HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

Highly Autonomous Indoor Assistant

Ausgeführt im Schuljahr 2018/19 von:

Kartograpierung und Navigation
Alexander Lampalzer 5BHIF

Tests, Monitoring und Object Detection
Simon Königsreiter 5BHIF

Betreuer / Betreuerin:

MMag. Dr. Michael Stifter

Wiener Neustadt, am 4th April, 2019

Abgabevermerk: Übernommen von:

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wiener Neustadt, am 4th April, 2019

Verfasser / Verfasserinnen:

Alexander Lampalzer	Simon Königsreiter

Contents

Foreword

0.1 Königsreiter Simon

For this diploma thesis I want to especially thank Stifter Michael and Lampalzer Alexander for helping me and keeping me motivated throughout this thesis. Mister Stifter also has my greatest respect as he is one of the most competent and kind project owners I have ever encountered in my school life.

I also want to thank my best friends Prinz Andreas, Homolka Nicolas, Paul Gregor and Bulut Yunus for being there for me when school got stressful.

And I also want to thank my gaming buddies Lampalzer Alexander, Lampalzer Konstantin and Klimont Joel for fun and relaxing gamming nights. Furthermore I want to thank Reichl Christoph, Nagy Lukas, Glavanits Marcel, Stavarache Mario, Killer Lorenz and Fischbacher Berndt for being awesom classmates.

And I want to thank my cat Amy for being the most awesome and cute cat imaginable.

0.2 Alexander Lampalzer

A lot of effort was put into this diploma thesis - first and foremost I would like to especially thank Dr. Michael Stifter for providing support, feedback and motivation throughout the whole process. Furthermore, I would like to give a special thanks to my family, for providing food and shelter. Last, but not least I am grateful for this opportunity, that arose through the workings of F-WuTS.

Kurzfassung

Das Gebiet der Robotik expandiert rasch und Spezialisten, die autonomes Mapping, autonome Navigation und Sensorfusion durchführen können, sind äußerst gefragt. Die zugrundeliegenden Algorithmen, wie Kalman-Filter und die grafisch basierte simultane Lokalisierung und Kartengenerierung, sind jedoch nicht auf dieses Gebiet beschränkt, sondern finden in einer Vielzahl von Anwendungen ihren Einsatz. Diese Algorithmen werden von "primitiven" Staubsaugrobotern, aber auch von großen Passagierflugzeugen und sogar in Satelliten verwendet.

Ziel dieser Diplomarbeit ist es, die grundlegenden Konzepte mobiler, bodengestützter Roboter zu erforschen und eine Plattform zu entwickeln, auf der zukünftige Studenten ihre Projekte aufbauen können. Insbesondere für Anfänger kann es sehr zeitaufwändig sein, sich das nötige Wissen anzueignen. Der Bau eines Roboters mit den erforderlichen Komponenten, Leistungseigenschaften und Batterielebensdauer ist ein komplizierter und zeitaufwändiger Prozess. Aus diesem Grund besteht ein weiteres Ziel darin, eine Vielzahl von Sensoren zu kaufen, zu testen und zu montieren und einen autonom navigierenden Roboter zu entwickeln, der einen Bereich kartographiert und Objekterkennung zum Suchen und Beschriften von Objekten verwendet.

Durch den Aufbau und die Erweiterung des ROS-Ökosystems konnten die Autoren eine Reihe modernster Algorithmen auswerten und den Benutzern die Möglichkeit geben, zu wählen, was für ihre jeweilige Situation das Richtige ist. Darüber hinaus wurde der AUT-AS-Roboter mit einer Vielzahl von kostengünstigen Sensoren ausgestattet und deren Funktionalität, Leistung und Grenzen analysiert. Durch die Integration mit dem Gazebo Simulator können die Autoren und zukünftige Projekte die Abhängigkeit von einem physischen Roboter reduzieren. Schließlich werden den zukünftigen Studenten umfangreiche Dokumentationen und öffentlich zugänglicher Quellcode und Beispielanwendungen zur Verfügung gestellt. All dies führt zu einer voll funktionsfähigen Plattform, auf der zukünftige Studenten ihre Ideen testen und anderen Institutionen die Möglichkeit geben können, das, was die Autoren entwickelt haben, auf kosteneffiziente Weise neu aufzubauen.

Abstract

The field of robotics is rapidly expanding and specialists capable of performing autonomous mapping, autonomous navigation and sensor fusion are in extremely high demand. The underlying algorithms, such as Kalman filters and graph-based simultaneous localization and mapping however, are not restricted to this field, but rather find purpose in a variety of applications, ranging from "primitive" vacuum cleaning robots to large passenger aircraft and even space satellites.

This diploma thesis aims to explore the fundamental concepts of mobile, ground based robots and further develop a platform upon which future students can build their projects on. Especially for beginners, it can be very time-consuming to gain the required knowledge, that is necessary. Further, building a robot, that has the required components, performance characteristics and battery life, is a complicated and time consuming process. For this reason, another aim is to purchase, test and mount a variety of sensors, and develop an autonomously navigating robot, that maps an area and uses object recognition to search for and label, objects.

By building upon and expanding the ROS ecosystem, the authors were able to evaluate a range of state-of-the-art algorithms and give users the possibility to choose, what is right for their particular situation. Moreover, the AUT-AS robot was equipped with a wide variety of low-cost sensors and the functionality, performance and limits of these were analyzed. Furthermore, by integrating with the gazebo simulators, the authors, and future projects, are able to reduce the reliance on a physical robot. Finally, extensive documentation and publicly available source code and example applications are provided to future students. All this results in a fully functional platform, that lets future students test their ideas and gives other institutions the ability, to re-build what the authors have developed, in a cost-efficient manner.

Chapter 1

Introduction

Author: Alexander Lampalzer

Since the beginning of the 21st century, the amount of computational power in computers has dramatically increased, which empowered many fields in computer science like robotics. It has allowed for more efficient and much more complex tasks to be automated by robots in a wide variety of different environments, such as the automotive industry. Furthermore, the vast amount of openly available information and software in this field has allowed newcomers to get started more easily and quickly.

1.1 Aim

This diploma thesis aims to show the challenges and opportunities encountered by the authors when developing a mobile robot and also aims to act as a guide for younger students to find inspiration and ideas for their own projects. The aforementioned robot, described in this thesis, is equipped with several sensors, like depth cameras and 2D distance measurement sensors, and programmed to perform autonomous navigation and 3D mapping of a, for the robot, unknown environment. Additionally, a comprehensive platform for managing, administrating and monitoring a fleet of autonomous robots is developed.

1.2 Features

If the mobile robot and the corresponding application are successfully developed, the following features must be implemented.
Robot:

- Autonomous 3D mapping of a previously unknown environment
- Autonomous navigation between different areas on the same ground level
- Creation of application specific maps (e.g.: Wifi strength, Temperature, etc.)
- Simulation of the mobile robot in a virtual, customizable environment

Application:

- Teleoperation of a selected robot
- Live view of robots sight
- Monitoring different components of one or more robots
- Administration of one or more robots

Based on the features stated above, the following use cases can be deducted:

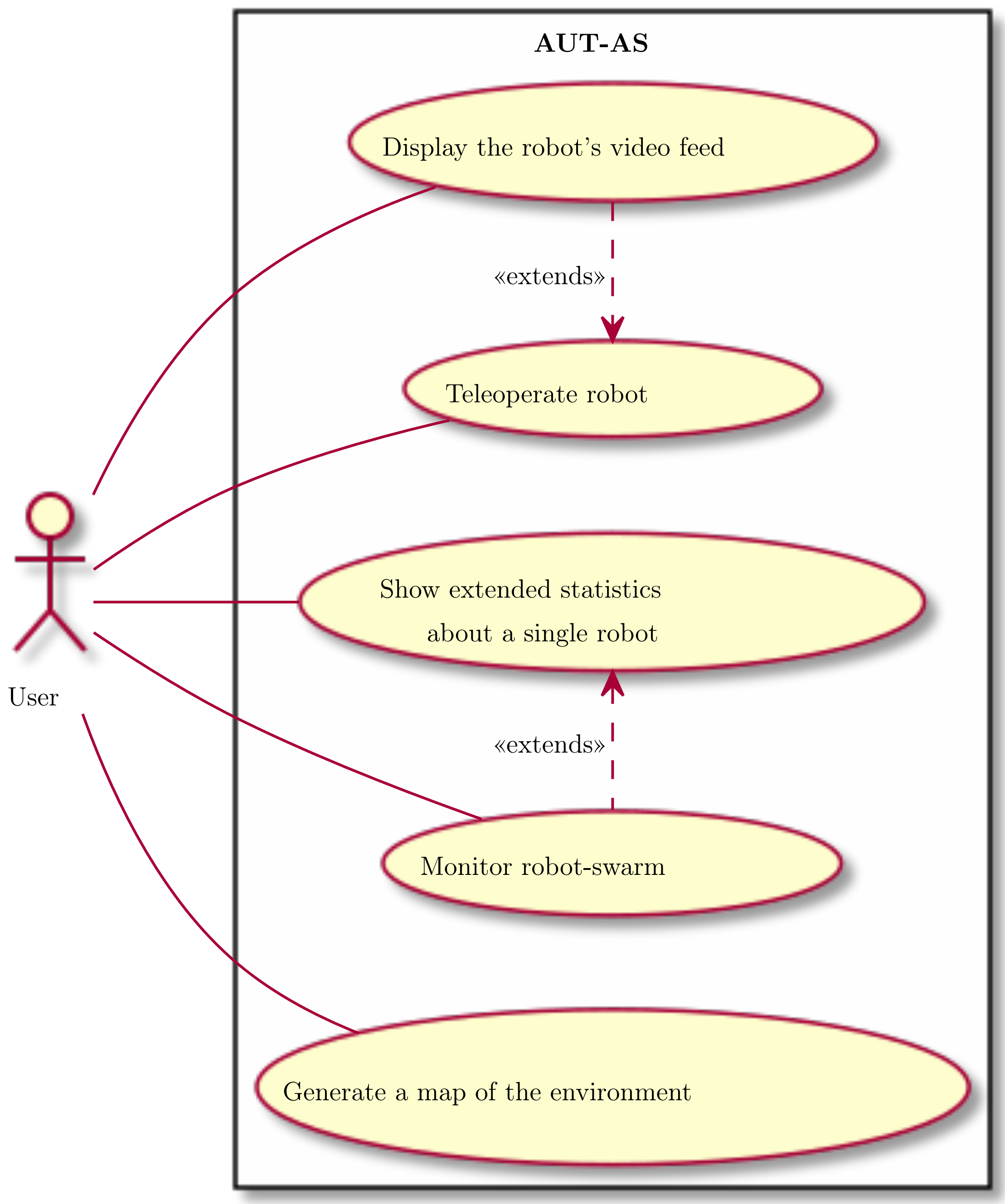
AUT-AS
Display the robot's video feed
«extends»
Teleoperate robot
Show extended statistics about a single robot
«extends»
Monitor robot-swarm
Generate a map of the environment
User

Figure 1.1: Use case diagram describing the core use cases.

1.3 The current state of robotics

According to an article published by the International Data Corporation (IDC) in 2017 the spending in the robotics sector is going to increase rapidly. The expected spending on robotics hard-, software and services is projected to reach \$230.7 billion in 2021 whereas the spending in 2017 summed up to \$97.2 billion which is an increase of 137% in four years. This increase of market potential will generate new jobs which need to be filled by qualified and educated employees.[1]

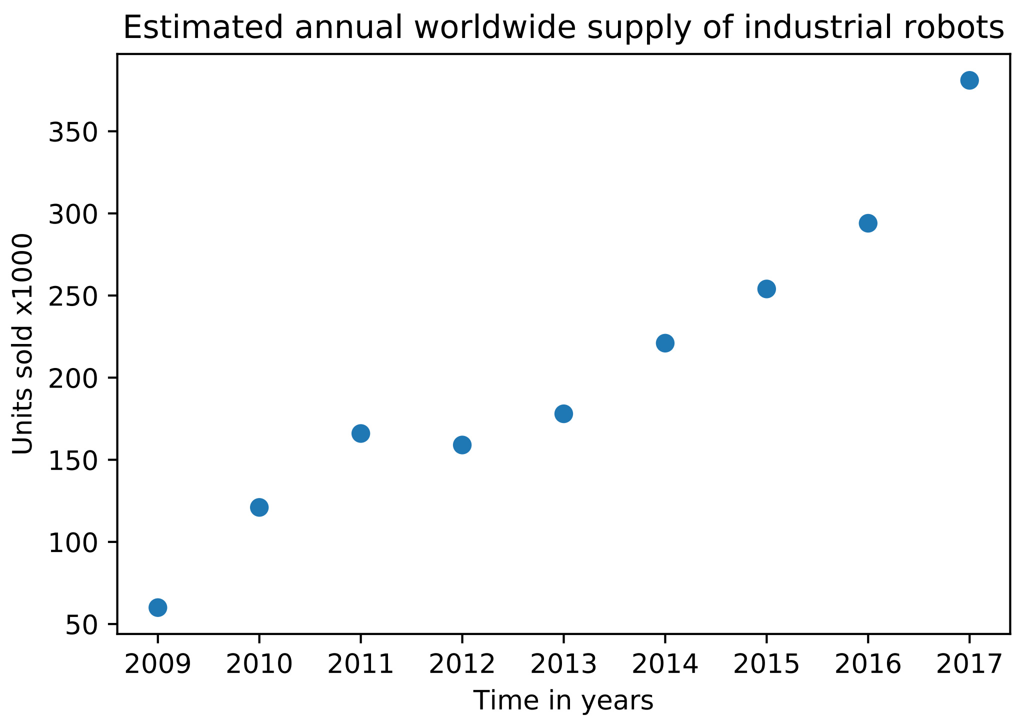

Figure 1.2: A diagram showing the worldwide supplies of industrial robots. The x-Axis represents time in years and the y-Axis represents 1000 units sold.[2]

1.4 The current state of open source ground-based mobile robots

The biggest platform for open source software, Github, contains approximately 73,000 public repositories concerning the topic of robotics. This means that there is a vast amount of hard- and software available. While most of these repositories contain functional pieces of code or hardware, they rarely document the process of the design or development, which makes it more difficult for students to get started in the broad field of robotics. One major framework used for developing robotic systems has established itself as the de facto standard for educational robotics. The so called Robot Operating System (ROS) started out as multiple smaller frameworks at Stanford University in May 2007, with the first official release of ROS in early 2010. Over the years, ROS has gained a several thousand worldwide users, ranging from the industrial sector to hobbyists.[3]

[1]Goepfert and Shirer, *Worldwide Spending on Robotics Forecast to Accelerate Over the Next Five Years, Reaching $230.7 Billion in 2021, According to New IDC Spending Guide*.

[3]Goepfert and Shirer, *Worldwide Spending on Robotics Forecast to Accelerate Over the Next Five Years, Reaching $230.7 Billion in 2021, According to New IDC Spending Guide*.

1.5 Application of ground-based mobile robots

Ground-based mobile robots can execute a variety of tasks. From simple household utility functions like cleaning to mobility like transport of various goods or even people. To perform these complicated tasks every robot needs at least a simple type of navigation. Beside navigation, every robot also needs sensors to function properly. Utilizing sensors for navigation provides robots with the tools necessary to support humans in their everyday life.
Navigation is also a vital component for industrial robotics. The navigation can be used in a variety of use cases ranging from the transport of heavy goods to a drop-off point or movement of robot arms in an assembly line.
Therefor navigation is one of the most vital parts of every robotics system and an also be one of the most important components.

1.6 The connection of robots and smartphones

Since the beginning of the millennium there has been a rapid increase of acceptance of smart-phones. As smartphones are now commonly available in wealthy countries it makes a good opportunity to combine smartphones with robots because both systems have the potential to support humans in their everyday life. With the rise of stable 4G connections throughout Europe it is also possible to transfer large amount of data over the air. This enables high quality live video streaming outside of the connection radius of high speed Wi-Fi.

1.7 Use of robots in the military sector

Author: Königsreiter Simon
Robots have been in use in the military sector for at least 40 Years. Certainly very important in the support of humans has been the Wheelbarrow bomb disposal robot. However one of the biggest problems of the Wheelbarrow is that it is solely remote controlled. Especially in the military sector the autonomous exploration of a previously unknown area provides a great opportunity.[4][5]

1.8 The use of robots in the medical industry

Robots have great potential in supporting humans in medical tasks. Highly complex systems like surgery robots are already in use around the world to assist surgeons at highly complex tasks. But there are other types of robots like telepresence robots which enable medical experts to be present at a location without physically being at that location. This gives them the chance to get advanced insight on the remote medical situation.[7]

1.9 Personal assistance robots

Robots are not only useful in military and industrial application but have also found their way into modern homes. These robots assist humans in their everyday life by reducing the

[4] Smith, "Calls to honour inventor of bomb disposal device".
[5] Allison, *What does a bomb disposal robot actually do?*
[6] British Army, *Remotely_controlled_bomb_disposal_tool.JPG (JPEG Image, 1024 × 768 pixels)*.
[7] University of Stanford, *Robotic Nurses | Computers and Robots*.

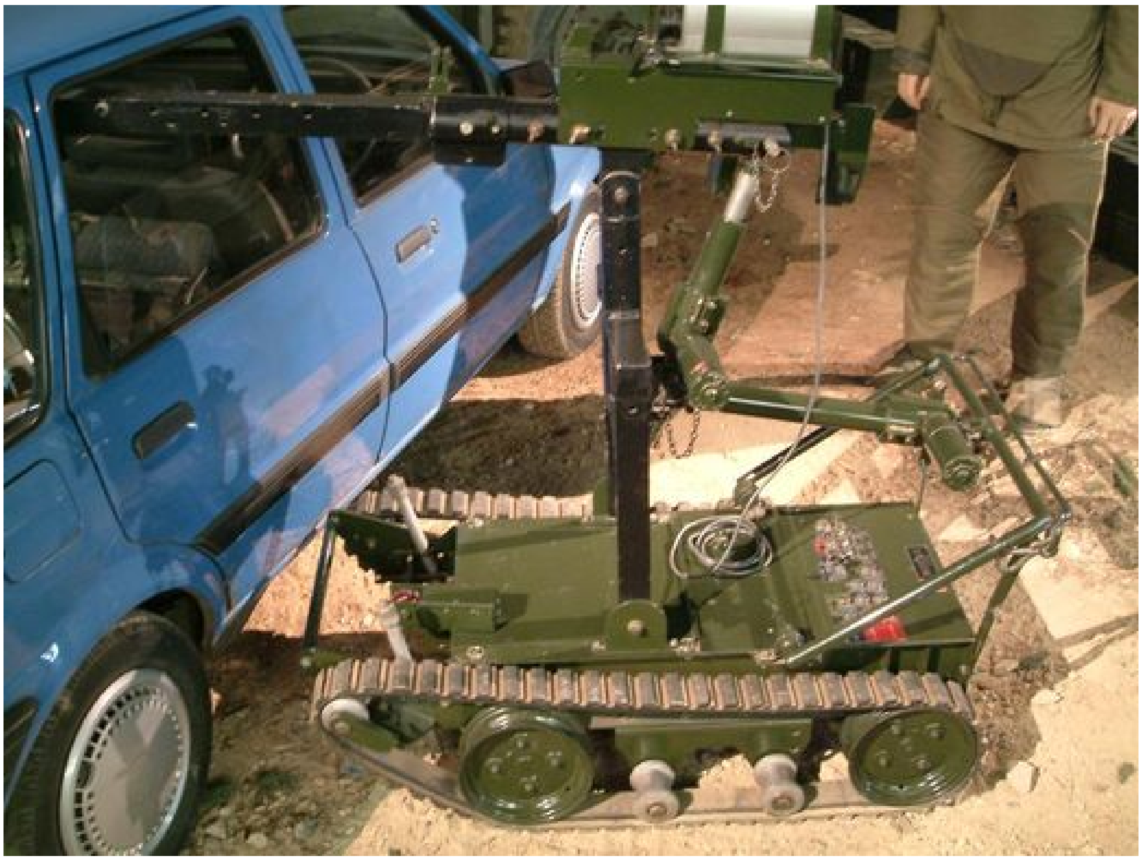

Figure 1.3: An early version of the Wheelbarrow bot with attached remote control cables[6].

amount of work needed to maintain a house. These modern assistants are capable of doing a variety of tasks all around the house like cleaning floors[8] or automated lawn mowing. There are also robots that can help elderly people in their everyday life. They help them by carrying the person from one position to another without human intervention or it acts as a support to help the elderly stand up.[9]

[8]iRobot, *Robot Vacuuming, Robot Mopping & Outdoor Maintenance | iRobot Online Store*.
[9]University of Stanford, *Robotic Nurses | Computers and Robots*.

Chapter 2

Project Management

Author: Königsreiter Simon

Larger project always establish some form of organization to streamline the process for project team members and produce the best possible result. In recent years there has been a massive interest in agile project management methodologies.

2.1 Traditional project management methods

Traditional project management methods are already established and find their application in a lot of domain areas from large finance institutes to architecture projects. These management methods are characterized by a big planing phase and analysis phase in the beginning in the project and an implementation phase which is not very responsive to changed requirements from the client[1].

2.2 Agile project management methods

Agile project management methods have seen an uprise in recent years, especially in the area of software development. Agile methods take an iterative approach on project management in contrast to the already established traditional methods. This enables them to react fast to changed requirements form the client. Agile methods in general follow the agile manifesto which has the following statements at its core[2]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

2.3 Kanban

The team of the project has decided to use Kanban as the project management method.

2.3.1 History of Kanban

Kanban dates back to around 1940 where the car manufacturer Toyota wanted to optimize their car production. Therefor they implemented a system for their factories where each

[1]Project Management Institute, *Don't throw the baby out with the bathwater*.

[2]Agile Alliance, *Agile Manifesto for Software Development*.

team would only request a specific resource as soon as they would run out of it to minimize excessive stock hoarding. While the *"Kanban"* was originally just a sheet of paper, it has been adapted to the 21st century as a mean to help developers build software.

2.3.2 Kanban board

The Kanban board is an integral part in Kanban. The board consists of multiple lanes where each lane represents a state of a task. Each lane has a specific *Work in Progress (WIP)* number which specifies the maximum amount of tasks that can be handled by the lane. If a lane has reached its WIP the previous lanes have to stop their work and help the blocking lane to clear their tasks. This has the big advantage over other agile methods like scrum that the work is continuous and not divided into fixed time frames like in scrum[3].

2.3.3 Setup for AUT-AS

The project used the following lanes and WIPs:

Name	Backlog	Planning	In Progress	Testing
Maximum of WIP	∞	2	2	2

2.4 Meetings with the client

Meetings with the client occurred weekly on every Friday except on holidays. This enabled to team to discuss changes with the project partner and identify possible problems within the team or the project.

[3]Atlassian, *Kanban - A brief introduction*.

Chapter 3

Overview

Author: Alexander Lampalzer

This chapter aims to give a rough understanding of the software and hardware components of the AUT-AS project. Furthermore, it explains some concerns, that have a decisive influence on how this architecture came to be.

3.1 Data Locality & Processing

Data locality, the minimization of data transfers to reduce network load and the choice of right components for data processing, is been one major concern throughout the whole diploma thesis. This aspect is explained in more detail in section 3.3, which is concerned with where and how much performance is needed. Generally speaking, there are three tiers of computing units: Microprocessors, Embedded Processors (FPGAs) and Application Specific Integrated Circuits (ASICs). These tiers are ranked after their specificity to certain tasks, with ASICs being designed for only one purpose, hence the name and microprocessors being capable of general computation tasks. This differentiation can be observed in a multitude of robots. For example, the PR2 robot by willow garage, which is designed for education purposes, contains multiple servers with two quad core Intel Xenon processors[1]. In contrast, the Xiaomi Mi vacuum cleaning robot is equipped with three embedded ARM processors. One vital concern for robot manufacturers and developers alike is battery life and it should be obvious, that more capable processors need more power and thus reduce battery life. Reallocating certain processes to remote computers, such as servers or the cloud is one possible approach, however this comes at the cost of more latency, which might not be optimal for certain types of applications.

Three different approaches are tested on the AUT-AS robot.

- The robotino robot, as is used in AUT-AS, comes with an embedded Intel Atom 1.8$\dot{G}$Hz dual core processor. This approach is constrained by only allowing data processing to happen on the integrated processor. The advantage of this approach is, that there is no need for any additional computers. However, it has been found, that the processing power available is insufficient and battery life is significantly reduced.
- In this second approach, a networked computer is responsible of data storage and processing. It has become apparent fairly quickly, that the introduced latency makes it impossible to navigate and perform SLAM at the same time. Furthermore, the connection to this networked computer is limited by the WIFI range.
- This third and most viable approach combines the previous two attempts. A laptop

[1] **willow_garage_pr2_nodate** .

is mounted on the robot and both the sensors and the robot are directly connected with this laptop via Ethernet and USB. This has significant advantages: First of all, there is an abundance of processing power available and with the additional battery provided by the laptop, the overall battery life is increased significantly.

3.2 Time

It is of vital importance in every multi-machine robotics system, that the computer clocks of all networked computers are synchronized - the systems responsible of data processing need to know when measurements have been taken and results have been computed, in order to guarantee the correct output. Common effects of not having a synchronized time are timeouts, because components often only allow a small margin between the time of recording and the time of processing. When messages appear to be too old, when in reality they are as good as new, errors will happen and it is often hard to exactly localize these sorts of errors. In the AUT-AS project chrony is utilized, because of the advantages it provides in isolated networks[2]. The NTP implementation by chrony is commonly used in ROS projects.

3.3 Construction

The AUT-AS robot utilizes a variety of sensors, a more in-depth analysis can be found in Chapter 5 of this diploma thesis.

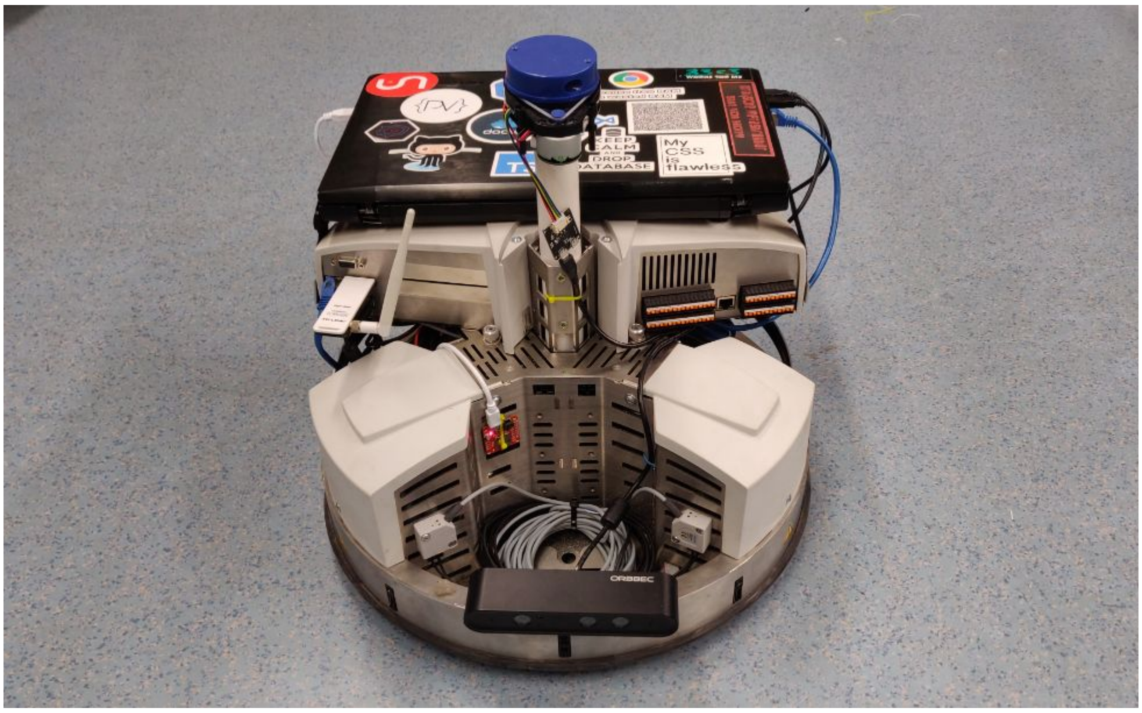

Figure 3.1: This is a picture of the finished construction of the AUT-AS robot. As a base, the Festo Robotino is used and equipped with a collection of sensors.

Following adjustments have been made to the stock robotino robot:

[2]The chrony authors, *Comparison of NTP implementations*.

- A ydlidar x4[3] LIDAR sensor is installed at the top, center of the robot with a 360° field of view, for use in SLAM (See Chapter 7). Due to a low budget, the specifications of several entry-level LIDAR sensors were compared and it was concluded, that this sensors is the most viable option, as of November 2018.
- At the front, center of the robot an Orbbec Astra Pro depth camera is installed[4]. With a camera resolution of 1080p and a depth resolution of 640p, this sensor is viable option of utilization in object detection and localization, and obstacle avoidance.
- For external communication, a 2.4ĠHz WIFI antenna is installed. In the future, this could eventually be replaced with an LTE or even 5G connection.
- As has already been stated in the previous chapter, the AUT-AS robot uses an external laptop for data processing of ingested data. It can be seen at the top of the picture.
- In order to improve odometry, a Sparkfun Razor 9DOF interial measurement unit is installed between the LIDAR and the depth camera.

[3] *YDLIDAR - X4*.
[4] Orbbec 3D, *Astra Series*.

Chapter 4

Robot Operating System

This chapter aims to describe the architecture of the Robot Operating System (ROS) and explains how the AUT-AS robot utilities this framework to achieve its goals. ROS is an open-source framework primarily designed to simplify the development, testing and monitoring of robotics applications. It provides a vast amount of tooling to abstract robotics hardware from the implementation. The ROS has a modular architecture, which allows developers to easily replace different components of a robotics application and mock certain parts of the system. Simulators, like the gazebo simulator, utilize this to provide tooling for simulating complete robots, including sensor data and sensor noise.[1] Especially in the academic area, the open source project has gained a lot of traction which results in a big group of regular contributors and ready to use software packages.[2]

Author: Alexander Lampalzer

The aim of this chapter is to describe the Robot Operating System's (ROS) architecture and explain how AUT-AS utilizes this framework to achieve its goals. First of all, ROS is an open-source framework primarily designed to simplify the development, testing and monitoring of robotics applications. It provides a vast amount of tooling to abstract robotics hardware from the implementation. By building upon a modular architecture, which allows developers to easily switch out different components of a robotics application and mock certain parts of the system, ROS is able to easily integrate with third parts software. Simulators, like the gazebo simulator, utilize this to provide tooling for simulating robotics systems, including sensor data and sensor noise.[3] Especially in the academic area, the open source project has gained a lot of traction, which results in a big group of regular contributors and a vast amoun of ready to use software packages.[4]

4.1 Architecture

Communication in the Robot Operating System is based upon a mesh architecture, where various nodes communicate with each other by publishing and subscribing to certain topics. On each topic, only one type of message can be used, such as "LaserScan" messages (typically published by LIDARs), "Odometry" messages (e.g.: result of motor encoders) or "Cmd2Vel" (controlling a robot's velocity). Additionally, it is also possible to define custom messages. ROS builds upon four basic building blocks. These are described in separate subsections below.

[1]Open Source Robotics Foundation, *Gazebo : Sensor Noise*.
[2]Open Source Robotics Foundation, *ROS packages*.
[3]Open Source Robotics Foundation, *Gazebo : Sensor Noise*.
[4]Open Source Robotics Foundation, *ROS packages*.

4.1.1 Node

ROS builds upon the UNIX philosophy which states that one program should do exactly one thing well and that it should be capable to work together with other programs. Thereby the UNIX philosophy chose a bottom-up strategy in favour of the top-down strategy[5]. ROS builds upon this guideline by building applications with nodes. A node is a process which performs a specific computation. Each node should be responsible for exactly one functionality and combined with it's dependents and dependencies a ROS application can be described as a graph.
This has many advantages not only from a developer's point of view. It reduces the complexity of the system as each node only has to be concerned with a certain task. It also makes the system fault tolerant as errors only compromise one node and not the whole system. The communication model over messages also makes nodes interchangeable as long as nodes listen on the correct type of topic and can correctly interpret the message which leads to the possibility to use a variety of implementations for the same task.
Every node in the ROS is uniquely identifiable at all times as each node has it's own unique name. These names are represented like UNIX paths for example */<node-name>*[6].

Services

Services are a special kind of node. Common nodes are impractical for the execution of remote procedure calls (RPC) as these calls usually return a response. Services are specifically designed for RPC actions as they can send a response to an incoming request which are hard to achieve with normal messages. The services usually show up in ROS just like any other node as they can be uniquely identified by a name[7].

4.1.2 Messages

Messages are one of the two building blocks of communication in ROS. Messages are used for communication by publishing them to a certain topic. Messages are type safe and support basic data types like integer, character strings and boolean types. Messages also support nesting of messages and arrays of either custom messages or basic datatypes.
Messages can be identified by the package name and the message filename with the file type *msg* stripped[8].

Message definition format

Messages can also be defined by the user. These message definitions have to reside in the *msg* subdirectory of the package. An exemplary message file can be seen below:

```
Header header
int32 operand_a
int32 operand_b

time stamp
```

So in general the message files follow this simple format[9]:

```
Type1 identifier1
```

[5]Peter H. Salus, *A Quarter-Century of Unix*.
[6]Open Source Robotics Foundation, *Nodes - ROS Wiki*.
[7]Open Source Robotics Foundation, *Services - ROS Wiki*.
[8]Open Source Robotics Foundation, *Messages - ROS Wiki*.
[9]Open Source Robotics Foundation, *msg - ROS Wiki*.

```
Type2 identifier2
TypeN identifierN
```

Message headers

Every message may include the special *Header* datatype which stores some common data like the timestamp or an incrementally increasing id. Some of the ROS client libraries can set some of these values automatically so the use of the Header type is highly encouraged. A common definition of the Header type may look like this[10]:

```
# sequence ID: consecutively increasing ID
uint32 seq

time stamp

#Frame this data is associated with
# 0: no frame
# 1: global frame
string frame_id
```

4.1.3 Topics

The second building block of communication in ROS are Topics. ROS topics are unidirectional named channels that enable nodes to communicate with each other. In general, nodes do not know who they are talking to but only know the message type of the topic they are either subscribing or publishing to. The producers of data publish to the channels while consumers of data subscribe to topics. There can be multiple producers which produce data at the same time on the same topic[11].

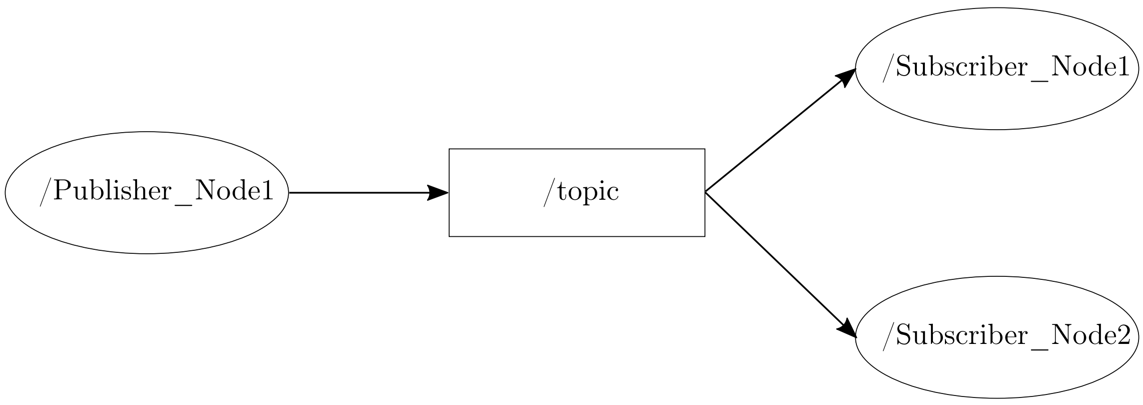

Figure 4.1: This figure shows the relationship between different nodes and topics. Multiple publishers can send data to topics and multiple subscribers receive data from these topics.

4.1.4 ROS Master

The most important part in every use of ROS is the so called ROS Master. This special node manages the communication between different nodes. It also acts as a service for the registration of nodes in the mesh.

[10]Open Source Robotics Foundation, *Messages - ROS Wiki*.
[11]Open Source Robotics Foundation, *Topics - ROS Wiki*.

4.2 tf Transform Library

Author: Alexander Lampalzer

The various coordinate systems of each part is a common problem for robots. Therefor the OSRF has developed the tf "transform library". This library is at the time of writing in its second generation and is responsible to keep track of the various coordinate systems. Developers can leverage this library by using the official Python or C++ libraries.[12] To achieve these goals, TF utilities a tree-like architecture, where every frame is a node. Developers can then use the provided libraries to ask questions similar to:

- Where was the right grappler in relation to the robot base 5 seconds ago?
- Where is the robot located inside my map?
- What is the relation between different maps of several robots?

In order to simplify the re-use of software components, ROS introduced several naming conventions for these coordinate frames[13], such as "earth", "odom" and "base_link".

4.3 Universal Robotic Description Format

Author: Alexander Lampalzer

The Universal Robotic Description Format (URDF) provides a unified XML-based file format and several tools for describing the components of a robot and their relationships[14]. URDF also supplies an integration with the gazebo simulator, to mock different parts of the robot, such as sensors, drive and motors. To achieve this, following XML tags are implemented:

- <link>
- <transmission>
- <joint>
- <gazebo>
- <sensor>
- <link>

URDF also provides several tools to parse, publish and convert these files. The urdf parser allows developers to parse files and convert them into c++ data structures. Furthermore, transform trees can also easily be published using the so called ̧robot_state_publisher and joint_state_publisher. The AUT-AS robot's URDF is visualised in Fig. 4.2.

Figure 4.2 depicts the physical relationships and transformations between sensors, their coordinate frames and the center of the AUT-AS robot. This allows calculation and utilization of multiple sensor sources in one single, unified system. It is a integral and vital part of every robotics system. In ROS, this is typically referred to as the transformation tree, or short tf tree. A set of best practices / naming conventions have been established in ROS enhancement proposal (REP) 103[15], 105[16] and 120[17].

[12]Foote, "tf: The transform library".
[13]Meeussen, *REP 105*.
[14]Open Source Robotics Foundation, *urdf - ROS Wiki*.
[15]Foote and Purvis, *REP 103*.
[16]Meeussen, *REP 105*.
[17]Moulard, *REP 120*.

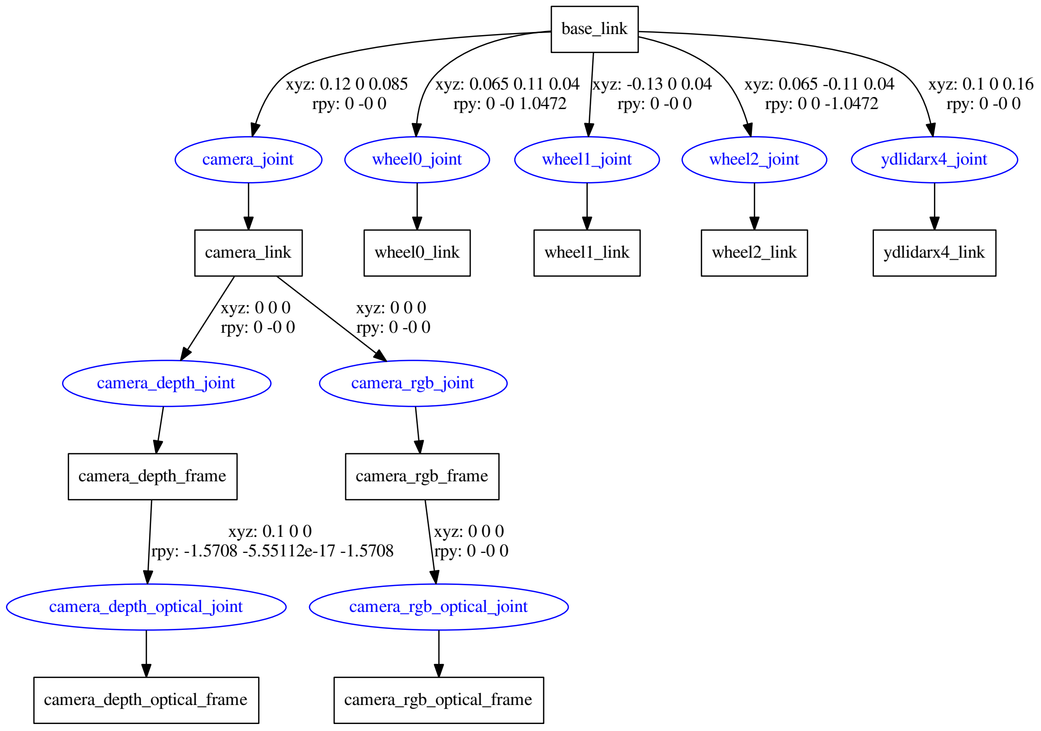

Figure 4.2: This figure visualises the different components of the AUT-AS robot as described by the URDF and the relations between them.

4.4 Gazebo Simulator

Author: Alexander Lampalzer

The modular architecture of ROS allows developers to switch out and mock different nodes in their systems. Gazebo uses this approach and provides a number of different nodes for simulating robots.
There are several ways to configure robots with Gazebo, such as SRDF, SDF and URDF. In order to unifiy the transform trees in the simulated and in the real environment, the AUT-AS robot is described using URDF, wich is then used for publishing the transform tree and simulating the different parts of the robot.

Chapter 5

Sensors

Author: Alexander Lampalzer

Mapping the environment and detecting obstacles in this environment is an essential task for the AUT-AS robot. In order to achieve this goal a variety of different sensors are needed.

5.1 Lidar

Light detection and ranging (Lidar) is a common distance measurement sensor. It sends out pulses of light and measures the time it takes for these pulses to return, hence it can also be classified as a time of flight sensor. By utilizing rotating mirrors, these sensors can also create maps of 2D/3D environments[1]. These distance measurements also form the basis for a lot of SLAM Algorithms. An example esult of a 2D lidar scan can be found in Fig. 5.1

5.1.1 ydlidarx4

To perform the required mapping of the environment, the AUT-AS robot uses the "X4" lidar made by the chinese company "ydlidar". The authors compared the most popular 2D lidar systems and concluded, that the ydlidarx4 provides a good entry solutions for teams with a limited budget, because of its relatively cheap price of 100.70 €.

Name	YDLIDAR X4	RPILIDAR A1M8
Price incl. UST.	100.70 €	100.70 €
Measurement Freq.	5 kHz	2 kHz
Rotational Freq.	7 Hz	1 Hz - 10 Hz
Max. Distance	11 m	6 m
Distance Resolution	$< 0.5\,\%$ (<2m); $< 1\,\%$	$< 0.5\,\%$ (<1.5m); $< 1\,\%$
Angular Resolution	0.48 ° - 0.52 °	1 ° @ 5.5 Hz
Weight	180 g	190 g

Table 5.1: A table comparing two entry-level lidar sensors.

[1]US Department of Commerce, *What is LIDAR*.

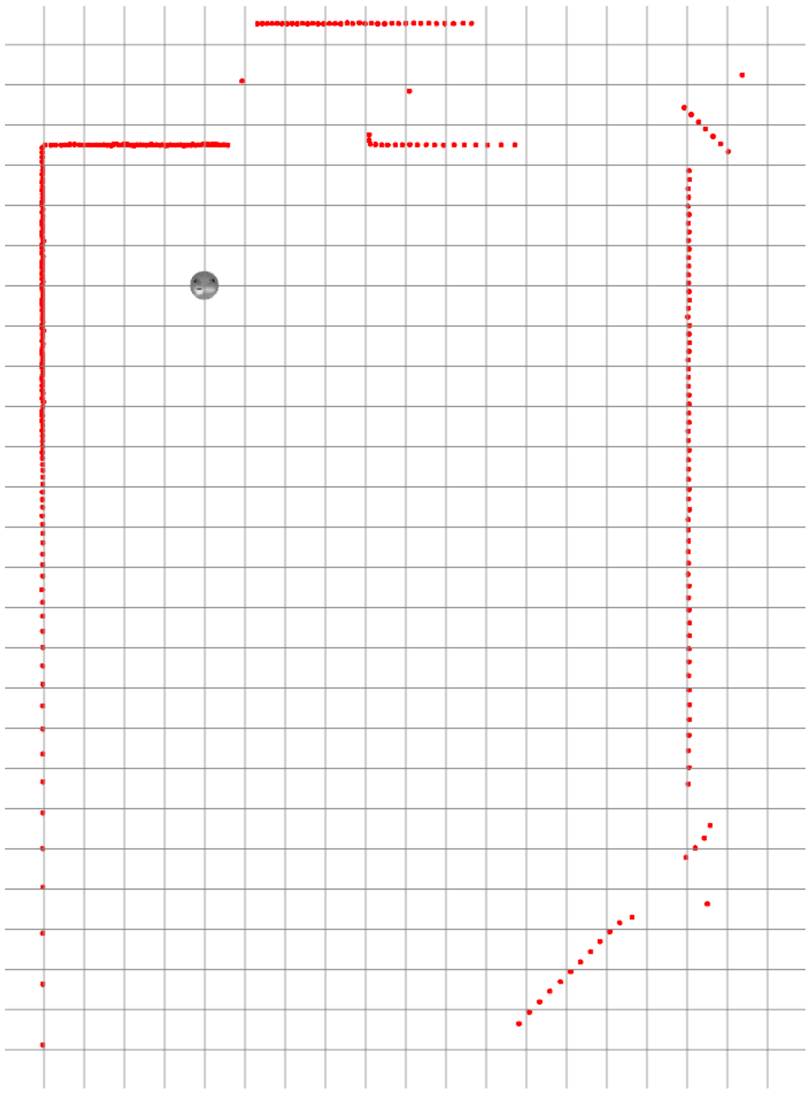

Figure 5.1: This image shows the result of a 2D lidar scanner. It was obtained by simulating the AUT-AS robot using the gazebo simulator. The density of the points decreases with distance from the sensor. This particular sensor has a maximum range of 15m, this graphic choses 0.5m as cell size.

5.2 Depth Cameras

Depth cameras provide depth information, in addition to regular color data. Since the release of the first affordable depth camera by Microsoft in 2010, the field has flourished and noumerus new applications have been developed. There is a variety of approaches to how depth cameras work, the most popular are explained below:

5.2.1 Structured Light

In an approach known as structured light, a projector (often infrared) is used to create a known pattern, such as a speckle or line pattern. This way, depth can be computed only using one camera and without the need of any external illumination source. However, reflective surfaces, such as mirrors, or infrared light, such as that produced by the sun, disrupt this pattern.[2]

5.2.2 Stereo triangulation

Another approach is to measure depth using multiple cameras by applying stereophotogrammetry. This does not require any external illumination source, like an IR projector and thus has an increased maximum distance. However, it needs the surfaces to be textured, because finding features on surfaces with e.g.: uniform colors is very inaccurate. Furthermore, stereophotogrammetry requires several computationally expensive steps.

5.2.3 Time of flight

Time of flight (TOF) cameras work simmilar to LIDAR sensors. By using an infrared light source, a pulse is emitted and the time needed for this light to return to the imaging sensor is measured. However, this doesn't come without disadvantages: In contrast to typical lidars, where just a single point is illuminated, a TOF camera illuminates the whole scene, This means, light may be reflected multiple times and therefore accuracy is decreased. It is also easily disturbed by other infrared light sources, such as the sun or light bulbs. However, there are also several advantages: Distance measurement in TOF sensors is relatively straight forward and as such it is computationally inexpensive in comparison with stereo cameras. This also allows the use in real-time applications[3]

5.3 Rotary encoders

A typical choice to measure the velocity, acceleration or angle of a shaft, such as a motor shaft, is a rotary encoder. These sensors are primarily designed to either output the current position or the velocity of an attached shaft. There are many applications this is useful for, especially in robotics. E.g.: it is used to estimate how far a wheel has moved, which then can be used to approximate how far the robot has traveled. Several approaches are used in the construction of encoders, most are either as mechanical, optical or magnetic. These greatly differ in their resolution and resistance to outside factors like dust and oil.[4]

5.4 Inertial Measurement Unit

Author: Königsreiter Simon

The Inertial Measurement Unit (*IMU*) is another type of sensor that measures the current angular velocity and acceleration. The *Razor IMU M0* is an attractive IMU by the company *Sparkfun*. This particular IMU combines an accelerometer to measure acceleration, a gyroscope to detect orientation and angular acceleration and a magnetometer to measure magnetic fields.

The general idea was to use the IMU for localization and tracking of the robot in combination with other sensors as the IMU is very inaccurate in calculating the distance traveled. Using the accelerometer for distance calculation is inaccurate because the values from the sensor have to be integrated two times. Each integration has a small error that sums up every time[5].

[2]Zanuttigh et al., "Operating Principles of Time-of-Flight Depth Cameras".

[3]Zanuttigh et al., "Operating Principles of Structured Light Depth Cameras".

[4]Eitel, *Basics of Rotary Encoders*.

[5]Christian B. Bellanosa, Ruth Pearl J. Lugpatan, and Diogenes Armando D. Pascua, "Position Estimation using Inertial Measurement Unit (IMU) on a Quadcopter in an Enclosed Environment".

5.4.1 Problems of the Razor IMU M0 with ROS

The Razor IMU M0 is per default not compatible with ROS. But there is an official ROS Firmware that enables the IMU to output ROS compatible messages to use in the system. The fact that the IMU features an Arduino (A company selling micro controllers and additional software) compatible micro-controller makes it easy to install the new firmware onto the device. The firmware requires to calibrate the IMU which is a cubersome task because it requires the user to slowly move the device in every possible direction without any rapid movement in all nine degrees while reading special values off a terminal window and then setting them in the firmware. After calibrating the sensors several times the team has found out that this particular IMU is affected by a known error in the software that is currently not possible to fix[6]. After finding out about this problem, the team has found another, community maintained, firmware that is also capable of outputting ROS compatible messages. Although this new firmware required some changes in the source code to actually produce output that is ROS compatible, it had several advantages over the officially promoted firmware:

- It actually works with the IMU.
- It didn't require any configuration and was pretty accurate out of the box.

Different coordinate systems

One problem that is especially important when mounting the IMU on the robot is that the coordinate systems used by ROS and the one described on the device itself differ. This is visualized in figure 5.2.

5.4.2 Measuring the accuracy of traveled distance

The team wanted to find out how much the previously mentioned error with the IMU is. Therefor a test was planned to measure how much this error is and how practical the IMU is to measure driven distance. Therefor several tests have been conducted in order to find the results.

All the experiments were programmed as a line follower to have a reference what the actual travelled distance should actually be. All the attempts were conducted in the gym of the HTL Wiener Neustadt.

Attempt one

The first experiment used the thick black line as shown in figure 5.3. Although several attempts were made using this line it has been discarded as a lot of black lines going away from the specified line made it difficult to consistently follow this specific line.

Attempt two

After the second attempt has been abandoned, the team has tried to use another set of lines in the gym that are more distinct to its near neighbour lines. The yellow line in figure 5.3 was unique in its colour compared to its Neighbors. Therefor a new line follower has been programmed to follow the yellow line. The problem with this approach was the reflective surface of the ground of the gym. Due to strong reflections of the light, the robot tended to detect the reflections of the sun through a window rather than the actual yellow line.

[6]Open Source Robotics Foundation, *razor_ imu_ 9dof - ROS Wiki*.

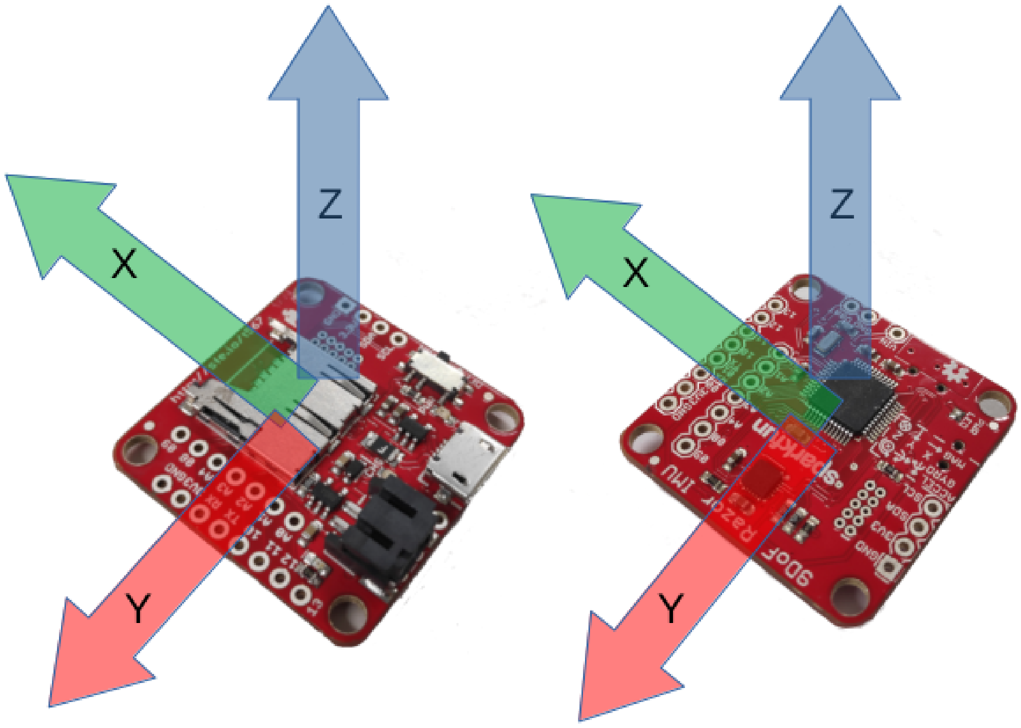

Figure 5.2: An image showing the difference of the coordinate Systems. Left: The coordinate system described on the IMU itself; Right: The coordinate system used by the ROS Messages and the firmware.

Attempt three

As it is not possible to access the gym at night the line follower of attempt two has been rewritten by combining the approaches of attempt one and two. For this effort the same line has been used but this time the input images are inverted in an attempt to get more distinguishable colours. Although the problem with the reflections got better it was still not enough for the robot to separate the line from the reflection.

Therefor the line following approach has been completely dismissed.

Attempt four

The last attempt is completely different than the others by tracking the robot with an external camera. For this approach the robot got a special AruCo tag that enables a camera to track the pose of the robot based on images received from a camera. The general idea is that the camera is positioned on an elevated position and looks down on the robot to make tracking simpler[7].

HaruCo Tags

HaruCo tags are special standardized tags that enable special libraries to precisely keep track of the position of the tags. The tags are usually used in combination with the *alvar* library that keeps track of the tag for the user. An image of these special tags can be found in Figure 5.4.

[7] *ar_ track_ alvar - ROS Wiki*.

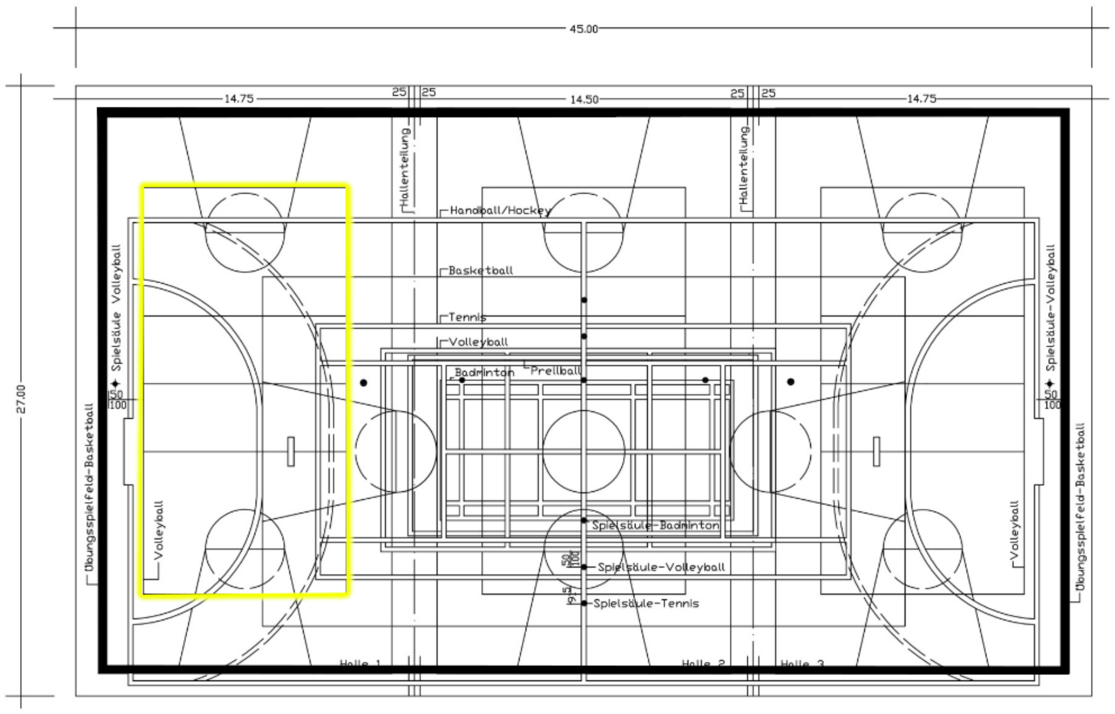

Figure 5.3: The schematic lines of the gym. Experiment one used the thick black line whereas experiment two and three used the yellow line.

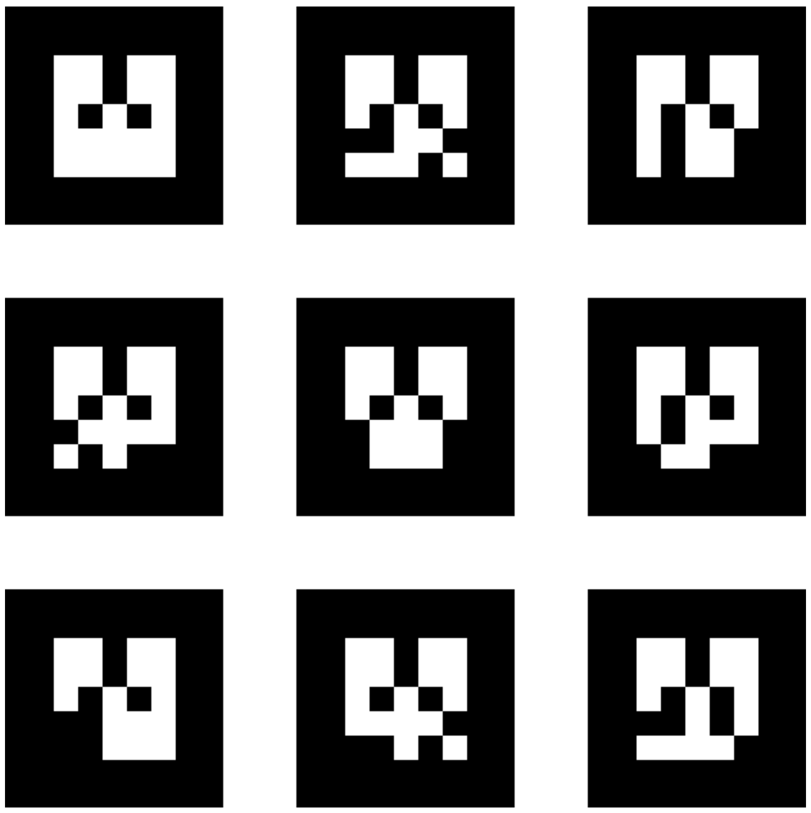

Figure 5.4: An image showing the special standardized tags used to accurately track a robots position[8].

Chapter 6

Sensor Fusion / State Estimation

Author: Alexander Lampalzer

A problem often encountered in modern robotics is that of state estimation, e.g.: positional or speed information. To achieve this goal it is necessary to measure the robot's actions using sensors. However, these sensors are prone to errors. By fusing data that originates from multiple sources, it is possible to increase the quality of an estimation. Modern state estimation techniques, such as the family of Kalman filters, heavily rely on input from multiple data sources. This chapter aims to describe how sensor fusion can be applied to the state estimation problem and what advantages and disadvantages different algorithms have.

The state estimation problem is often expressed in two very simple equations. 6.1 describes how state changes, this is expressed by $f(x)$. Often, additional input into the system is provided, which is described with $B(x)u$. Furthermore, it is assumed, that the output of a system is measurable. Described here with $h(x)$.

$$\dot{x} = f(x) + B(x)uy = h(x) \tag{6.1}$$

The state x contains information about the system - in a mobile robotics context this might be position and speed of the robot or information about objects in the environment.

The measurement z, often also called observation, commonly includes information relevant to the state, such as sensor measurements.

The controls u are inputs to the system, in mobile robots this most likely is odometry information.

6.1 Odometry

Author: Königsreiter Simon

Odometry is a special form of localization that enables a robot to calculate its position relative to its starting position. For ground based robots moving perfectly linear it is easy to calculate the relative position. To calculate the new position it is important to know the robots initial pose represented by the triple (x, y, θ) where x and y is the (x, y) position on the ground plane and θ is the direction the robot is facing[1][2].

The new pose can be calculated with the formula:

[1] Mordechai, *Elements of robotics*.

[2] Clark, "ARW – Lecture 01 Odometry Kinematics".

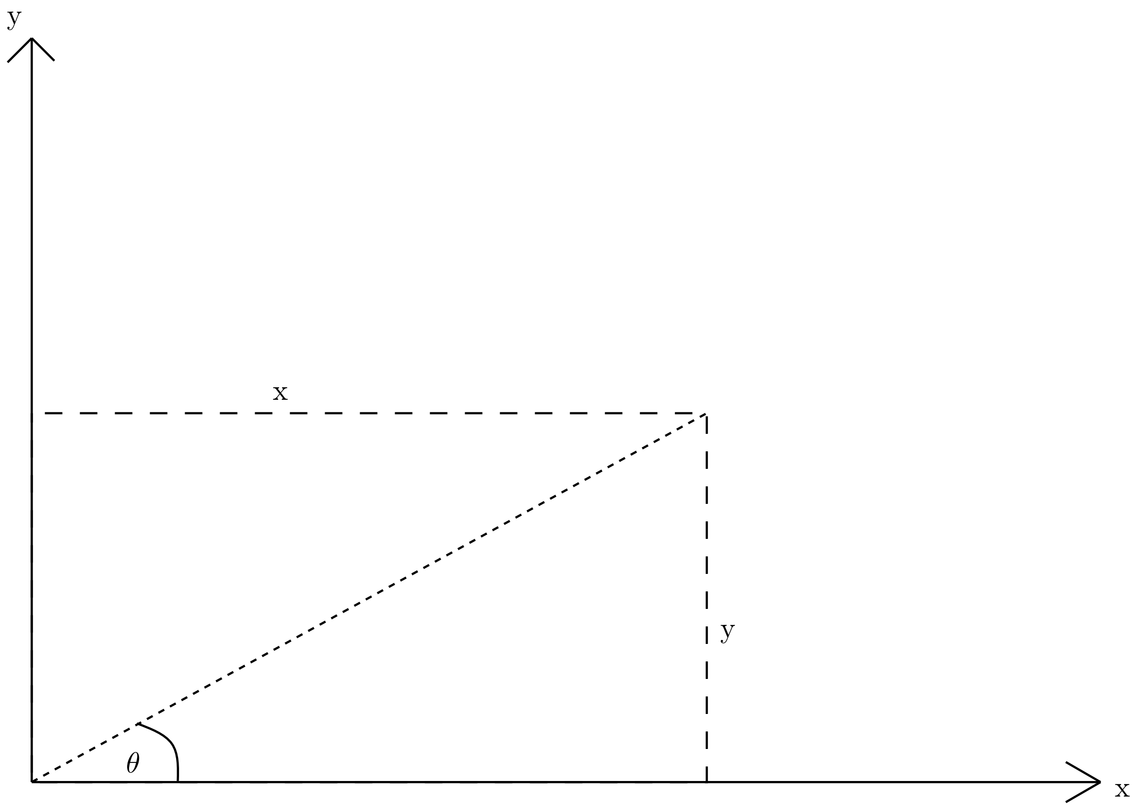

Figure 6.1: A figure showing the general way the pose is represented in the real world and how the state can go from an initial state to the next.

$$\begin{aligned} y &= vt * \sin\theta \\ x &= vt * \cos\theta \end{aligned} \tag{6.2}$$

6.1.1 Non-linear odometry

Due to differences in motor strength, wheel diameter and ground traction it is unrealistic to assume a perfectly linear model of movement. This section will now explain a method to calculate the state of the robot when the path contains arcs and turns.

Using the rotary encoders described in section 5.3 it is possible to calculate the traveled distance using the wheel radius r_s and the wheels' revolutions represented by ω_s where s can be either l for the left wheel of r for the right wheel or c for the center of the robot:

$$d_s = 2\pi r_s \omega_s t \tag{6.3}$$

Now to calculate the turn in radians the formula

$$\theta = \frac{d_s}{r_{sP}} \tag{6.4}$$

can be used where P is the origin of the turn.

To calculate theta with an unknown P the formula 6.4 can be used:

$$\begin{aligned} \theta r_{lP} &= d_l \\ \theta r_{rP} &= d_r \\ \theta r_{rP} - \theta r_{lP} &= d_r - d_l \\ \theta &= \frac{(d_r - d_l)}{r_{rP} - r_{lP}} \end{aligned} \tag{6.5}$$

Using the formula 6.5 one can also calculate the movement of the center of the robot r_{cP}:

$$\begin{aligned} d_{cP} &= \theta r_{cP} \\ d_{cP} &= \theta(\frac{r_{lP} + r_{rP}}{2}) \\ d_{cP} &= \frac{\theta}{2}(\frac{d_l}{\theta} + \frac{d_r}{\theta}) \\ d_{cP} &= \frac{d_l + d_r}{2} \end{aligned} \tag{6.6}$$

For small moved distances the change in the direction the robot is looking is θ and the change on the (x, y) plane can be calculated with

$$\begin{aligned} dx &= -d_{cP} \sin\theta \\ dy &= d_{cP} \cos\theta \end{aligned} \tag{6.7}$$

which results in the pose after the turn being

$$(-d_{cP} \sin\theta, d_{cP} \cos\theta, \phi + \theta) \tag{6.8}$$

based on its starting point.

This method is only working for small distances to keep the calculation simpler and the formula assumes a constant speed which is only possible for small distances[3].

Figure 6.2 Shows the various variables of the formulas above visualized.

6.1.2 Errors in Odometry

As it has been previously stated, odometry is prone to errors. These can be accumulated over time and are especially significant if the robot changes its heading. To calculate the error of a completely linear movement for a expected driven distance s and a maximum error of odometry p, the formula

$$\Delta x = n * \frac{p}{100} \tag{6.9}$$

can be used. To calculate the error of driven distance with a change in heading $\Delta\theta$ with a maximum error of up to p percent the formula

[3]Mordechai, *Elements of robotics*.

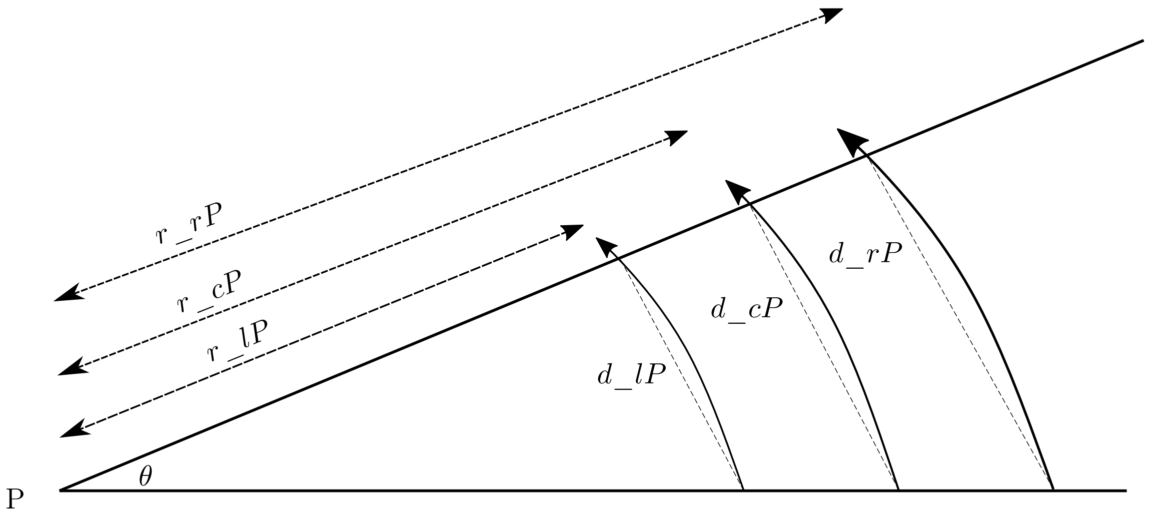

Figure 6.2: This figure shows the various figures to calculate the distance traveled with a two-wheeled robot making a slight left turn. The angle θ represents the turn radius in radians. r_{rP}, r_{cP} and r_{lP} represent the radius of the destination point based on an origin point P of the turn. d_{lP}, d_{cP} and d_{rP} represent the actual distance traveled by the left and right wheel and the center of the robot.

$$\begin{aligned}\Delta\theta &= 360 * \frac{p}{100} = 3.6 * p \\ \Delta y &= n * sin(3.6 * p)\end{aligned} \tag{6.10}$$

is used. To demonstrate the error of odometry for both the linear movement and a change in heading see Figure 6.3 which uses an $n = 10m$ and visualizes the error of $p \leq 10\%$.

To overcome the error prone input of odometry a selection of methods has been developed to combine the information from various sensors to increase accuracy. This method is called sensor fusion and is described in more detail in the following sections[4].

6.2 Bayes Filter

Author: Alexander Lampalzer

The Bayes Filter Algorithm provides a framework, used for recursive state estimation, meaning that it calculates the belief $bel(x_t)$ by utilizing the current belief $bel(x_{t-1})$. By assuming a **Markov property**, it only has to keep the current state x_t in memory and also saves computational resources. Bayes filter consists of two steps, commonly known as prediction and correction.

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\; bel(x_{t-1})\; \mathrm{d}x_{t-1} \tag{6.11}$$

The prediction step incorporates the **motion model** $p(x_t \mid u_t, x_{t-1})$ and uses this to predict the future given the last controls.

[4]Mordechai, *Elements of robotics*.

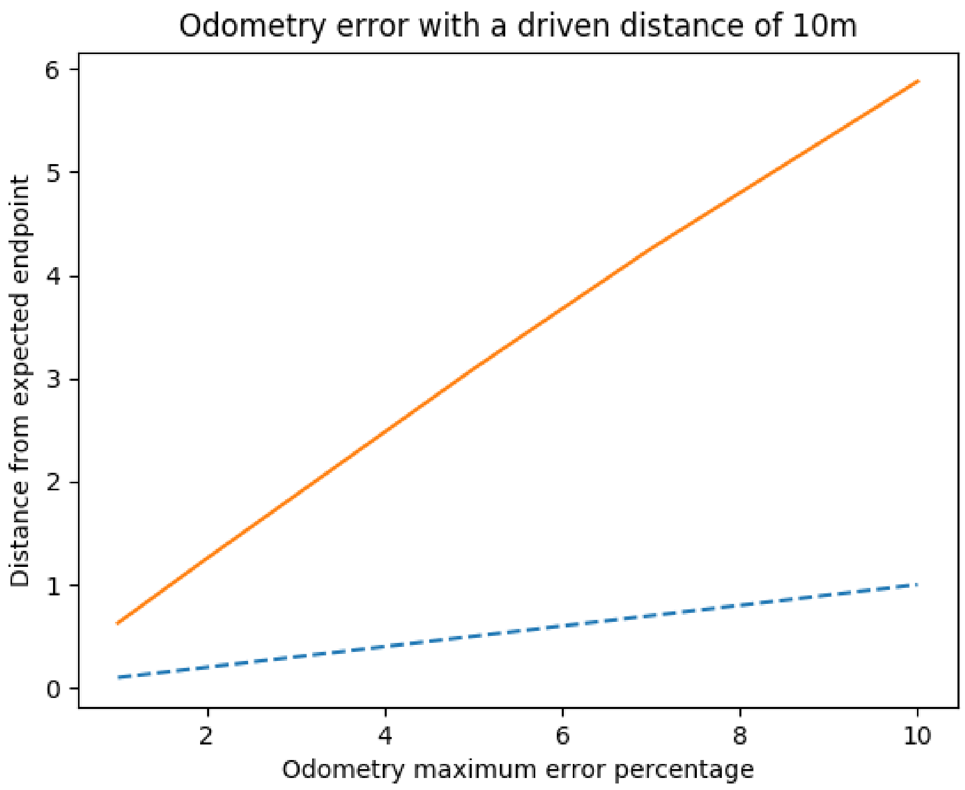

Figure 6.3: A figure representing ,0the error of odometry for linear movement and a non linear movement. The dashed line represents the error for the linear movement whereas the line is the error of non-linear movement. The x-Axis represents the maximum percent and the y-Axis the maximum offset of the target distance of 10 meters.

$$bel(x_t) = \eta \, p(z_t \mid x_t) \, \overline{bel}(x_{t-1}) \tag{6.12}$$

The next step is commonly called correction or measurement update step. Here, the belief $\overline{bel}(x_{t-1})$ is multiplied by what is commonly referred to as the **sensor model** $p(z_t \,|\, x_t)$. In some cases, this might not integrate to 1, because of this the result is normalised by multiplying with η.

The following sections describe different implementations of the bayes filter algorithm, each having its own advantages and disadvantages.

6.3 Kalman Filter

The Kalman Filter is probably the most known implementation of the Bayes Filter. It is, just like the bayes filter, a recursive algorithm. However, it assumes its internal state to be a linear dynamic system. Meaning, that at each iteration, a linear operator is applied to the state to generate a new state. Often, this assumption is not necessarily true for more complicated systems. The Kalman filter also assumes the observation and process noise

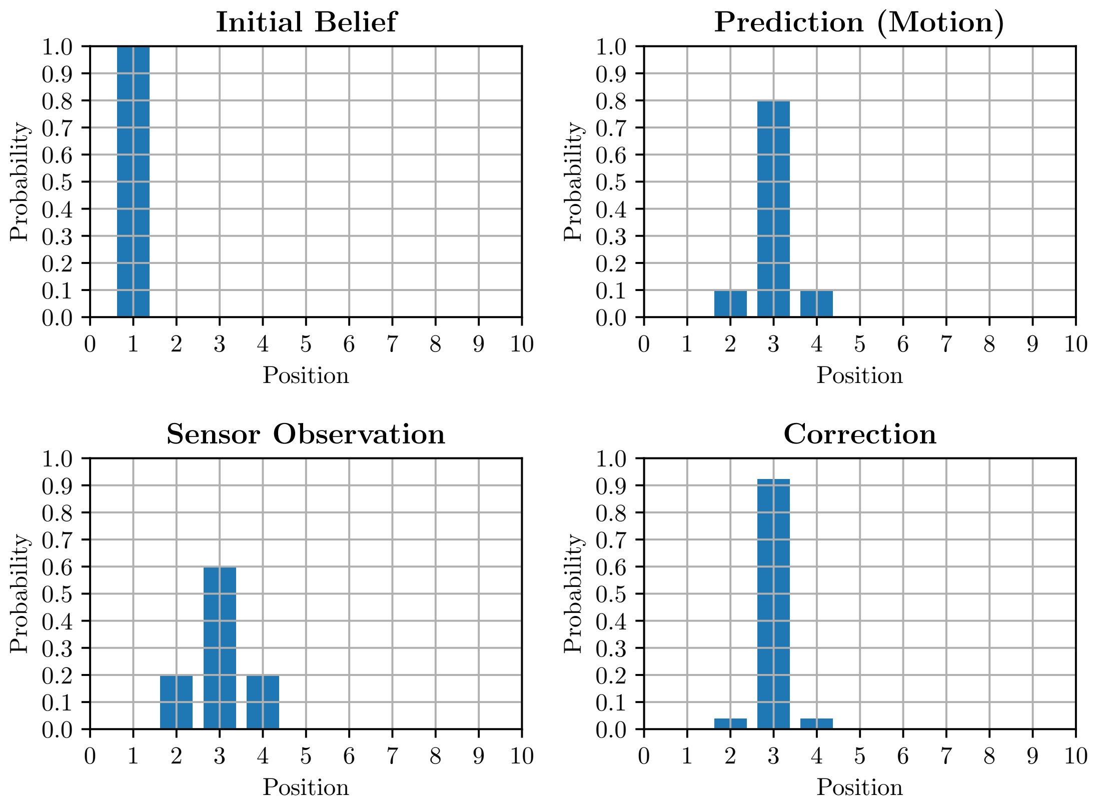

Figure 6.4: This graphic depicts the discrete bayes filter algorithm and its different stages. An initial belief at position 1 is assumed. Following this, a motion two positions forward with included gausian uncertainty is introduced. This motion is verified using a sensor observation and the prediction is corrected.

to be Gaussian, which also is not necessarily correct. Nonetheless, it is still widely used technique, especially for sensor fusion.

6.3.1 Mathematical Definition

Variable	Description	Dimension
x	State Vector	$n_x \times 1$
P	Covariance Matrix	$n_x \times n_x$
u	Control Vector	$n_u \times 1$
z	Observation Vector	$n_z \times 1$
F	State-transition Model	$n_x \times n_x$
B	Control-input Model	$n_x \times n_u$
H	Observation Model	$n_x \times n_z$
R	Observation Noise Covariance Matrix	$n_z \times n_z$
w	Process Noise Vector	$n_x \times 1$
Q	Process Noise Covariance Matrix	$n_x \times n_x$
K	Kalman Gain	$n_x \times n_z$

Table 6.1: This table shows the different variables required for computing the Kalman algorithm with their size.

```
def kalman_filter($x_{t-1}$, $u_t$, $z_t$):
    $\overline{x}_t = F_t x_{t-1} + B_t u_t + w_t$          # Prediction of a new state
    $\overline{P}_t = F_t P_{t-1} F_t^T + Q_t$            # Covariance associated with state

    $K_t = \overline{P}_t H_t^T (R_t + H_t \overline{P}_t H_t^T)^{-1}$    # Kalman Gain
    $x_t = \overline{x}_t + K_t (z_t - H_t \overline{x}_t)$          # Correction of state
    $P_t = \overline{P}_t (I - K_t H_t)$              # Correction of associated covariance

    return $x_t, P_t$
```

Listing 6.1: Kalman-filter algorithm described in pseude-code.

6.3.2 Example

In order to demonstrate the usage of a Kalman-Filter, consider the following example: Some vehicle is moving along a two-dimensional plane. It is assumed, that the object's velocity stays constant. This vehicle has sensors built-it, that return odometry information. However, using this alone does not suffice to estimate the position, as even small errors in the odometry accumalate and lead to something, that is known as drift. To counteract this, the estimations from a GPS sensor are fused with the data produced by the odometry. For more information about the models used, see[5]. The state is defined as having a positional and a speed component:

$$x_t = \begin{bmatrix} p_x & v_x & p_y & v_y \end{bmatrix}^T, F = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 & 0 & 0 \\ \frac{1}{2}\Delta t^2 & \Delta t & 0 & 0 \\ 0 & 0 & \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & \frac{1}{2}\Delta t^2 & \Delta t \end{bmatrix} \sigma_w^2$$

GPS sensors only determine the current position. Applications might include additional components to measure the heading or acceleration.

[5]N. Shimkin, "Kinematic Models for Target Tracking".

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, R = \begin{bmatrix} \sigma_z^2 & 0 \\ 0 & \sigma_z^2 \end{bmatrix}$$

For this simulation, a frequency of measurement for the GPS sensor of $10\,Hz$ and an accuracy of $2.5\,m$ was chosen. An accuracy of $0.1\,m$ was chosen for the odometry. The results can be found in 6.5. It can be seen, that the odometry produced is a very smooth path, that however diverges as time continues. GPS measurements are not very accurate, however they provide enough information to reduce the over error, which results in an estimation superior to when using GPS or odometry alone.

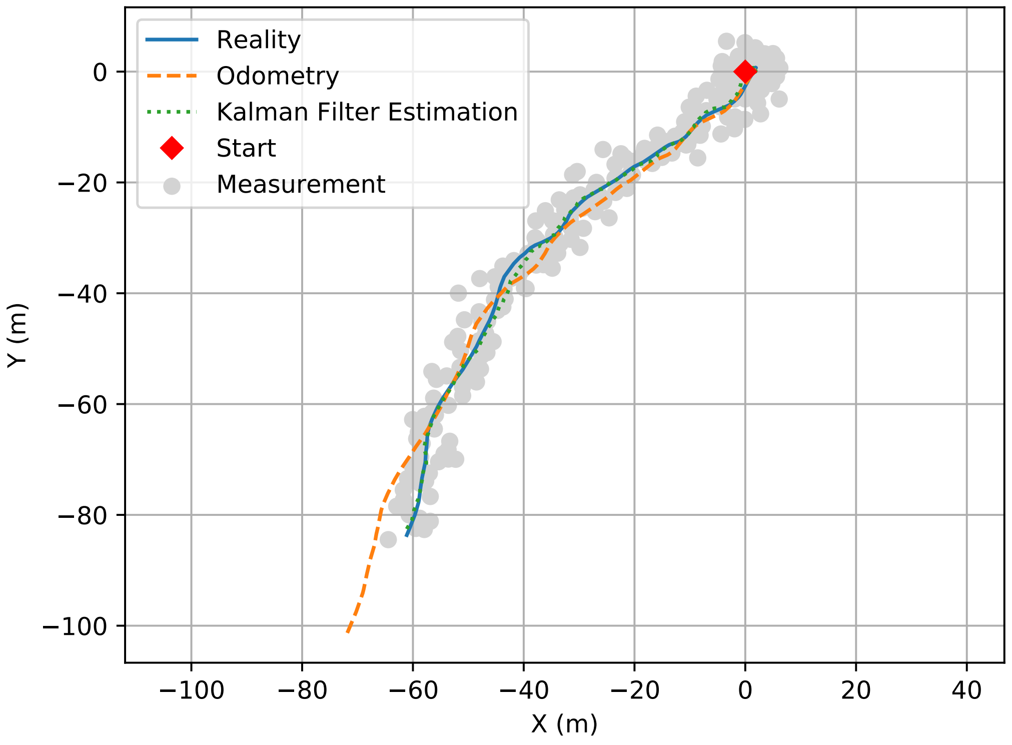

Figure 6.5: Simulation of a object moving with constant velocity along a plane. Introduction of a GPS sensor and a Kalman filter allow an accurate estimation of the current position of this object.

6.4 Extended Kalman Filter

The Kalman filter discussed in the last section uses linear equations for the process and observation models. In reality however, a lot of problems are non-linear. One approach to handle this problem is by linearizing the system at the point of current estimate. This section aims to detail how exactly this is achieved.

6.4.1 Jacobian matrix

The Extended Kalman Filter (EKF) makes extensive use of jacobian matrices - for this reason it is necessary to explain what jacobians are and their notation. A Jacobian matrix consists of all first-order partial derivatives of a function. It is best explained by considering a simple example with functions $f_1(x,y) = x^2 + y^2$ and $f_2(x,y) = \cos x + y$

$$\mathbf{f}(\begin{bmatrix} x \\ y \end{bmatrix}) = \begin{bmatrix} x^2 + y^2 \\ \cos x + y \end{bmatrix}, \mathbf{J_f}(x,y) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x} & \dfrac{\partial f_1}{\partial y} \\ \dfrac{\partial f_2}{\partial x} & \dfrac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x & 2y \\ -\sin x & 1 \end{bmatrix} \tag{6.13}$$

6.4.2 Linearization

One underlying assumption of the Kalman family of filters is, that they assume everything to be gaussian. This is easily violated when the observation or process model are non-linear. The EKF approaches this problem by taking the partial derivative of these non-linear functions and evaluating at the mean. However, this is the root cause of problems associated with the EKF, because by linearization only an approximation of the original function is used. Multiple solutions exist for this, such as the Unscented Kalman Filter. This results in the following formulas:

```
def extended_kalman_filter($x_{t-1}$, $u_t$, $z_t$):
  $F_t = \frac{\partial f}{\partial x}|_{x_{t-1},u_t}$
  $\overline{x}_t = f(x_{t-1}, u_t) + w_t$
  $\overline{P}_t = F_t P_{t-1} F_t^T + Q_t$

  $H_x = \frac{\partial h}{\partial x}|_{\overline{x}_t}$
  $K_t = \overline{P}_t H_t^T (R_t + H_t \overline{P}_t H_t^T)^{-1}$
  $x_t = \overline{x}_t + K_t(z_t - h(\overline{x}))$
  $P_t = \overline{P}_t(I - K_t H_t)$

    return $x_t, P_t$
```

Listing 6.2: The EKF algorithm in pseudo-code.

6.5 Unscented Kalman Filter

As can be seen in the previous section, the EKF utilizes taylor expansion to linearly approximate non-linear functions around their mean. This however, can introduce large errors. The Unscented Kalman Filter (UKF) aims to increase the performance and hence reduce the introduced errors. The UKFs approach is quite simple: Instead of taking only one point (the mean), several points (Sigma Points) are sampled from the original distribution and transformed through the non-linear function. The new mean and covariance are then calculated from the result. Furthermore, these points can also be weighted. This is also called the "unscented transform".

To compute the unscented transform, a set of sigma points has to be chosen. Many methods to choose these have been proposed, however in general the approach by S. Julier et. al[6] is used. They suggest the use of $2n+1$ sigma points χ_i, with n being the dimen-

[6]Uhlmann, S. Julier, and Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators".

sionality of the state x. The following calculations assume λ to be a scaling parameter, where $\lambda = a^2(n+\kappa) - n$. α and κ determine the spread of sigma points around the mean, typical values for are $\alpha = 1^{-4}$ and $\kappa = 0$. β is related the distribution of the state vector - for gaussian distributions the optimal value of $\beta = 2$. Implications of different values for α, β, κ can be found in Bitzer, *UKF-Exposed*

$$\begin{aligned} \chi_0 &= x \\ \chi_i &= x + (\sqrt{(n+\lambda)P})_i, \quad i = 1, \dots, n \\ \chi_i &= x - (\sqrt{(n+\lambda)P})_{i-n}, \quad i = n+1, \dots, 2n \end{aligned} \tag{6.14}$$

$$\begin{aligned} W_0^{\mu} &= \lambda/(n+\lambda) \\ W_0^{\Sigma} &= \lambda/(n+\lambda) + (1 - \alpha^2 + \beta) \\ W_i^{\mu} &= W_i^{\Sigma} = \frac{1}{2(n+\lambda)}, \quad i = 1, \dots, 2n \end{aligned} \tag{6.15}$$

The next step is to transform all sigma points through the non-linear function f, which results in a new set of sigma points $\gamma_i = f(\chi_i)$. This new set is used to approximate a new mean and covariance.

$$\overline{\gamma} = \sum_{i=0}^{2n} W_i^{\mu} \gamma_i \tag{6.16}$$

$$\overline{P} = \sum_{i=0}^{2n} W_i^{\Sigma} (\gamma_i - \overline{\gamma})(\gamma_i - \overline{\gamma})^T \tag{6.17}$$

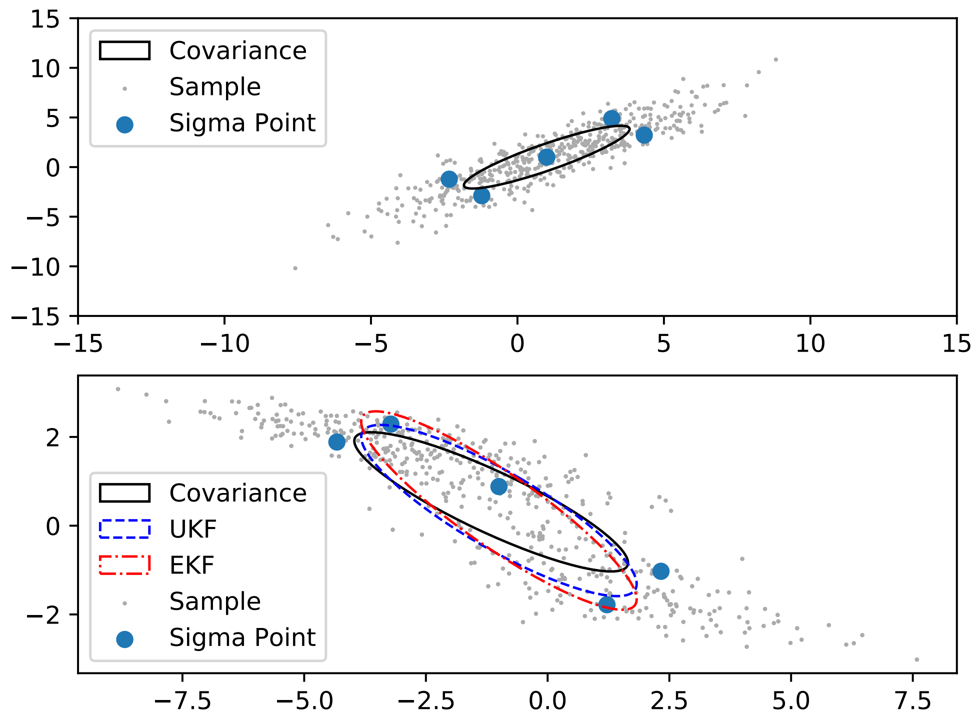

Figure 6.6: This diagram compares 2D EKF and UKF filters, with the covariance obtained by sampling 500 points. The top figure shows the randomly generated input data and it's associated covariance. In the bottom figure, the results of the EKF, the ukf and it's sigma points are shown. It can be seen, that the (dashed) covariance of the UKF represents the true covariance more closely than the EKF counterpart.

Chapter 7

Simultaneous Localization and Mapping

Author: Alexander Lampalzer

At it's core, the simultaneous localization and mapping (SLAM) problem has two goals: Estimating a map and the position of a robot within this map. In mathematical terms: Given observations Z and the controls U, estimate the current position x_t and the environment m. This is often referred to as "Full SLAM", meaning that the whole path and map are estimated.

$$P(x_{0:t}, m|z_{1:t}, u_{1:t}) \tag{7.1}$$

What is referred to as "Online SLAM", means that only the current pose and map are estimated. In this variation, new data is introduced to the system while it is running - thus the name.

$$P(x_t, m|z_{1:t}, u_{1:t}) \tag{7.2}$$

Over the years, this problem has been introduced to many fields and applications, such as home appliances like vacuum cleaners, autonomous driving, unmanned aerial vehicles and even planetary rovers such as the mars curiosity rover.

7.0.1 Map Representations

The two most common types of map representations used are grid-based maps and feature / landmark based maps - both with their respective advantages and disadvantages. In the AUT-AS project, grid maps are utilized, as they include the topology of the physical environment and thus allow navigation and path planning. Grid based maps utilize occupancy grids, by dividing these maps into cells (eg. 10 by 10 centimeter) and storing whether this cell is blocked by a wall. In contrast, feature based maps only contain the location of certain landmarks, which can then be used for localization.

7.0.2 Kalman Filter Approach

Kalman filter based approaches utilize an EKF, hence the name. The author suggests a good understanding of this technique, before diving into this section. See 6.3 for a basic introduction. The EKF-based approach utilizes a landmark based map, this results in the map m containing the position of the robot, the positions of the landmarks and the covariance of both. Each cycle of this algorithm, following steps are conducted:

1. State prediction: In this step, the motion model is applied and the position and covariance of the robot are updated. Do note, that the position of the landmarks is not changed yet!
2. Measurement prediction: The 2nd step utilizes the changed position and predicts, what the new measurement could look like.
3. Measurement: A new measurement is take from the observation sources.
4. Data association: Now the difference between the actual and the predicted measurement is computed.
5. Update: Finally, the position and covariance of the robot and all landmarks are updated.

7.0.3 Graph-based SLAM

In contrast to the Kalman-Filter approaches, graph-based approaches utilize a graph (hence the name) to represent the slam problem. In this graph, nodes correspond to robot positions and observations and edges correspond to constraints, such as odometry edges. This graph is optimized and the error minimized, which in the end allows for a map to be created.

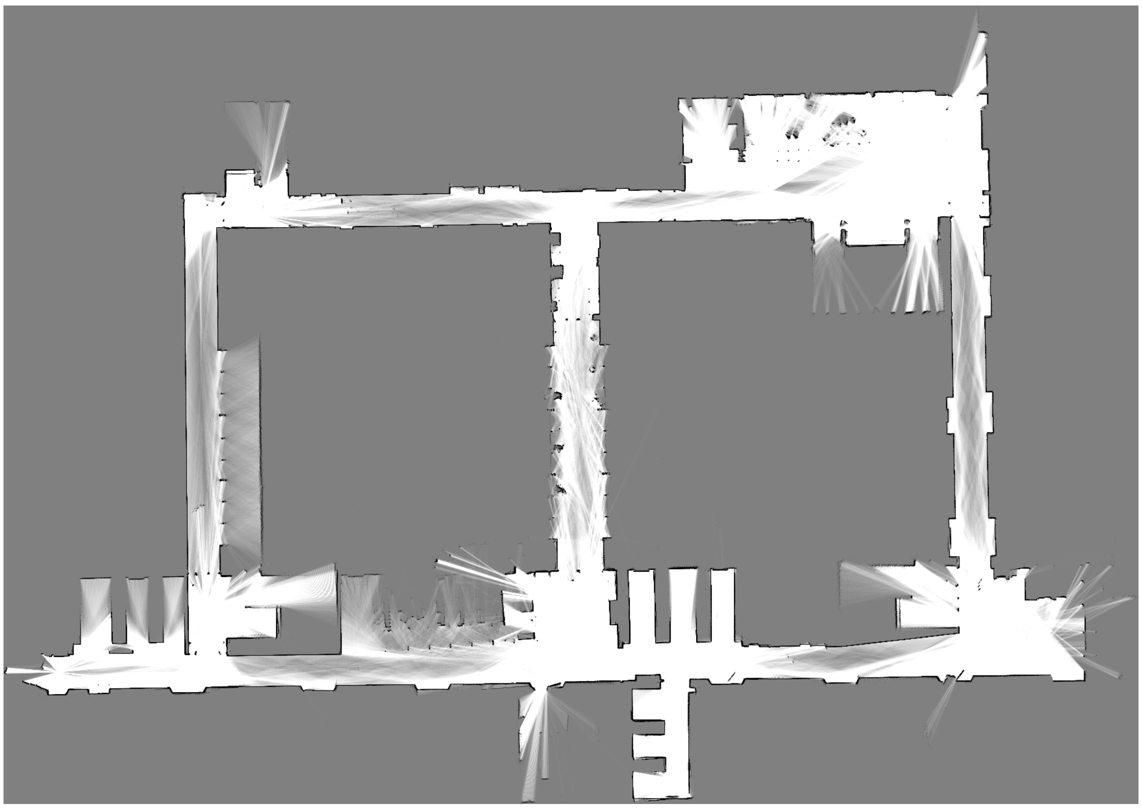

Figure 7.1: Example of a map generated by the cartographer slam algorithm, of the 2nd floor of the HTBLuVA Wiener Neustadt faculty for IT. This map includes many errors, such as glass reflections.

Chapter 8

Locomotion

Author: Königsreiter Simon

Just as important as the robots perception is its movement. Therefor the type of locomotion is just as important as the equipped sensors. Locomotion is the counterpart to manipulation where in manipulation the robot's arm is fixed and force is applied to an object. In locomotion on the other hand the robot is fixed and applies a force to the environment. Most of the propulsion types have been inspired especially the various types of legged locomotion. Another type of robot is the wheeled robot. Although wheels don't appear biologically in nature like legged types of drive, the wheel can be partially derived of the bipedal walk of humans. The bipedal walk of humans can be modeled after a rolling hexagon where the length of a side is the step size and the radius of the wheel is the height of the leg.

The main considerations when choosing or designing the type of drive are:

- Stability
- Maneuverability
- Controllability

The following sections are going to elaborate the various kinds of robots in more detail and each of its advantages and disadvantages[1].

8.1 Legged robots

Legged robots have been heavily inspired by nature as it provides various efficient examples like humans or ants. There have been working examples in the field of robotics with various leg counts ranging from one legged robots with up to six or more legs. Legged robots have a variety of advantages compared to wheeled robots. The biggest is the ability to leap over gaps with up to the length of the leg. Furthermore it is also possible to navigate more energy efficient through muddy terrain and uneven ground or sudden changes in level like stairs which do not impose such a big of a problem. The disadvantage of legged robot on the other hand is the need to keep the balance of the robot depending on the count of legs. If there are more than one leg it is also required to perform leg coordination to efficiently move forward[2][3].

[1] Siegwart, Nourbakhsh, and Scaramuzza, *Introduction to Autonomous Mobile Robots*.
[2] Böttcher, "Principles of robot locomotion".
[3] Christensen, "Robot Locomotion".

8.1.1 One legged robots

Although this is a rather uncommon form of robot, it is possible and can actually be implemented. Rather than walking like humans, this robot uses a hopping style of movement to keep its balance. With the efficient use of springs it is possible to use most of the used energy gathered through the impact of the jump and reuse it. Therefor it is not necessary to carry big battery packs and this enables the robot to keeps its overall mass low. This type of movement is incapable of static balance and has to use dynamic balance. Static balance describes a type of balance where it is not necessary to regularly apply force to keep the robot balanced. Dynamic balance is the opposite of static balance and requires regular application of force.

8.1.2 Two legged robots

Two legged robots have mostly been modeled after the bipedal style of walking humans. This type of movement is the first type of movement that requires the need of leg coordination in star contrast to one legged robots. With the help of an additional leg it is also easier to keep balance but still requires some force applied to the ankles to keep the robot upright.

8.1.3 Three legged robots

Three legged robots are the first type of legged robot where static balance is possible as long as all three feet are on the ground.

8.1.4 Four legged robots

These types of robots have been inspired by various types of animals like cats and dogs. The count of feet also makes it possible to statically balance the robot. The four legged robot is especially used in the field of integrating robot pets into human's life. The additional two legs compared to the two legged robot enable the robot to climb or overcome obstacles higher than its waist.

8.1.5 Six legged robots

These robots use insects as their primary inspiration. The six feet are useful to keep the robot balanced as it is possible to keep three feet on the ground at all times. This also implies that the robot can be balanced at all time. The complexity of the six legged robot can be far more complex compared to the previous robots as an additional joints need to be placed on the body of the robot to enable turning. Therefor the legs have to be coordinated with the legs to prevent collisions of the legs with each other[4][5].

8.2 Wheeled robots

Wheeled robots are usually found in all types of man made machines and robots with wheels are very efficient. Depending on the type of wheel they are significantly easier to handle than legs, but the challenge with wheeled robots lies in the design to arrange the wheels in a way that they are best suited for any terrain.

The various types of wheels include the following:

[4]Christensen, "Robot Locomotion".

[5]Siegwart, Nourbakhsh, and Scaramuzza, *Introduction to Autonomous Mobile Robots*.

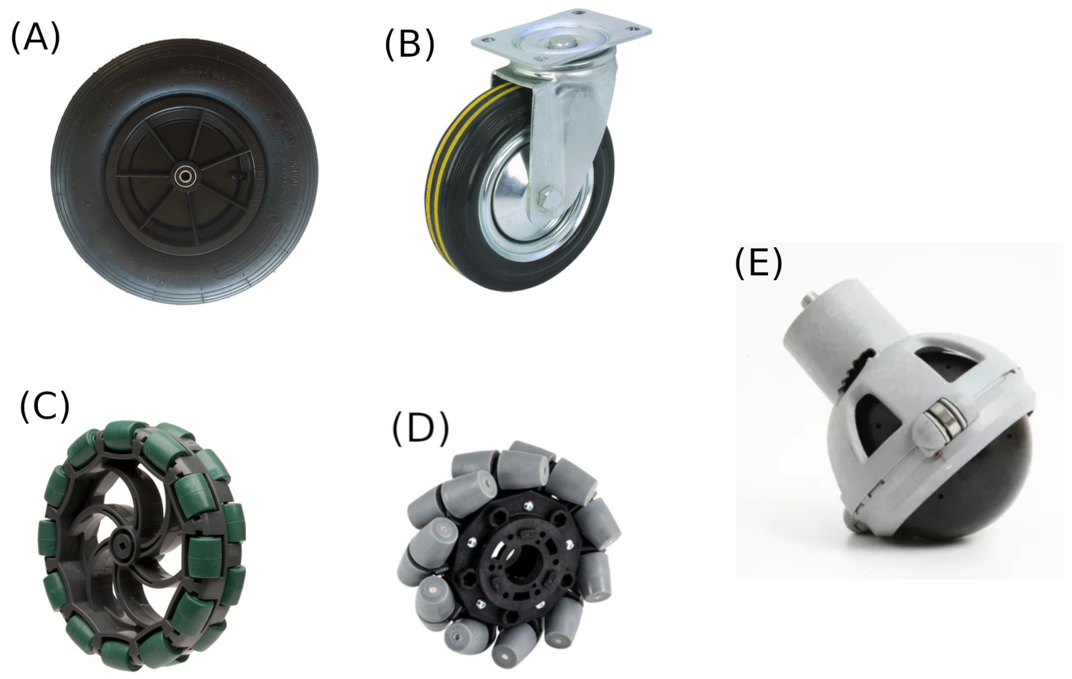

Figure 8.1: An image representing the 4 common wheel types. (A) Standard wheel with a motorized axel. (B) Castor wheel with a motorized axel and another axel at the top. (C) and (D) Swedish wheels either 90° or 45°. They have a high degree of freedom as they can glide across the ground with low friction. (E) Spherical wheel with the highest degree of freedom and really omnidirectional

- The standard wheel: This is a simple wheel with two degrees of freedom. It can rotate around the usually motorized axle and the point of contact with the ground.
- The castor wheel: This wheel also has two degrees of freedom of movement around the ground contact point and the offset axis on top.
- The Swedish wheel: This type of wheel has three degrees of freedom. The motorized axle, the contact point on the ground and it can also move across the ground plane using the passive wheels on the edge of the main wheel. This has the advantage that the robot can move freely across the (x, y) plane.
- The spherical wheel: A truly omnidirectional wheel that can move freely in all possible direction on the ground plane. Usually implemented by placing a powered roller on top of the sphere to apply force to it.

The different wheel types described above are important as they directly impact the wheeled robots three main properties: its maneuvrability, stability and controllability[6].

8.2.1 Stability

To achieve static stability it is necessary to have at least three contact points at each point in time to keep the robot balanced. Although it is possible to achieve static balance with two wheels it is impractical as the center of mass has to be located below the main axle which quickly results in a large wheel radius.

[6]Siegwart, Nourbakhsh, and Scaramuzza, *Introduction to Autonomous Mobile Robots*.

8.2.2 Maneuverability

Depending on the type of wheel the maneuverability can be greatly improved or can be really limited. One example of great maneuverability is a robot using three Swedish wheels as it can move across the ground in any direction. The star contrast is the Ackermann wheel architecture usually found in cars. This configuration limits the maneuverability and robots or cars using this configuration usually require large radii to turn.

8.2.3 Controllability

The complexity to control a robot is usually directly tied to its maneuverability. The spherical wheel is the most complex type of wheel to control as a lot of calculations have to be performed to convert the desired direction movement into actual wheel commands. The degrees of freedom in movement also have an impact on the amount of error corrections that have to be performed while driving. A small robot using the Ackermann wheel architecture (a wheel configuration commonly found in cars consisting of two motorized standard wheels and two passive steering wheels) can easily drive in a straight line by simply locking its steering wheels, whereas a robot using omnidirectional wheels has to continuously correct errors in the motor speeds or when the ground is slippery.

Chapter 9

Image Classification and Object Detection

Author: Königsreiter Simon

To support humans with the AUT-AS robot an additional Object Detection and Classification system has been added to the platform. The system can help humans in detecting and identifying objects with the help of the robot. The following sections will introduce the concepts of Classification with the help of neural networks and then compare a selection of various implementations of the task.

9.1 Definition of classification

The task of classifying things is a part of many human activities. But especially in the field of machine-assisted classification it can be defined as extracting features from a given input and then the assignment of a certain type to the input. If the classes of an object are known beforehand it may also be referred to as pattern recognition, supervised learning or discrimination. The task of machine-assisted classification can find a variety of different use-cases like sorting letters with a machine after reading the zip code with a camera or a precautionary treatment based on symptoms in a medical faculty until the final results have been anounced. Many approaches have been taken that share the same goals of exceeding the human's brain in the ability to make decisions and to generalize a large amount of data so unknown data may also be processed by the system. Three of these approaches are the neural network, machine learning and the statistical.

- *Statistical approach:* This approach uses an underlying probability method. This leads to the result that the output of the statistical approach is rather a list of probabilities rather than a single classification.
- *Machine learning:* Machine is an approach where the system should learn the connections from given example data. This may also require large amounts of example data for the system to form the connections for classifications. An example for machine learning is a decision tree or genetic algorithms.
- *Neural networks:* Neural networks were created with the human mind as an inspiration where a neural network consists of multiple layers of neurons that are connected with each other. There are input nodes that get their input directly from the available

data whereas there are output nodes that ultimately determine the classification[1].

9.2 A basic introduction to neural networks

As described above, a neural network (NN) is a weighted directed graph where the neurons represent nodes in the graph and the connections between them represent the edges. Learning is accomplished by adjusting the weights between the nodes. If a classification for a specific input is correct then the weight will be increased so the neuron will activate with similar data. If the output of the NN is incorrect on the other hand, the weight will be decreased for the so the node will not activate for similar data again.

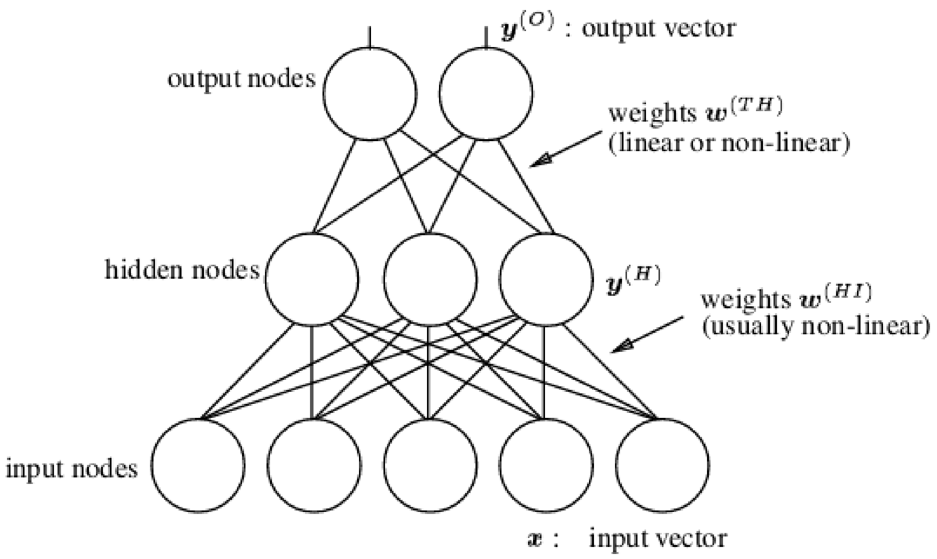
$y^{(O)}$: output vector
output nodes
weights $w^{(TH)}$ (linear or non-linear)
hidden nodes
$y^{(H)}$
weights $w^{(HI)}$ (usually non-linear)
input nodes
x : input vector

Figure 9.1: A diagram showing the structure of a Multi Layer Perceptron (MLP) similar to those used in modern systems.

Figure 9.1 shows a simple scheme for a Multi Layer Perceptron (MLP) which is used in today's neural networks. The network has x input nodes and unlike in the graphic an undefined number of hidden layers. The hidden layers in general tend to be small to force the network to generalize data[2].

9.2.1 Training the NN with backpropagation

The hidden layers in a neural network propose a problem as it is not as easy to find out which node lead to the error in an output. Therefor the backpropagation algorithm has been developed to deal with the error of hidden layers. It builds upon the delta rule that on the other hand builds on top of the mathematical concept of gradient descent. To understand the mathematical background of backpropagation the basics of the delta rule will be elaborated further first.

The delta rule is one basic training rule for neural networks and can basically be used to minimize the error of a neural network. The NN in general for the delta rule has no

[1]Michie, Spiegelhalter, and Taylor, "Machine Learning, Neural and Statistical Classification".
[2]Michie, Spiegelhalter, and Taylor, "Machine Learning, Neural and Statistical Classification".

hidden layers and an arbitrary number of input and output nodes o. Now take a random output node o_n and any element from the input vector X_p. Then the target output of the neural network can be defined as $t(X_p, o_n)$. The actual output of the node o_n can then be defined as

$$y(X_p, o_n) = s(a(X_p, o_n)) \tag{9.1}$$

where s is a non-linear function usually also called *sigma* that returns a value between 0 and 1 and is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9.2}$$

and a is the neurons activation function that either activates - or returns a value not equal to zero - for a given input or not. Therefore adjusting the weight of each node minimizes the error between the functions t and y which ultimately makes the NN more accurate. To calculate the error of some X_p and some output node o_n the formula

$$\frac{1}{2} * (t(X_p * o_n) - y(X_p, o_n))^2 \tag{9.3}$$

is used. To get the average error of the whole network the following python code can be used:

```python
sum = 0
for p in range(0, len(trainging_vector)):
    for n in range(0, output_nodes):
        sum += 1/2 * ((t(p, n) - y(p, n)) ** 2)
average_error = sum / len(training_vector)
```

Listing 9.1: An example python code that calculates the average error of a given NN

To get the change for a specific weight W_i the formula

$$\alpha * s'(a(X_p, o_n)) * (t(X_p, o_n) - y(X_p, o_n)) * X_p(i, o_n) \tag{9.4}$$

can be used where α represents the learning rate, a number $0 < \alpha < 1$, which represent how radical the NN changes the weights and $X_p(i, o_n)$ represents the weight i for an output node o_n with the specified input X_p.

Based on this information the backpropagation algorithm can be explained further more. For the backpropagation algorithm it is important to distinguish between input nodes i, output nodes j and hidden nodes n. The backpropagation especially concerns itself with the difference between t and y. The change to a weight W_i for a hidden node is defined as:

$$\alpha * s'(a(X_p, o_n)) * d(n) * X_p(i, o_n) \tag{9.5}$$

where d is a function that determines how much the hidden node contributes to the overall error. d returns numbers close to zero or zero if it doesn't influence the error of the NN or numbers close to one if it is highly influential.

The contribution to the error of an output node j is represented by $d(j)$ and the influence a hidden node has on an output node is represented by the function $W(n,j)$. As any hidden node n may influence any output node j the actual influence of the hidden node can be written as $\sum d(j) * W(n,j)$ for all j. The training rules with backpropagation can then be divided into two parts:

1. The changes in weight between any hidden node n and output node j: $\alpha * s'(a(p,n)) * (t(p,n) - y(p,n)) * X_p(n,j)$
2. The change of weight between and input node i and an output node j: $\alpha * s'(a(p,n)) * (\sum d(j) * W(n,j)) * X_p(i,n)$
[3]

9.2.2 Types of neural networks

The following sections will describe the different variations of NNs.

Convolutional neural networks

Convolutional neural networks (*CNN*) are a certain type of NN especially suited for image classification as they have been inspired by the animal visual cortex. The CNN consists of multiple layers. While the layers at the beginning are extracting basic features of an image like vertical or horizontal lines are the layers towards the end of the network responsible to combine the simple features to more complex attributes. Each layer of the CNN has a specific type that are elaborated in detail below.

One of them is the *convolutional* layer that applies a function to its input. The convolution layer takes it input and separates it into a grid. Afterwards the node applies a filter - a matrix of arbitrary size - to the input and returns the output to the next node.

Another type is the *pooling* layer. The pooling layers are used to reduce the amount of features from the previous node. A common example of a pooling layer is a max pooling layer. The max pooling layer also divides it input into sections and returns the biggest number of the section to next layer. An example is illustrated below:

$$\begin{bmatrix} 33 & 14 & 46 & 68 \\ 22 & 16 & 33 & 20 \\ 56 & 54 & 75 & 17 \\ 40 & 51 & 82 & 58 \end{bmatrix} \rightarrow \begin{bmatrix} 33 & 68 \\ 56 & 82 \end{bmatrix} \tag{9.6}$$

An example for a feature map with an applied filter that extracts vertical lines can be seen below in figure 9.2.

The last special type is the *fully connected layer*. As the name suggests this layer connects every input node with every output node. A figurative example is the Figure 9.1 where each layer is fully connected to its previous input layer[4].

9.3 Object Detection with the You only look once (*YOLO*) Approach

You only look once (*YOLO*) is a new approach to the field of object detection with the help of machine learning. YOLO is distinct to other machine learning algorithms as the general

[3]IBM, *An introduction to neural networks*.
[4]IbM, *Deep learning architectures*.

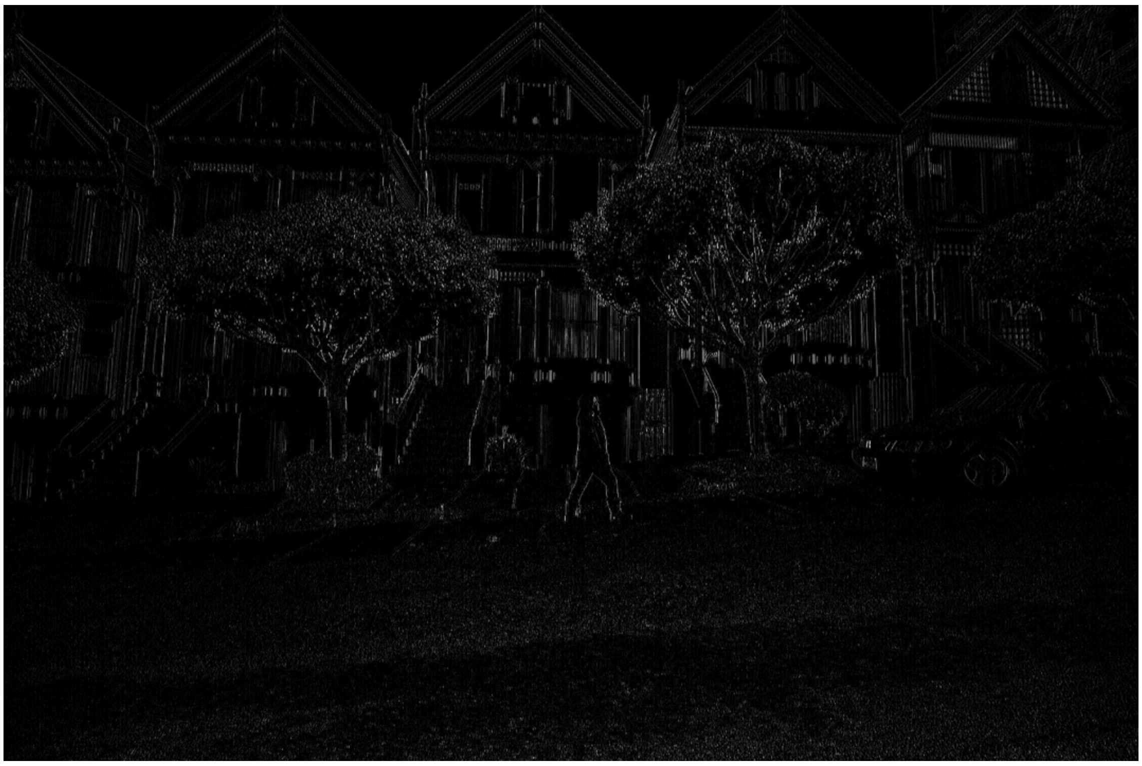

Figure 9.2: An image after it has passed through a pooling layer. The filter of the pooling layer has extracted the vertical lines (white) whereas horizontal lines have been darkened.

algorithm is different. YOLO is modeled after humans. Humans are capable to look at an image and can relatively easy detect multiple objects in a short amount of time. YOLO uses a similar approach and combines two different activities in one step. The convolutional neural network (*CNN*) that powers the YOLO algorithm is only allowed to take one look at a given picture. It then predicts the image classes and possible object bounding boxes in one step and then combines the result of these two activities to generate the final object classifications. As the CNN looks at the picture as a whole unlike other approaches like regional proposal networks it is possible for the neural network to minimize false detection in the background as the CNN is capable to understand the context of the whole picture.

9.3.1 How YOLO works

This section will elaborate the general algorithm used in the YOLO approach in greater detail.

The system separates the image into a grid consisting of the same amount of rows and columns. Each field in the grid then predicts a varying number of bounding boxes and a confidence of how certain it is, that the bounding box is accurate and contains an object. So formally the confidence can be defines as following:

$$confidence = P(O) + IoU^{truth}_{prediction} \tag{9.7}$$

Whereas $P(O)$ represents the probability that there is an actual object in the cell and

$$IoU^{truth}_{prediction} = \frac{truth \cup prediction}{truth \cap prediction} \tag{9.8}$$

represents the accuracy of the predicted bounding box. If there is no object in a cell the confidence should be equal to zero. Each predicted bounding box consists of several predictions: The x and y coordinates of the box relative to the grid cell, a *height* and a *width* relative to the whole image, the *confidence* as defined above and a class probability $C = P(Class_i|Object)$.

9.3.2 Drawbacks of YOLO

Although YOLO is a promising algorithm for the area of object detection it also imposes some major drawbacks. The biggest of them are the big hardware demands as the algorithm requires and NVidia GForce Titan X to achieve a Frame rate of up to 45 frames per second. The other problem is that the bounding boxes are crutial for the classification of the object. So if there are too many small objects in one grid cell the network can't distinguish each of those objects and may predict wrong classes for those objects[5][6].

9.4 Object detection with Faster R-CNN

Faster R-CNN is a NN designed for object detection and classification. It was introduced in the year of 2016 and builds upon the achievements and experiences of Fast R-CNN. Faster R-CNN combines a region proposal network (*RPN*) and the Fast R-CNN detector that classifies the objects. The RPN is a fully connected CNN and is used to detect regions of interest (*RoI*). The RPN returns beside the proposal of the RoI also the objectness score that indicates how confident the NN is, whether there is an object or not. The Fast R-CNN object detector then looks at the region of interest and tries to classify the objects in the RoI.

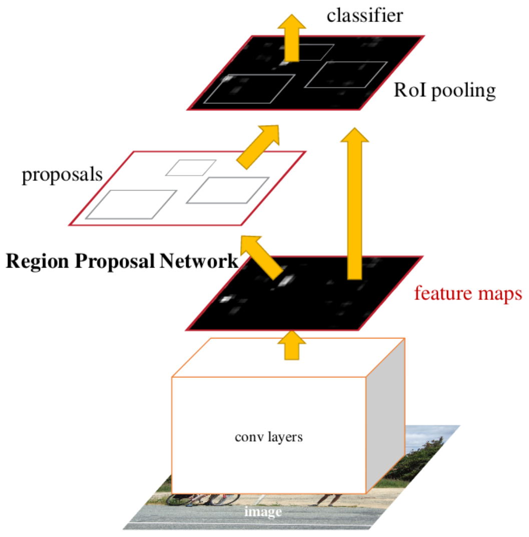

Figure 9.3: A diagram visualizing the workings of Faster R-CNN. The input image gets passed through a CNN that builds a feature map. The RPN uses the feature map to generate the RoI and the last layer then classifies the objects.

[5]Redmon, Divvala, et al., "You Only Look Once".
[6]Redmon and Farhadi, "YOLOv3: An Incremental Improvement".

9.4.1 Drawbacks of Faster R-CNN

The biggest drawback of RPNs and also Faster R-CNN is that the RPN tends to falsely detect objects in the background of the image as it always only looks of part of the image and not the image as a whole. Therefore there tend to be problems with landscapes[7].

9.5 Performance comparison

The following sections will show various performance metrics of the previously discussed implementations of neural networks.

The tests were performed on a Laptop equipped with an Intel Core i7-4700MQ with a frequency of 2.40 GHz and 8 GB of RAM. The test video that lasts 125 seconds was from a robotics competition provided by the robotics team items from the HTL Wiener Neustadt.

9.5.1 Performance of YOLO

This section will asses the performance of an implementation of the YOLO algorithm.

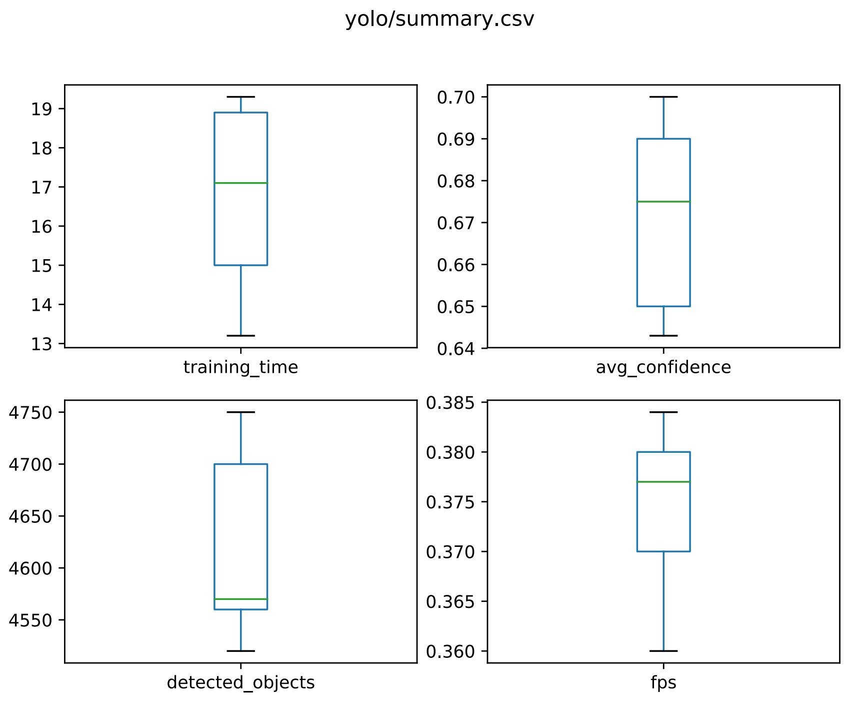

Figure 9.4: Various boxplot showing the performance of the YOLO implementation. The top-left diagram shows training time. The top-right the average confidence and the boxplot visualizing the detected objects. The bottom right shows the frames per second in a boxplot.

The diagram above show that the training time with a mean of 17 seconds is relatively high. The average confidence is very stable at 67.5%. The detected objects are very high with around 4500 to 4700. The frames per second are also relatively high with around 0.375.

[7]Ren et al., "Faster R-CNN".

9.5.2 Performance of RPN

This implementation of an RPN uses Google's machine learning framework TensorFlow to provide the classification.

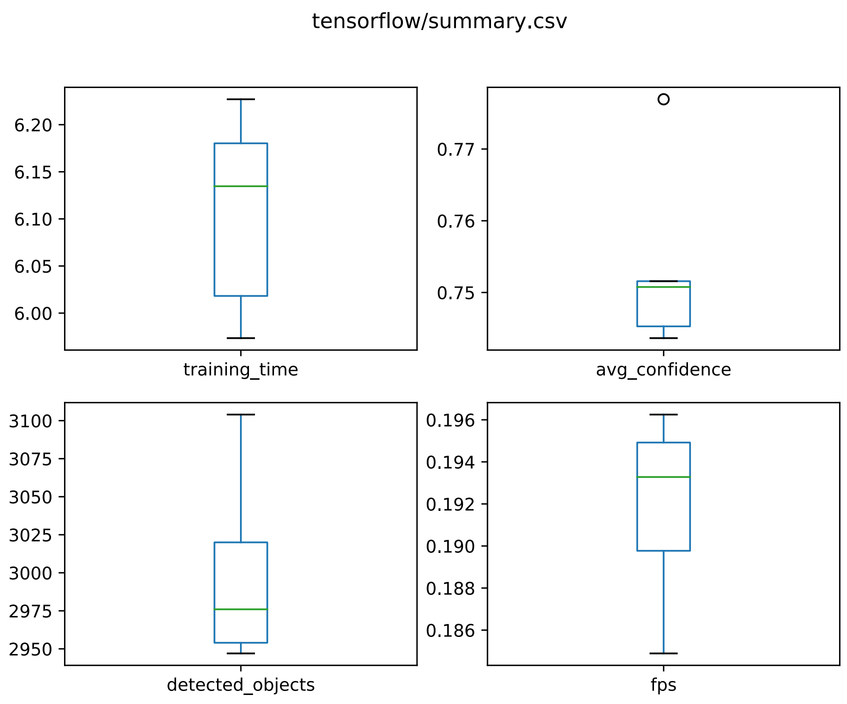

Figure 9.5: Various boxplot showing the performance of the RPN implementation. The top-left diagram shows training time. The top-right the average confidence and the boxplot visualizing the detected objects. The bottom right shows the frames per second in a boxplot.

The performance of YOLO and RPN are significantly different. Whereas YOLO had a long training time with around 17 s it is comparatively short with RPN with around 6 s. RPN is also more confident with its predictions with around 75% confidence. Compared to YOLO RPN hasn't detected as many objects in the video. Only around half as much objects have been detected by the RPN. Compared to YOLO the frames per second are quiet low. The RPN neural network has a maximum of 0.196 frames per second (*fps*) that is less than half of the highest fps of YOLO.

9.5.3 Algorithm for AUT-AS

To decide which algorithm to use for AUT-AS each metric of the algorithm will be weighted by a certain amount and the bigger sum is the winner of the comparison.

The team of AUT-AS has selected the confidence of the NN as the number priority. For that reason the factor has been set the highest. The second most important category are the frames per second to provide the user with a smooth experience even with the added processing strain of the image classification. The detected objects have the least priority in this comparison because in real-world use it is not of such high priority to detect as many objects as possible but rather to be correct with the classifications. Although the training time has been previously mentioned it has been excluded from this table as the impact over the long run of the robot isn't really important.

The Factor is a value assigned to each item based on the priority of the element. The most important aspect has the highest factor and the value gets decreased the lesser important each element gets.

	YOLO			RPN		
Metric	Value	Factor	Product	Value	Factor	Product
Average confidence	67 .435%	3	202.305	75.3638%	3	226.09164
FPS	0.3754	2	0.7509	0.19182	2	0.38364
Detected objects	4618.4	1	4618.4	3000.2	1	3000.2
Sum			4821.4559			3226.675284

Therefore for all further image classification and object detection tasks will be fulfilled with the YOLO algorithm as the result clearly shows that YOLO won with a reasonable margin.

Although the FPS seem very low with less than one, it has to be noted that object recognition is a very computational expensive task that requires a lot of resources. For example the official YOLO paper made their benchmarks with an Nvidia Geforce Titan X which is a lot faster than the one in the laptop where the test was made[8].

[8] Redmon and Farhadi, "YOLOv3: An Incremental Improvement".

Chapter 10

AUT-AS Application

Author: Simon Königsreiter

A sub-goal of this thesis is, to develop an interface for operators to interact with the AUT-AS robotic platform. This chapter outlines the advantages and disadvantages of different approaches for developing cross-platform applications.

10.1 Native Apps

One way to implement the app for AUT-AS would be the native approach. When writing native apps the main benefit is improved performance of the application. The biggest downside on the other hand is an increased development effort, as each mobile platform has its own programming language with Android being Java, Kotlin or C++ and iOS using Objective-C or Swift. This makes the application not very portable across different operating systems. The improved performance gain is especially important in hardware demanding application like mobile games. However, as the hardware requirements for the AUT-AS application aren't high and the enormously increased development effort to maintain several code-bases is too much for one developer to handle, the decision was made to discard this approach[1][2].

10.2 Hybrid Apps

One approach to build cross-platform applications for mobile devices and desktop devices are hybrid apps. These apps function like native apps but instead of the operating system's native language, web languages such as HTML, JavaScript and CSS are used. Because they are not written in the operating system's native language it is necessary to wrap the application in a stripped down browser which can be barely noticed during everyday usage. This has the great advantage, that the mobile apps can be installed from the app store just like every other native app. The biggest downside on the other hand is the relatively low performance compared to native apps and the lack of access to features like the file system. Although it is possible to overcome these barriers, it is not as easy as with native interfaces to the Operating System[3].

[1]Google, *Application Fundamentals*.

[2]Apple Inc., *Develop - Apple Developer*.

[3]1&1, *Hybrid apps – combining the best of web and native apps*.

10.3 Web Apps

The last approach to building cross-platform mobile applications is a web app. This approach utilizes HTML, CSS and JavaScript to build webpages. Web apps have the great advantage that every device which has a web browser can access the app. It also reduces development costs as only one version of the application has to be developed which can be used from every device. Another feature of web apps is the ability to quickly present the newest versions to the consumers as updates are automatically downloaded when loading the web page. However, this also leads to several disadvantages. The biggest problem is the fact that most modern browsers are sandboxed. This means that web applications can't access features of the device unless the browser has implemented a way for the app to use these features[4].

	Native App	Hybrid App	Web App
Language	Platform's native language	HTML, CSS, JavaScript	HTML, CSS, JavaScript
Development effort	High	Medium	Low
Performance	High	Medium	Medium
Portability	Not portable	Partially Portable	Portable

Table 10.1: A table comparing the various app types[5].

Due to the tremendously increased development effort of native apps, this approach been discarded and due to the familiarity of the team with web apps it has been decided to implement the AUT-AS mobile application as a web app.

10.4 Comparison of JavaScript frontend frameworks

To improve the development speed and overall quality of the application the team has decided to use a JavaScript framework to create the app. The following sections will compare the three most popular client side JavaScript (JS) frameworks.

10.4.1 Angular

Angular is an open source framework for mobile and desktop web applications that is developed and maintained by Google. It is in development since 2010 and it's most recent version is version 7. It uses TypeScript which is a superset of JS with static typing. Angular in general is a very opinionated framework which means that the framework makes many decisions for you. This makes it especially easy as every Angular project has the same file structure. As Angular is comparatively the largest framework in terms of built-in features, with some of them beeing routing, seperation of concerns and dependency injection. Angular's performance is quite good but the biggest downside to the framework is it's large file-size compared to some of its competitors.[6]

10.4.2 React

React is a JS user interface library backed by Facebook which is significantly smaller than Angular. React focuses exclusively on the presentation of the application which means it

[4]Salesforce, *Native, HTML5, or Hybrid*.
[5]Mitch Pronschinske, *The State of Native vs. Web vs. Hybrid - DZone Mobile*.
[6]Vue.js, *Comparison with Other Frameworks — Vue.js*.

doesn't include libraries for making HTTP calls to remote services. React has a relatively good performance as it uses a virtual DOM to efficiently update it's elements as efficient as possible. Although it is not efficient because when a change is detected all subcomponents have to be re-rendered. React also makes heavy usage of JSX is an extension for JavaScript which combines HTML and JS more closely. It is a relatively small library and doesn't provide anything but the basics to construct a user interface[7].

10.4.3 VueJS

Vue is a reactive JS client-side framework which is also mainly concerned with the presentation of user interfaces. Unlike the other two frameworks it isn't backed by one large company. It's most recent major version is version 2 which was released in 2016. Vue is very efficient in updating the user interfaces thanks to it's reactivity and the virtual DOM. When the state of one component changes it will automatically identify it's dependencies which makes it very efficient to react to state changes. Although Vue is mostly concerned with the user interface presentation it also has officially supported companion libraries for routing and state management. Vue's size is close to React's[8].

10.5 Description of VueJS

Due to the previously mentioned points and the familiarity of the team members with VueJS the team has decided to use VueJS to implement the user interface. The following sections will describe VueJS thoroughly.

10.5.1 Components

VueJS (Vue) uses so called components to design the user interface. Components are reusable units and can be used just like normal HTML but they can be extended with custom properties and event listeners. Components can be defined in a single file and are therefor called Single-File Components. Single-File Components have the great advantage that everything related to a component can be found at the same place. The following piece of code describes a simple ToDo item in a ToDo List:

```
<template>
    <li>{{ tag }}: <slot></slot></li>
</template>

<script>
    export default {
        props: [ "tag" ],
        data() {
            return {};
        }
    }
</script>
```

Listing 10.1: An example of VueJS' template system.

And to use this component from another component it would look like this:

[7]Vue.js, *Comparison with Other Frameworks — Vue.js*.
[8]Vue.js, *Introduction — Vue.js*.

```
<template>
    <ul>
        <todo-item tag="Shopping">Buy milk</todo-item>
    </ul>
</template>

<script>
    import TodoItem from "TodoItemComponent.vue";

    export default {
        components: {
            "todo-item": TodoItem
        },
        data() {
            return {};
        }
    }
</script>
```

Listing 10.2: An example showing the use of a custom component with VueJS.

The use of components makes it easy to redistribute components as component libraries for other developers to use and it also makes the code more readable as the tag-names (e.g.: *todo-item*) are more descriptive[9][10].

Vue's template tag

The *<template>* tag in Single-File Components is used describe what should be actually rendered for a component. Inside the *<template>* any valid HTML can be used and some extra special things from Vue itself. By default Vue can dynamically bind HTML attributes to dynamic data from the components *<script>* part. This can be achieved with the v-bind directive. There is also a shorthand notation instead of v-bind by simply prefixing the attribute with a colon. Inside Vue's template it is also possible to dynamically insert text with the *mustache* syntax which can be used with to curly braces. This makes it possible to insert text dynamically into the component.

```
<template>
    <a :href="linkTarget">{{ linkText }}</a>
</template>

<script>
    export default {
        data() {
            return {
                linkTarget: "https://google.com/",
                linkText: "Google"
            };
        }
    }
</script>
```

If now one of the properties *linkTarget* or *linkText* would change, Vue would automatically update the values in the link thanks to it's reactive nature[11].

[9] Vue.js, *Single File Components — Vue.js*.
[10] Vue.js, *Components Basics — Vue.js*.
[11] Vue.js, *Components Basics — Vue.js*.

Vue's script tag

The *<script>* tag is used in a Single-File Component to export all the information the template needs to work. Inside the exported object can be various pieces of information:

- *props*: This can be either a list, or an object that describes what properties can be passed to the component. When a list is used to describe the properties only the name can be specified. When an object is used it is also possible to define the data-type, whether the property is required or not or even a validator can be specified that validates the input.
- *data*: This is a function that returns an object which describes the initial state of the component.
- *computed*: These are similar to normal properties but are actually cached functions. Computed instance variables are used in templates to reduce verbosity.

There is a variety of additional options that can be defined in the *<script>* tag[12].

10.6 Vue's companion libraries

Vue comes with two officially developed companion libraries. This section will provide a short introduction for Vue's state management library *Vuex* and the official routing library *vue-router*.

10.6.1 Vuex

In larger Applications it is a difficult problem to manage shared state. The only options to implement shared state without a store pattern is to pass props to the components which is very tedious and poses new challenges when working with sibling elements. Therefor Vue provides the Vuex library which makes it easy to manage shared state. The Vuex store can be injected into every component and each mutation to the store's state has to be commited. This makes it easier to debug applications and it also makes the program easier to manage as each state modification can be explicitly tracked. Components using Vuex can access the global store via the *this.$store* variable. Vuex differentiates between two different kind of state changes: *mutations* and *actions*. Mutations are the only functions that can directly mutate the global state and have to be synchronous. Actions on the other hand can be asynchronous but they can't directly mutate the states and have to invoke mutations instead. Vuex follows Vue's reactive nature and therefor as soon as the stores state changes it will automatically trigger a re-render of all the depending components. For a visualization of VueX please see figure 10.1[13].

10.6.2 vue-router

The second library which is officially supported by Vue is the *vue-router*. When using the router there is a special tag named *<router-outlet>* which indicates that the routed component should be displayed at the given position. Routes can be defined as an array with objects that map the path to some component:

Paths can be referenced with the *<router-link>* tag where the attribute *to* specifies the route. Vue's router also supports dynamic routes. Dynamic elements in a route are prefixed with a colon and can be accessed in the component via the *this.$route.params* object[14].

[12] Vue.js, *API — Vue.js*.
[13] Vue.js, *Vuex Documentation*.
[14] Vue.js, *Vue Router*.

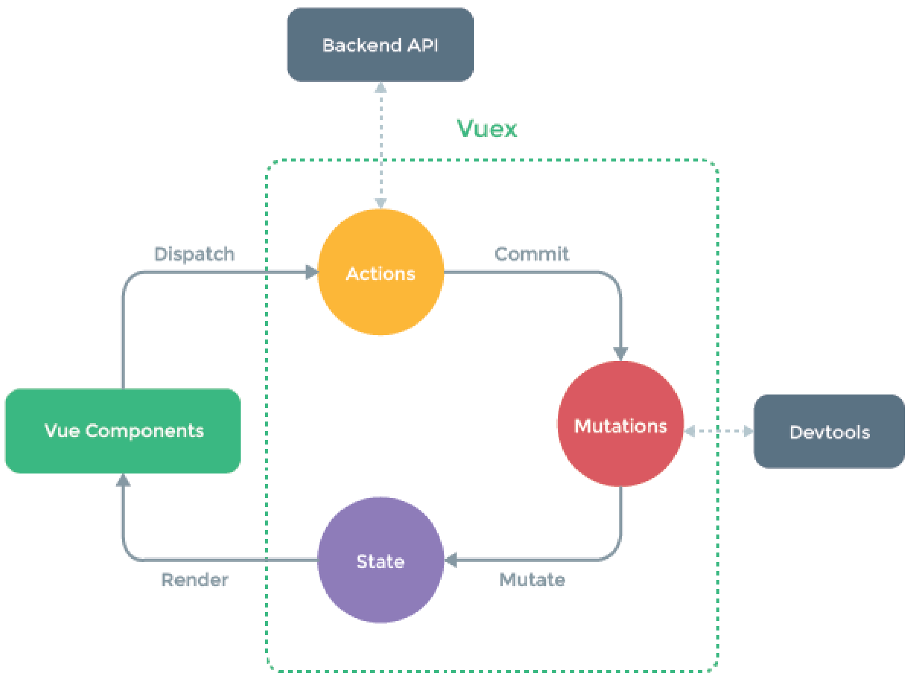

Figure 10.1: A graphic showing the relation of actions, mutations and state with the components.

```
const routes = [
    { path: "/foo", component: FooComponent },
    { path: "/bar", component: BarComponent }
];
```

Listing 10.3: A code snippet defining two routes. The path "/foo" will render the *FooComponent* and the "/bar" path will render the *BarComponent*.

10.7 Exposing diagnostic values of the robot via HTTP

To integrate the AUT-AS platform with external monitoring tools, the team has decided to expose internal metrics via the Hyper Text Transfer Protocol (*HTTP*) and JavaScript Object Notation (*JSON*). HTTP is a widely used protocol in the world wide web that is often combined with JSON as it is tightly integrated with JavaScript. *JSON* has the big advantage over other data formats like the e*X*tensible *M*arkup *L*anguage (*XML*) that it is more readable. An example showing how a mmonitoring response in *XML* could look like is listed below:

The same diagnostic message in the *JSON* format can be found below in listing 10.6.

```
<host name="Ubuntu_Sascha">
    <entry key="Core 1 Temperature" value="0.0DegC" />
    <entry key="Time Since Update" value="3.78194904327" />
    <entry key="Update Status" value="OK" />
    <entry key="Core 0 Temperature" value="47.0DegC" />
</host>
```

Listing 10.4: An imaginary response in XML from diagnostic API

10.7.1 A brief introduction to JSON

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write"[15]. This citation is a good description of JSON. JSON was designed to be written and read by humans but it is also suited for communication between various programming languages. JSON has five fundamental data-types:

- **Object**: An object is a set of key-value pairs each separated by a comma. An object begins with an opening curly brace and ends with an closing curly brace.
- **Array**: An array is a collection of value whose order is fix. It begins with an opening square bracket and ends with an closing square bracket and each value is separated by a comma.
- **String**: A string is surrounded by double quotes and contains an arbitrary number of unicode characters.
- **Number**: A number is a normal number, either an integer or a floating point number.
- **Value**: A value is either one of the previously mentioned types or the value *true*, *false* or *null* which represents no information.

16

10.7.2 Description of the API

The Application Programming Interface (*API*) is exposed through a Python web server that uses the Bottle library. The API exposes four basic HTTP enpoints:

```
/
/<category>
/<category>/<hostname>
/health
```

Listing 10.5: A list of the various routes of the API.

- */*: This is the root path of the API and returns all the available information at the time of the request. It doesn't filter by any criteria.
- */<category>*: This endpoint returns all the available information for the given category *<category>*.
- */<category>/<hostname>*: This endpoint filters by the category first and afterwards it applies the given *<hostname>* as the second filter.
- */health*: This is a very simple endpoint that checks for the general availability of the robot and always returns *UP* as long as the robot is reachable over the network.

[15]**ECMA_JSON_nodate**.
[16]ECMA, *JSON*.

Example

An exemplary call to the API route */CPU_ Temperature* may return the following JSON:

```

{
    "Ubuntu_Sascha": {
        "Core 1 Temperature": "0.0DegC",
        "Time Since Update": "3.78194904327",
        "Update Status": "OK",
        "Core 0 Temperature": "47.0DegC"
    }
}
```

Listing 10.6: An example response in JSON to the diagnostic API

Chapter 11

Conclusion

11.1 Image Classification and Recognition

Author: Simon Königsreiter

Although ROS makes it very easy to integrate image classification with the help of neural networks there are still some problems regarding the different algorithms. The major problem is that the package providing the YOLO implementation uses its own set of messages. This is against the ROS paradigm to use common messages so it is easy to change the implementation depending on the problem. The officially provided TensorFlow object detector uses the de-facto standard vision messages that are used by a lot of projects.

Besides that YOLO is a very fast and accurate object detection algorithm that takes a completely new approach compared to its competitors. But to actively use the image classification especially on the AUT-AS platform it is necessary to provide better hardware as the current system is unusable with the classification system running.

11.2 Interfacing with ROS via the RobotWebTools

The RobotWebTools are a good collection of JavaScript libraries that enables developers to integrate ROS with the world wide web. But there are various problems regarding the RWTs as soon as the application goes beyond the simple publication of messages.

First and foremost is the lack of documentation and the lacking examples that only showcases the simplest types of applications. Another problem is the difficulty to interact and customize with the classes of the library. Especially the provided maps lack the capabilities to customize the output and visuals of the map to integrate it better into the application.

11.3 Festo Robotino - Locomotion

The Festo company has made a smart decision by designing the Robotino as an omnibot. This gives the robot a high degree of maneuverability and through the help of the integrated rotary encoders it is possible to rudimentarily locate the robot relative to its starting position. One downside of the wheels is the fact that they are fixed within the frame and there is no possibility add a suspension to the bot. This creates challenges when the robot has to drive over a doorstep.

11.4 Festo Robotino - Software

Author: Alexander Lampalzer

Overall, working with the Festo Robotino was a very bumpy and rough experience - mainly due to the fact, that the pre-installed operating system was released in 2014, which is at the time of writing 5 years old and 4 generations behind Ubuntu 18. Upgrading this operation system to the newest version manually is very complicated and the latest release by Festo (1.2.4) is still based on Ubuntu 14.04. The performed upgrade to 16.04 allows future use of the containerization software docker. Furthermore, ROS was installed and configured for network-use, which provides a great framework for this diploma thesis and future projects to build upon. Performance wise, it is extraordinary, that the robot contains a fully functional x64 embedded processor. Often, robotics systems build upon an ARM architecture, which introduces a lot of complexity, because of the architectural change and the necessity or re- or cross-compile. To conclude this section, overall the Festo Robotino is a great platform for use in the educational sector, because it provides a wide range of programming interfaces (C, C++, Java, .Net, ROS, LabVIEW, MATLAB, ...) and when upgraded to the newest software, it is capable of running highly sophisticated algorithms. Last, but not least, the robotino has two PCIe extension slots, that could be utilized in the future, when hardware upgrades, such as a graphics processing unit (GPU), are needed.

11.5 Sensors

Overall, the chosen low-cost sensors have provided a great experience. However the utilization of these sensors also hasn't come without flaws. First of all, the YDLIDAR X4 LIDAR sensor is supposedly capable of 12 Hz. In reality, it is not possible to change the rotational frequency with the included components, but rather it is necessary to send certain signals over an unconnected port, according to the documentation, which is non-trivial. All in all, the provided 7 Hz frequency, the accuracy and maximum distance make this sensor an appealing low-cost option for getting started - and this with a relatively low price. Moreover, the ROS community has developed a very capable, but not flawless, driver. Re-writing this driver could be a good introduction for students on understanding and implementing I^2C communication protocols and ROS nodes - for the upcomin mayor release of ROS, ROS2 there is no dedicated driver available, at the time of writing.

When installed properly and after several weeks of debugging and hacking, the Astra Pro depth camera by Orbbec is a great to work with. It provides a very stable experience performance-wise and is suited for a range of difficult environmental situations. It is also capable of displaying the data produced by the IR camera, which is a very bonus. As mentioned earlier, the initial software-experience of this sensor is very bad, because the official ROS drivers do not work out of the box and there is no openly available camera calibration file. For future use, the authors recommend creating a new and separate, open source driver for this camera, to allow not only the HTBLuVA Wr. Neustadt, but also the whole ROS community to profit.

The Microsoft "Kinect for XBOX One" sensor, has outstanding performance characteristics, but is a pain to work with. First of all, both the inbuilt 1080p camera and the 424p depth camera provide a stable 30 frames per second and this with extraordinarily small measurement noise. The included 4 microphones are also quite a nice addition. However, there are a number of concerns, that need to be addressed: Originally, the included USB-B cable transmits power and data at the same time - modern computers and especially laptops rarely come with these ports installed and as such, a USB-A to USB-B adapter is necessary. This however results in the need for another source of electricity and a such it is necessary to solder connectors onto the board and transfer power using an external power supply. Moreover, due to the sheer amount of data transferred, both the receiving PC, the adapter cable need to be capable of USB 3.0 and a cable with extra shielding is recommended. For future use, the authors recommend to re-work and open source the software and provide extensive documentation on how to use this sensor.

11.6 Outlook

As part of this diploma thesis, the AUT-AS platform was developed, which simplifies the development of ground-based mobile robots, that need exploration and navigation capabilities. A quantitative analysis of SLAM algorithms was planned, however obtaining a reliable and accurate ground truth for comparison was more complex than anticipated and the time needed for this experiment was not available. Furthermore, an analysis of exploration algorithms would be beneficial.

During the development of this thesis, the authors noticed, that the development of web-based user interfaces is hindered by a lack of documentation for the "robot web tools". The development of a new abstraction library for web-based user interfaces would be a benefit to the whole ROS community - preferably utilizing either VueJS or React.

Index

Author Index

Literature Index

Bibliography

1&1. *Hybrid apps – combining the best of web and native apps*. en. URL: https://www.ionos.com/digitalguide/websites/web-development/hybrid-apps-combining-the-best-of-web-and-native-apps/ (visited on 10/29/2018).

Agile Alliance. *Agile Manifesto for Software Development*. en-US. June 2015. URL: https://www.agilealliance.org/agile101/the-agile-manifesto/ (visited on 10/10/2018).

Allison, Peter Ray. *What does a bomb disposal robot actually do?* en. News Site. July 2016. URL: http://www.bbc.com/future/story/20160714-what-does-a-bomb-disposal-robot-actually-do (visited on 09/26/2018).

Apple Inc. *Develop - Apple Developer*. en. URL: https://developer.apple.com/develop/ (visited on 10/29/2018).

ar_track_alvar - ROS Wiki. URL: http://wiki.ros.org/ar_track_alvar (visited on 04/01/2019).

Atlassian. *Kanban - A brief introduction*. en. URL: https://www.atlassian.com/agile/kanban (visited on 10/10/2018).

attachment:markers0to8.png of ar_track_alvar - ROS Wiki. URL: http://wiki.ros.org/ar_track_alvar?action=AttachFile&do=view&target=markers0to8.png (visited on 04/01/2019).

Bitzer, Sebastian. *The UKF exposed: How it works, when it works and when it's better to sample*. en. Jan. 2016. DOI: 10.5281/zenodo.44386. URL: https://zenodo.org/record/44386 (visited on 01/20/2019).

Böttcher, Sven. "Principles of robot locomotion". en. In: (), p. 25.

British Army. *Remotely_controlled_bomb_disposal_tool.JPG (JPEG Image, 1024 × 768 pixels)*. URL: https://upload.wikimedia.org/wikipedia/commons/3/37/Remotely_controlled_bomb_disposal_tool.JPG (visited on 10/29/2018).

Christensen, Henrik I. "Robot Locomotion". en. In: (), p. 47.

Christian B. Bellanosa, Ruth Pearl J. Lugpatan, and Diogenes Armando D. Pascua. "Position Estimation using Inertial Measurement Unit (IMU) on a Quadcopter in an Enclosed Environment". en. In: *International Journal of Computing, Communication and Instrumentation Engineering* 3.2 (July 2016). ISSN: 23491477. DOI: 10.15242/IJCCIE.AE0516306. URL: http://iieng.org/images/proceedings_pdf/AE0516306.pdf (visited on 03/07/2019).

Clark, Chris. "ARW – Lecture 01 Odometry Kinematics". en. In: (), p. 50.

ECMA. *JSON*. URL: http://json.org/ (visited on 01/05/2019).

Eitel, Elisabeth. *Basics of Rotary Encoders: Overview and New Technologies*. May 2014. URL: https://www.machinedesign.com/sensors/basics-rotary-encoders-overview-and-new-technologies-0 (visited on 11/01/2018).

Foote, Tully. "tf: The transform library". In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. Apr. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373.

Foote, Tully and Mike Purvis. *Standard Units of Measure and Coordinate Conventions*. en. Oct. 2010. URL: http://www.ros.org/reps/rep-0103.htm (visited on 02/27/2019).

Goepfert, Jessica and Michael Shirer. *Worldwide Spending on Robotics Forecast to Accelerate Over the Next Five Years, Reaching $230.7 Billion in 2021, According to New IDC Spending Guide*. English. Dec. 2017. URL: https://www.idc.com/getdoc.jsp?containerId=prUS42880417 (visited on 09/20/2018).

Google. *Application Fundamentals*. en. URL: https://developer.android.com/guide/components/fundamentals (visited on 10/29/2018).

IBM. *An introduction to neural networks*. en-US. Aug. 2018. URL: https://developer.ibm.com/articles/l-neural/ (visited on 01/02/2019).

IbM. *Deep learning architectures*. en-US. Sept. 2017. URL: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/ (visited on 01/05/2019).

Internatioinal Federation of Robotics. *Global industrial robot sales doubled over the past five years*. en. URL: https://ifr.org/ifr-press-releases/news/global-industrial-robot-sales-doubled-over-the-past-five-years (visited on 10/29/2018).

iRobot. *Robot Vacuuming, Robot Mopping & Outdoor Maintenance | iRobot Online Store*. en. Product Page. URL: https://kaufen.irobot.at/homepage?lang=de_AT&_ga=2.123462155.1427023341.1538056315-1140317942.1538056315 (visited on 09/27/2018).

Meeussen, Wim. *Coordinate Frames for Mobile Platforms*. en. Oct. 2010. URL: http://www.ros.org/reps/rep-0105.html#id8 (visited on 10/30/2018).

Michie, D, D J Spiegelhalter, and C C Taylor. "Machine Learning, Neural and Statistical Classification". en. In: (), p. 298.

Mitch Pronschinske. *The State of Native vs. Web vs. Hybrid - DZone Mobile*. en. June 2014. URL: https://dzone.com/articles/state-native-vs-web-vs-hybrid (visited on 10/30/2018).

Mordechai, Ben-Ari. *Elements of robotics*. en. New York, NY: Springer Berlin Heidelberg, 2017. ISBN: 978-3-319-62532-4.

Moulard, Thomas. *Coordinate Frames for Humanoid Robots*. en. Nov. 2011. URL: www.ros.org/reps/rep-0120.html (visited on 02/27/2019).

N. Shimkin. "Kinematic Models for Target Tracking". en. In: *Estimation and Identification in Dynamical Systems*. Fall 200. 048825. Technion – Israel Institute of Technology, Department of Electrical Engineering, 2009. URL: http://webee.technion.ac.il/people/shimkin/Estimation09/ch8_target.pdf (visited on 04/01/2019).

Open Source Robotics Foundation. *Gazebo : Sensor Noise*. en. URL: http://gazebosim.org/tutorials?tut=guided_i3 (visited on 10/29/2018).

— *Messages - ROS Wiki*. en. URL: http://wiki.ros.org/Messages (visited on 10/04/2018).

— *msg - ROS Wiki*. en. URL: http://wiki.ros.org/msg (visited on 10/04/2018).

— *Nodes - ROS Wiki*. en. URL: http://wiki.ros.org/Nodes (visited on 10/04/2018).

— *razor_imu_9dof - ROS Wiki*. URL: http://wiki.ros.org/razor_imu_9dof (visited on 03/07/2019).

— *ROS packages*. en. URL: http://www.ros.org/browse/list.php (visited on 09/27/2018).

— *Services - ROS Wiki*. en. URL: http://wiki.ros.org/Services (visited on 10/04/2018).

— *Topics - ROS Wiki*. en. URL: http://wiki.ros.org/Topics (visited on 10/04/2018).

— *urdf - ROS Wiki*. en. URL: http://wiki.ros.org/urdf (visited on 10/29/2018).

Orbbec 3D. *Astra Series*. en-US. URL: https://orbbec3d.com/product-astra-pro/ (visited on 03/07/2019).

Peter H. Salus. *A Quarter-Century of Unix*. en-US. 1994. ISBN: 0-201-54777-5.

Project Management Institute. *Don't throw the baby out with the bathwater*. en. 2007. URL: https://www.pmi.org/learning/library/combine-agile-traditional-project-management-approaches-7304 (visited on 10/10/2018).

Redmon, Joseph, Santosh Divvala, et al. "You Only Look Once: Unified, Real-Time Object Detection". en. In: *arXiv:1506.02640 [cs]* (June 2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640 (visited on 03/27/2019).

Redmon, Joseph and Ali Farhadi. "YOLOv3: An Incremental Improvement". en. In: (), p. 6.

Ren, Shaoqing et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". en. In: *arXiv:1506.01497 [cs]* (June 2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497 (visited on 03/27/2019).

Salesforce. *Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options - developer.force.com.* en. URL: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options (visited on 10/30/2018).

Siegwart, Roland, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots.* Second. MIT Press, 2011.

Smith, Michael. "Calls to honour inventor of bomb disposal device". en-GB. In: (Apr. 2001). ISSN: 0307-1235. URL: https://www.telegraph.co.uk/news/uknews/1316277/Calls-to-honour-inventor-of-bomb-disposal-device.html (visited on 09/26/2018).

The chrony authors. *Comparison of NTP implementations.* en. Oct. 2018. URL: https://chrony.tuxfamily.org/comparison.html (visited on 03/07/2019).

Uhlmann, J., S. Julier, and H.F. Durrant-Whyte. "A new method for the nonlinear transformation of means and covariances in filters and estimators". en. In: *IEEE Transactions on Automatic Control* 45.3 (Mar. 2000), pp. 477–482. ISSN: 0018-9286. DOI: 10.1109/9.847726.

University of Stanford. *Robotic Nurses | Computers and Robots: Decision-Makers in an Automated World.* en-US. URL: https://cs.stanford.edu/people/eroberts/cs181/projects/2010-11/ComputersMakingDecisions/robotic-nurses/index.html (visited on 09/27/2018).

US Department of Commerce, National Oceanic and Atmospheric Administration. *What is LIDAR.* EN-US. URL: https://oceanservice.noaa.gov/facts/lidar.html (visited on 10/30/2018).

Vue.js. *API — Vue.js.* en. URL: https://vuejs.org/v2/api/ (visited on 10/31/2018).

— *Comparison with Other Frameworks — Vue.js.* en. URL: https://vuejs.org/v2/guide/comparison.html#Angular-Formerly-known-as-Angular-2 (visited on 10/30/2018).

— *Components Basics — Vue.js.* en. URL: https://vuejs.org/v2/guide/components.html (visited on 10/31/2018).

— *Introduction — Vue.js.* en. URL: https://vuejs.org/v2/guide/ (visited on 10/30/2018).

— *Single File Components — Vue.js.* en. URL: https://vuejs.org/v2/guide/single-file-components.html (visited on 10/31/2018).

— *Vue Router.* URL: https://router.vuejs.org/ (visited on 11/01/2018).

— *Vuex Documentation.* URL: https://vuex.vuejs.org/ (visited on 11/01/2018).

Willow Garage. *PR2 Hardware Specs.* en. URL: http://www.willowgarage.com/pages/pr2/specs (visited on 03/06/2019).

YDLIDAR - X4. URL: http://www.ydlidar.com/product/X4 (visited on 03/07/2019).

Zanuttigh, Pietro et al. "Operating Principles of Structured Light Depth Cameras". In: *Time-of-Flight and Structured Light Depth Cameras: Technology and Applications.* Cham: Springer International Publishing, 2016, pp. 43–79. ISBN: 978-3-319-30973-6. DOI: 10.1007/978-3-319-30973-6_2. URL: https://doi.org/10.1007/978-3-319-30973-6_2.

— "Operating Principles of Time-of-Flight Depth Cameras". en. In: *Time-of-Flight and Structured Light Depth Cameras: Technology and Applications.* Ed. by Pietro Zanuttigh et al. Cham: Springer International Publishing, 2016, pp. 81–113. ISBN: 978-3-319-

30973-6. DOI: 10.1007/978-3-319-30973-6_3. URL: https://doi.org/10.1007/978-3-319-30973-6_3 (visited on 10/31/2018).