a_1, a_2, a_3, a_4, b_1, b_2]$ at time step k.

$$\varepsilon(k, \Phi*) = \mathbf{y}(k) - \hat{\mathbf{y}}(k|\Phi*) \tag{7.16}$$

There are multiple possible norms to choose from, we will use the quadratic one.

$$\|\varepsilon\| = \frac{1}{2}\varepsilon^2 \tag{7.17}$$

To compare measured and modelled responses, the data needs to have the same timing, to achieve this, the times were matched using linear interpolation of the discrete measurements. We can obtain a vector of N equally spaced measurements Z_N from the linear interpolation of measured velocities. These measurements have a period T in between them.

$$Z_N = [y(k), u(k), y(2T), u(2T)..., y(NT), u(NT))] \tag{7.18}$$

The problem of finding the proper model parameters $\hat{\theta}_N$ to fit measured data Z_N can be then defined as a minimization problem:

$$\hat{\Phi}_N = \hat{\Phi}_N(Z_N) = \underset{\Phi}{\operatorname{argmin}} \frac{1}{N} \sum_{k=1}^{N} \frac{1}{2}\varepsilon^2(t, \Phi) \tag{7.19}$$

The next step is to use gradient descent optimisation to find this minimum. After gradient descent optimisation we received these parameters of the model.

$$\mathbf{A} = \begin{bmatrix} -0.24 & 0.056 \\ 0.448 & -0.428 \end{bmatrix} \tag{7.20}$$

$$\mathbf{B} = \begin{bmatrix} -0.086 \\ 1.236 \end{bmatrix} \tag{7.21}$$

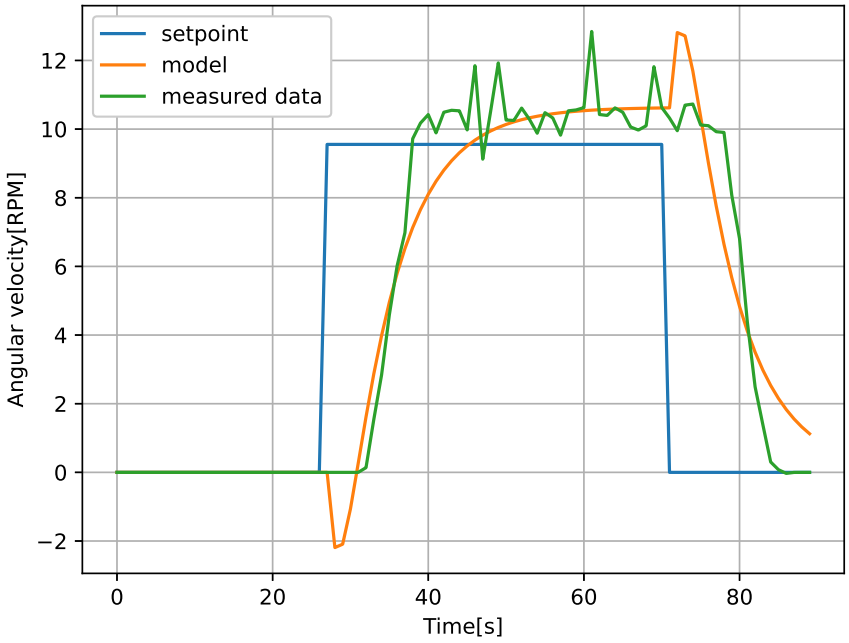


Figure 7.3: Real system and model comparison

The modelled behaviour shown in Figure 7.3 is similar to the measured one, however, the model seems too simplified and has trouble modelling some parts of the robot's behaviour. The approximately 5-step delay isn't modelled at all and is instead compensated by undershooting in the opposite direction relative to the setpoint change.

One possible way to solve this would be to increase the model order, however, this would increase the complexity and training time. A more simple way would be to model only the delay, so it would be necessary to tune only one extra parameter. This was implemented by adding a queue for inputs. By changing the length of the queue, we change the delay of the system. With delay noted as k_d, the model will be described by the following equations

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k-k_d) \tag{7.22}$$

$$\hat{\mathbf{y}}(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}(k) \tag{7.23}$$

The final parameters of the model are the following

$$\mathbf{A} = \begin{bmatrix} -0.368 & 0.09 \\ -0.096 & -0.898 \end{bmatrix} \tag{7.24}$$

$$\mathbf{B} = \begin{bmatrix} 0.68 \\ 0.894 \end{bmatrix} \tag{7.25}$$

$$k_d = 6 \tag{7.26}$$

Test number	Measured	Encoder	Model	Calibrated model
1	93	95.8	95.7	97.5
2	99	99.3	98.0	99.8
3	99	97.5	95.7	97.5
4	107	106.4	102.4	104.3
5	98	97.3	95.8	97.5

Table 7.2: Results of model-based estimation

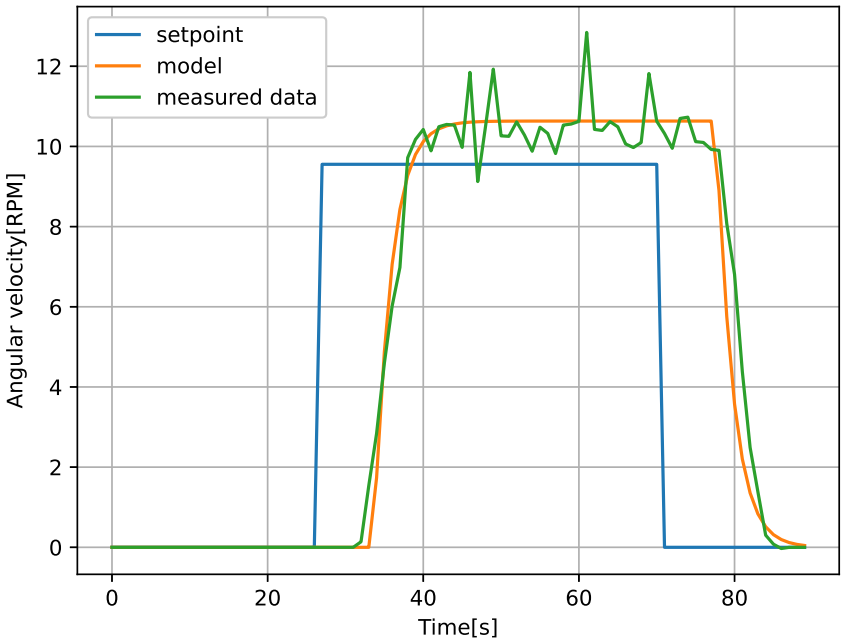


Figure 7.4: Real system and model with delay comparison

This model represented the data with a cumulative error of 0.24. The result can be seen in Figure 7.4. The output of the model follows the measured data quite accurately and does under or overshoot as in the previous part. The model of third order was also tested, but it didn't provide any significant change apart from increasing training time.

Testing on non-training data showed similar results to the training data. However the purpose of this model is to provide an estimation of travelled distance, so the next step is to compare the sum of encoder and model velocities with measured velocity. This comparison shows, that this estimate is significantly worse than the one based on encoder data as it has more than double the average relative error of 2.9% compared to 1.3% of encoder-based estimation. However, this estimation seems biased as it has an average value of only 97.5 cm, compared to 99.3 cm of both tape measurement and encoder estimation. We can try to improve it by multiplying the model estimate by a calibration constant. This mitigates the bias and improves the average relative error to 2.1%.

Chapter 8

Rotation estimation using IMU

One way to increase the precision of localisation is to utilize additional sensors. A common sensor used for localisation is an IMU. It can be obtained as a microelectromechanical system, which allows it to be a small and cheap sensor. In their work, Qilong Yuan and I-Ming Chen [29] explored the tracking of humans using IMU. They were able to localise the running person with a positioning accuracy of around 2% and the walking person with an accuracy of <1%. One of the issues connected to using IMU is the drift. Because of this, the sensor keeps accumulating errors even when it is not moving. This can be mitigated by using different technologies, such as fibre optic gyroscopes, which have much lower drift. However those are more expensive, space demanding and they mitigate the drift of the sensor, but not the drift caused by earth rotation. To deal with this problem, the ZUPT algorithm has been developed [10]. This algorithm resets the velocity to zero when no movement is detected. The detection of no movement is usually done by comparing the IMU value to some predefined threshold. This was tested at the University of Freiburg, where they used the characteristics of human walk. When the IMU is placed on a human foot, it is static during the time between every step when the foot is on the ground. This time can be used to reset the velocity and highly decrease the drift error. With this setup, they declare a maximal error of about 3%. Based on the mentioned works, the relative errors seem similar to the one of the encoders. However, the causes for the errors are different and a combination of those approaches could lead to an improvement of precision. The main drawback of such a setup would nevertheless be the fact, that IMU as well as encoder-based localisation are both relative positioning techniques. Even if the relative error decreased, the absolute error would still tend to accumulate.

One of the advantages of the IMU is the fact that the sensor is already present on the robot. The machine vision algorithms require high-quality visual input to work correctly. Fravebot robots use industrial OAK D Pro cameras which were developed specifically for Computer vision and robotics and contain a BMI270 IMU chip. This will allow us to gather gyroscope and accelerometer data without requiring additional hardware and without increasing the robot's costs. It can provide us with additional measurements of the robot's rotation.

8.1 Data acquisition

To get an overview of the data it is measuring, we started by gathering a measurement of a four-minute experiment, during which the camera was lying on the table, picked up and moved chaotically by hand and laid on the table afterwards. Gathered raw data can be seen in the following Figures 8.1 and 8.2.

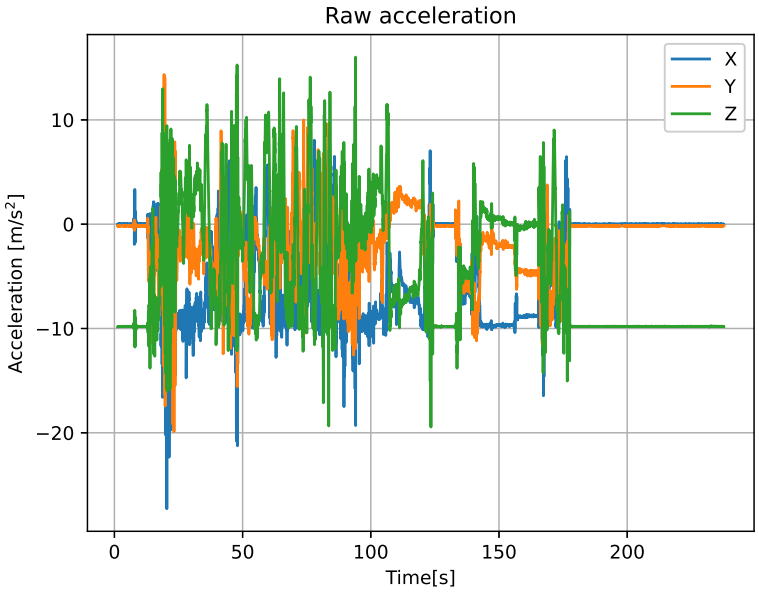


Figure 8.1: Raw output of the IMU accelerometer

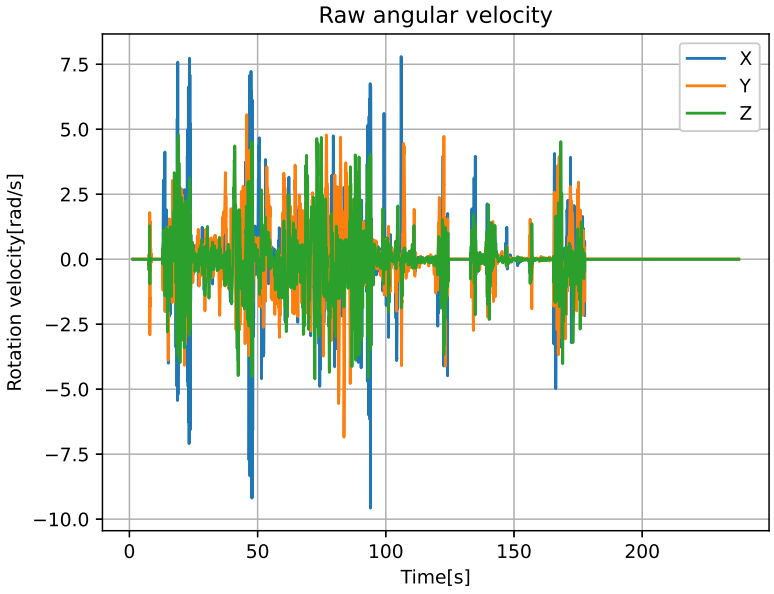


Figure 8.2: Raw output of the IMU gyroscope

These graphs show quite noisy behaviour of the sensor output. There are noticeable intervals of the sensor being static on the table, where the output of both sensors was near zero except for the Z axis of the accelerometer, measuring gravitational acceleration.

The main purpose of this measurement is to get additional sensors measuring the robot's rotation. A common way to obtain rotation from IMU is to use data from accelerometer, magnetometer and gyroscope and fuse them together [2]. The first possible sensor within the IMU is an accelerometer. The tilt angles can be computed from the accelerometer as

$$\begin{bmatrix} \zeta \\ \iota \\ \theta \end{bmatrix} = \begin{bmatrix} \text{arctan2}(a_y, a_z) \\ \text{arctan2}(-a_x, \sqrt{a_y^2 + a_z^2}) \\ 0 \end{bmatrix} \tag{8.1}$$

Where ζ is roll angle, ι is pitch angle and θ is yaw angle. As this rotation estimation is based on a gravity vector, it can be used to estimate roll and pitch, but not yaw. The greenhouse floor is horizontal, so the yaw angle is the only one changing during the movement.

Usually, this is compensated by using a magnetometer for yaw, however, the version of IMU is not equipped with one. Even if we had a measurement of the magnetic field, it would probably be affected by the proximity of the motors of the robot. The only remaining sensor usable for estimation of yaw angle is the gyroscope.

The angles are estimated from the previous rotation using the measurements of angular velocities $\dot{\zeta}$, $\dot{\iota}$ and $\dot{\theta}$ as

$$\begin{bmatrix} \zeta_k \\ \iota_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} \zeta_{k-1} + \dot{\zeta} dt \\ \iota_{k-1} + \dot{\iota} dt \\ \theta_{k-1} + \dot{\theta} dt \end{bmatrix} \tag{8.2}$$

where dt is the time interval between the measurements.

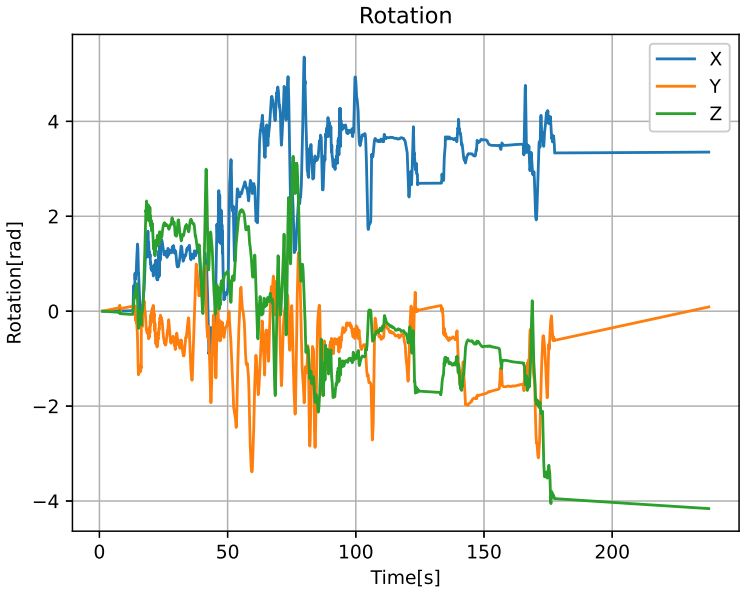


Figure 8.3: Rotation obtained by numerical integration of angular velocity

The estimated rotation is shown in Figure 8.3. We don't have a ground truth reference for this measurement, but the measurement is in the expected range. There is a visible drift of the measured rotation, especially on the Y and Z axis. This drift is present even if the sensor is stationary. In the second measurement, visible in Figure 8.4, we left the sensor on the table without any movement.

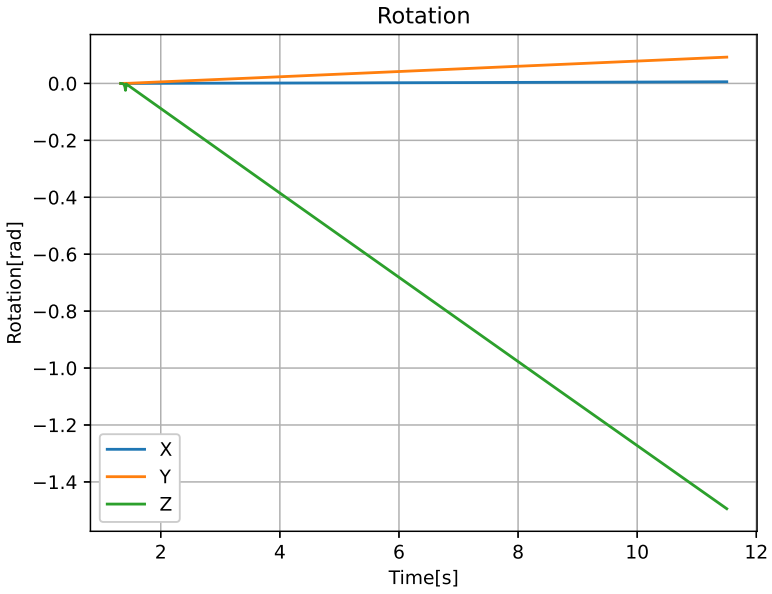


Figure 8.4: Rotation obtained by numerical integration of angular velocity during the second measurement

The drift shows very linear behaviour. We will try to model it and subtract it from the measurement. First, we need to distinguish between the drift and real movement. This can be done by simply modelling the drift when the sensor is stationary and only the drift is present. On the robot, there are many times, when it is not moving and those can be used to calibrate the sensor. For this, we need to decide if the sensor is moving or not automatically. For this, we will calculate a difference between every measurement and a previous one and compare this difference with a predefined threshold. If this difference is lower than the threshold in a certain amount of consecutive steps, it will be considered still. In this case, the number of requested still steps were selected as 50.

```
    def is_still(self, acceleration):
        """Returns whether the sensor is moving or not based on acceleration"""
        dx = abs(acceleration[0]-self.last_acceleration[0])
        dy = abs(acceleration[1]-self.last_acceleration[1])
        dz = abs(acceleration[2]-self.last_acceleration[2])
        self.last_acceleration = acceleration
        if dx+dy+dz > self.STILL_SENSITIVITY:
            self.still_length = 50
        else:
            self.still_length -= 1
        return self.still_length <= 0
```

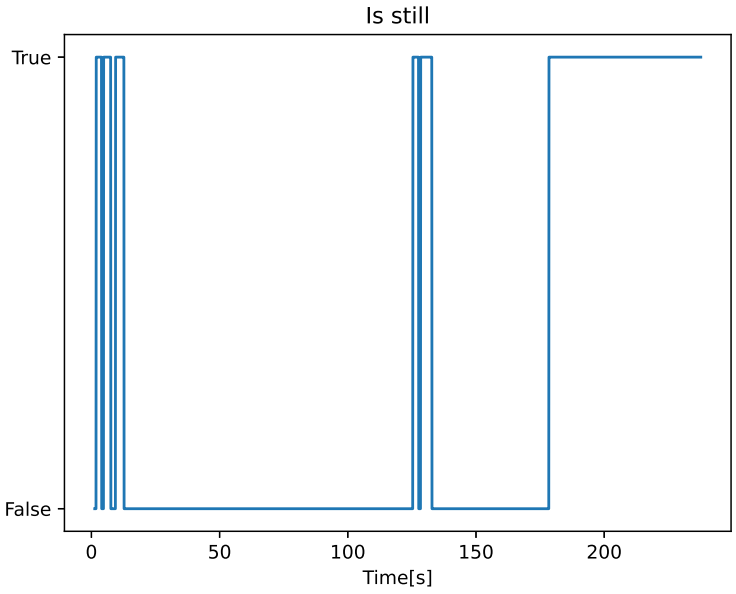


Figure 8.5: Output of the movement detection algorithm

Next, we will use this function to model the drift when the sensor is still. For this, we will use the recurrent least square method with exponential forgetting.

$$\varepsilon_k = \mathbf{y}_k - (\mathbf{y}_{k-1} + \hat{\Phi}_{k-1}) \tag{8.3}$$

$$K_k = \frac{P_{k-1}}{\lambda + P_{k-1}} \tag{8.4}$$

$$P_k = \frac{P_{k-1} - K_k P_{k-1}}{\lambda} \tag{8.5}$$

$$\hat{\Phi}_k = \hat{\Phi}_{k-1} + K_k \varepsilon_k \tag{8.6}$$

We need to estimate just a single parameter, therefore Φ will represent the slope of the drift. We start by calculating the prediction error ε_k as a difference between the measured value y_k and the expected value $y_{k-1} + \hat{\Phi}_{k-1}$. P is the error covariance matrix of the estimate and K is the update gain. For exponential forgetting, we will use the forgetting factor $\lambda = 0.99$

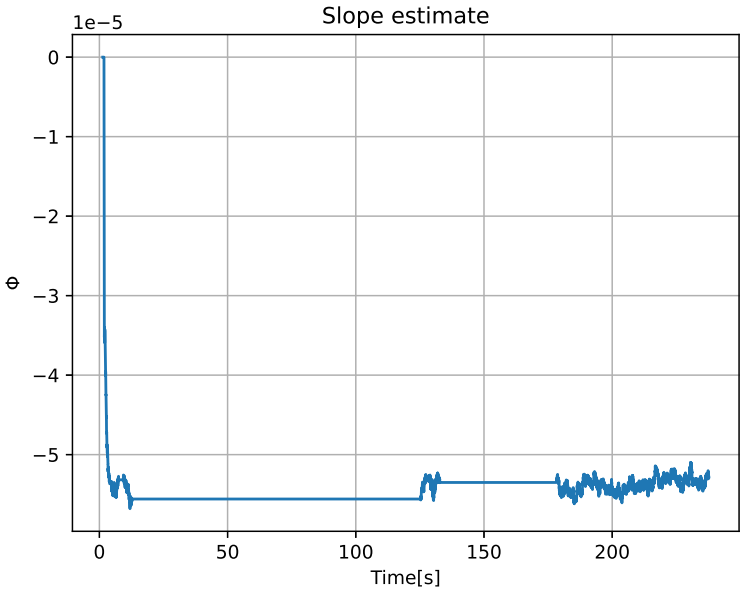


Figure 8.6: Evolution of slope estimate in time

Figure 8.6 shows slope estimate Φ quickly drops to a value where it stabilises. In this case, systems required 3-4 seconds to estimate the drift. Next, we subtract this estimate from the measurement. This will result in the rotation estimate shown in Figure 8.7.

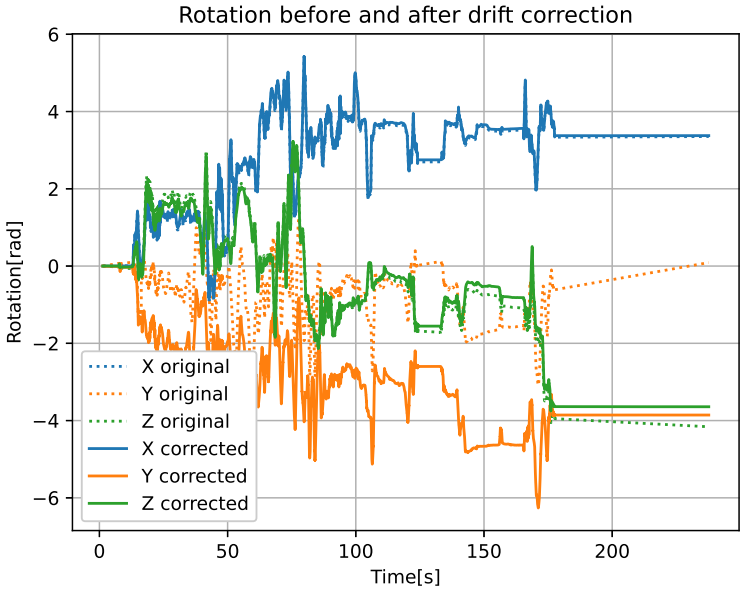


Figure 8.7: Comparison of original and corrected rotation

Test number	Rotation[°]	IMU estimate angle[°]
1	90	91.3
2	90	90.5
3	90	90.8
4	90	89.4
5	90	91.1

Table 8.1: Rotation measurements

Next, we rotated the camera on the desk along a protractor by 90°. This should correspond to the movement of the camera on the robot.

The measurements in Table 8.1 showed, that the sensor can give an interesting estimate of the rotation. The variance of these five measurements was $\sigma^2 = 0.446$ and the relative error was 0.96%. One of the measurements is visible in Figure 8.8.

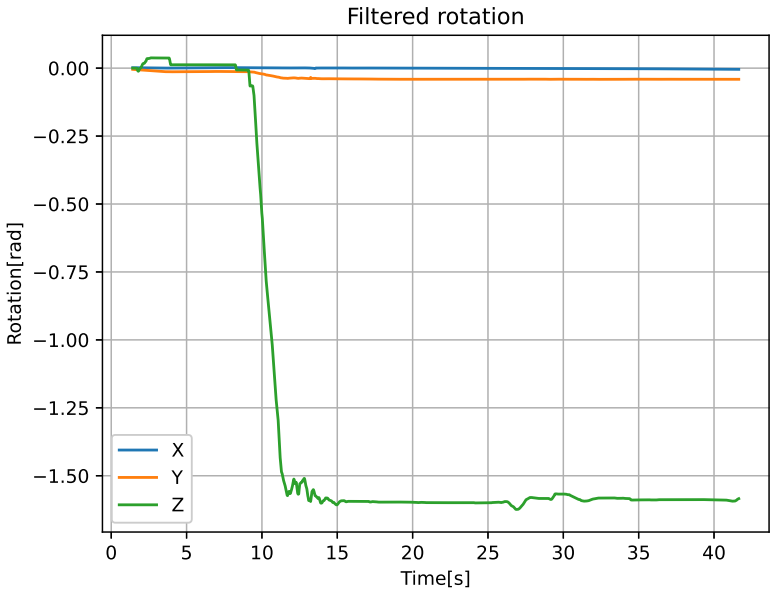


Figure 8.8: Measurement of 90° rotation

The results of these measurements are sufficient to be used together with rotation estimation from the encoder to create a better estimate. The main benefit of IMU rotation estimation is its independence on the current state of the caster wheels, which caused significant disturbance in encoder positioning.

Chapter 9

Conclusion

This work starts with a general introduction to precision agriculture and the history of technological advances in agriculture. The rest of the thesis describes the development and testing of a few subsystems of a robotic solution from the company Fravebot. First, the main hardware components were described with their technical properties. The third chapter describes the communication between IPC and PLC. There were multiple supported protocols, ROS versions 1 and 2 and MQTT. This was done using a custom TIA portal library and a Python bridge node. Chapter four focuses on the control of the two motors running the active wheels. First, the setpoints of robot motion are transformed into wheel rotation velocities using inverse kinematics script, then these setpoints are forwarded to SpeedAxis technology objects which are responsible for the low-level control of the stepper motors. As the AGV was capable of simple movement, it was necessary to develop a high-level control. Chapter Five describes the development process of control sequence planning. This approach combined the path planning and control into one optimisation problem. Solving for just one step ahead proved to be insufficient, so the algorithm was generalised to solve for any chosen number of steps on a receding horizon. Next, a few more improvements are described, such as a suitable choice of the cost ratio between position and rotation, initialisation of the algorithm using already calculated values to decrease the calculation time, automatic increase of a number of steps to prevent the AGV from getting stuck or implementation of more safe end of the rail detection. The next chapter is about testing the developed system in a Siemens NX MCD simulation tool. This software simulates all the physics, while PLCSIM advanced simulates the PLC with control scripts. This simulation allowed testing and debugging of the system before commissioning on the real robot. It already showed that the algorithms are functional, but require precise positioning feedback. The last two chapters describe two different approaches to localisation. Chapter Seven starts with simple feed-forward positioning, next the feedback loop is introduced, however, only one wheel velocity is known and the second one needs to be estimated. A model of the right wheel was developed based on the measured data of the left wheel. The complexity of the model has been increased until the second-order model with delay proved sufficient for our use. The average relative error of those measurements compared to the ground truth was 1.3% for the encoder and 2.1% for the estimation based on the model. The last chapter explores the use of an inertial measurement unit for rotation estimation . The IMU sensor was already present in the camera used, so no additional hardware was necessary. By using a least square method with motion detection, it was possible to model and mitigate the drift error. This thesis described ongoing work on the development of a mobile agricultural robot, which will further continue. Future work will likely involve estimating the entire pose using IMU, rather than just rotation, more localisation techniques, such as visual one, or sensor fusion.

Symbolslist

Sign	Description	Unit
A_x	X coordinate of point A	m
A_y	X coordinate of point A	m
B_x	X coordinate of pose B	m
B_y	X coordinate of pose B	m
E	Encoder output	
G_r	Gear ratio	
J	Cost representing the angular and linear distance the AGV has to travel	
K	Update gain of least square method	
P	Error covariance matrix	
R	Distance between instantaneous centre of curvature and centre of the wheels	m
S_c	Theoretically obtained encoder steps per meter	
S	Encoder steps per meter	
T	Time	s
V_l	Linear velocity of the left wheel along the ground	m/s
V_r	Linear velocity of the right wheel along the ground	m/s
V	Velocity of the AGV	m/s
Φ	Set of model parameters	
α	Angular coordinate of pose A	
β	Angular coordinate of pose B	
γ	Angular distance between two poses	
λ	Forgetting factor of least square method	
$\mathbf{A}$	State matrix	
$\mathbf{B}$	Input vector	
$\hat{\mathbf{y}}$	Modelled output of linear system	
$\mathbf{u}$	Input of linear system	
$\mathbf{x}$	State vector of linear system	
$\mathbf{y}$	Output of linear system	
ω_{for}	Forward component of the angular velocity	RPM
ω_{ls}	Velocity setpoint for left wheel	RPM
ω_l	Angular velocity of left wheel	RPM
ω_{rot}	Angular component of wheel angular velocity	m
ω_{rs}	Velocity setpoint for right wheel	RPM
ω_r	Angular velocity of right wheel	RPM
π	Ratio of a circle's circumference to its diameter	
σ	Standard deviation	
θ	Angular coordinate on the greenhouse map, zero on AGV facing right	rad

Sign	Description	Unit
ε	Model error	
d_m	Distance measured by a tape measure	m
k_d	Modelled delay	
l	AGV track width	m
r	Radius of the AGVs wheel	m
x	Horizontal coordinate on the greenhouse map	m
y	Vertical coordinate on the greenhouse map	m

Bibliography

[1] A. Barrientos et al. "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots". In: *J. Field Robot.* 28 (2011), pp. 667–689. DOI: 10.1002/rob.20384.

[2] *Complemetary Filter*. URL: https://ahrs.readthedocs.io/en/latest/filters/complementary.html.

[3] J. Del Cerro et al. "Aerial Fleet in RHEA Project: A High Vantage Point Contributions to ROBOT 2013". In: *ROBOT 2013*. Ed. by M. A. Armada, A. Sanfeliu, and M. Ferre. Vol. 252. Advances in Intelligent Systems and Computing. Cham, Switzerland: Springer, 2014. ISBN: 978-3-319-03412-6.

[4] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.

[5] L. Emmi et al. "Fleets of Robots for Precision Agriculture: A Simulation Environment". In: *Ind. Robot Int. J.* 40 (2013), pp. 41–58.

[6] *Employment in agriculture over the long-run*. URL: https://ourworldindata.org/employment-in-agriculture#employment-in-agriculture-over-the-long-run (visited on 08/03/2023).

[7] Taha Fadhaeel et al. "Design and development an Agriculture robot for Seed sowing, Water spray and Fertigation". In: *2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*. IEEE. 2022, pp. 148–153.

[8] *Greenhouse trolley*. URL: https://www.steenks-service.com/pipe-rail-trolleys/greenhouse-trolley/ (visited on 06/27/2023).

[9] *Heating systems*. URL: https://www.horconex.com/en/solutions/heating-systems (visited on 06/27/2023).

[10] Fabian Höflinger, Rui Zhang, and Leonhard M. Reindl. "Indoor-localization system using a Micro-Inertial Measurement Unit (IMU)". In: *2012 European Frequency and Time Forum*. 2012, pp. 443–447. DOI: 10.1109/EFTF.2012.6502421.

[11] *Invention of Cotton Gin*. URL: https://ehistory.osu.edu/articles/invention-cotton-gin (visited on 10/16/2023).

[12] A. Janani, L. Alboul, and J. Penders. "Multi Robot Cooperative Area Coverage, Case Study: Spraying". In: *Lecture Notes in Computer Science*. Vol. 9716. Berlin/Heidelberg, Germany: Springer, 2016, pp. 165–176. ISBN: 978-3319403786.

[13] *Jetson Orin Modules and Developer Kits*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ (visited on 08/17/2023).

[14] Lennart Ljung et al. "Theory for the user". In: *System identification* (1987).

[15] Steven Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[16] *MQTT:THe Standard for IoT Messaging*. URL: https://mqtt.org/ (visited on 10/30/2023).

[17] N. Noguchi and O.C. Barawid. "Robot Farming System Using Multiple Robot Tractors in Japan Agriculture". In: *IFAC Proc. Vol.* 44 (2011), pp. 633–637.

[18] *PLC Programming with SIMATIC STEP7*. URL: https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal/software/step7-tia-portal.html (visited on 07/03/2023).

[19] Andrew R Plummer. "Model-in-the-loop testing". In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 220.3 (2006), pp. 183–199.

[20] *Precision Ag Definition*. URL: https://www.ispag.org/about/definition (visited on 10/16/2023).

[21] *Precision Agriculture*. URL: https://www.grap.udl.cat/en/presentacio/que-fem/agricultura-de-precisio/ (visited on 10/16/2023).

[22] *Product REM-E-106C15E-9E43B-B3M12*. URL: https://www.turck.de/en/product/100044357 (visited on 09/05/2023).

[23] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015. ISBN: 978-1-4493-2389-9.

[24] *rosbag*. URL: http://wiki.ros.org/rosbag (visited on 10/30/2023).

[25] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[26] *SiePortal*. URL: https://mall.industry.siemens.com/ (visited on 06/27/2023).

[27] *snap7*. URL: https://snap7.sourceforge.net/.

[28] *SOLUTIONS FOR SMART GREENHOUSES*. URL: https://www.fravebot.com/ (visited on 07/10/2023).

[29] Qilong Yuan and I-Ming Chen. "Localization and velocity tracking of human via 3 IMU sensors". In: *Sensors and Actuators A: Physical* 212 (2014), pp. 25–33. ISSN: 0924-4247. DOI: https://doi.org/10.1016/j.sna.2014.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0924424714001204.

[30] Jingjuan Zhao et al. "Global agricultural robotics research and development: Trend forecasts". In: *Journal of Physics: Conference Series*. Vol. 1693. 1. IOP Publishing. 2020, p. 012227.