
DIPLOMARBEIT

Autonomous Aerial Reconnaissance Drone

Ausgeführt im Schuljahr 2018/19 von:

Entwicklung zur Automatisierung des Flugverfahrens einer Drohne
Konstantin LAMPALZER

5BHIF

Entwicklung grafischer Benutzeroberfläche, Hindernisvermeidung
Matthias GRILL

5BHIF

Betreuer / Betreuerin:

MMag. Dr. Michael Stifter

Wiener Neustadt, am 4. 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 4. April 2019

Verfasser / Verfasserinnen:

Konstantin LAMPALZER

Matthias GRILL

Contents

Eidesstattliche Erklärung	i
Acknowledgement	vi
Diplomarbeit Dokumentation	vii
Diploma Thesis Documentation	ix
Kurzfassung	xi
Abstract	xii
1 Introduction	2
1.1 Goal	2
1.2 Motivation	3
1.3 Challenges	3
1.3.1 Fragility of Drones	3
1.3.2 Obstacle Detection and Avoidance	4
1.3.3 Consistency	4
1.4 Autonomy	4
1.4.1 History of Autonomy	4
1.4.2 Evolution of autonomous robots	5
1.4.3 Machine learning in autonomous robots	5
2 Project Management	6
2.1 Kanban	6
2.1.1 Visuals	6
2.1.2 Columns	6
2.1.3 Work In Progress Limits	6
2.1.4 Commitment point	7
2.1.5 Delivery point	7
2.2 History of Kanban	8
2.3 Meetings with our advisor	8
2.4 Working hours	8
3 Robot Operating System	9
3.1 Core Concepts	9
3.1.1 Packages	9
3.1.2 Nodes	9
3.1.3 Topics	10

3.1.4	Services	10
3.1.5	Messages	11
3.1.6	Master	11
3.1.7	Bags	11
3.1.8	Parameters	11
3.1.9	Transform	12
3.2	Community	12
3.3	Docker	12
3.4	Simulation	13
3.5	Visualization	14
3.5.1	RVIZ	14
3.5.2	rqt	15
4	ArduPilot	16
4.1	Hardware	16
4.2	Firmware	16
4.3	Ground Control Software	17
4.3.1	Mission Planner	17
4.3.2	QGroundControl	17
4.4	MAVLink	17
4.4.1	Telemetry	18
5	PX4	19
5.1	Flight Modes	19
5.1.1	Manual Flight Modes	19
5.1.2	Assisted Flight Modes	19
5.1.3	Auto Flight Modes	20
5.2	QGroundControl	20
5.3	Simulation	21
5.3.1	Supported Simulators	21
6	Sensors	22
6.1	Altimeter	22
6.2	Accelerometer	23
6.3	Satellite Navigation Systems	24
6.4	Gyroscope	25
6.5	Filtering	25
6.5.1	Low-pass filter	26
6.5.2	Moving Average Filter ¹	26
6.5.3	Moving Median Filter	26
6.5.4	Exponential Filter ²	27
6.5.5	Spike Filter	28
6.5.6	Summarizing	28
7	Pathfinding	30
7.1	Concept	30
7.2	Applications of Pathfinding	31

¹Smith, *Digital Signal Processing*.

²Standards, (U.S.), and SEMATECH, *NIST-International Sematech E-handbook: NIST Handbook 151. Dataplot : NIST Handbook 148*.

7.2.1	Pathfinding in Computer Games	31
7.2.2	Pathfinding in Exploration	31
7.2.3	Pathfinding in industrial applications	32
7.3	Pathfinding Algorithms	33
7.3.1	Breadth First Search	33
7.3.2	Dijkstra's Algorithm	33
7.3.3	A* search algorithm	34
7.3.4	Pathfinding in 3D environments	35
7.3.5	Efficiency of pathfinding algorithms	35
7.4	Conclusion	35
8	Graphical User Interface	37
8.1	Introduction	37
8.2	Choice of platform	37
8.3	Vue.js	37
8.4	Rosbridge	38
8.5	AARD GUI	39
8.6	Docker	40
9	AARD Application	41
9.1	Introduction	41
9.2	Mission Planning	41
9.2.1	Waypoints	42
9.2.2	Commands	43
9.3	Flight modes	43
9.4	AARD Mission	44
9.4.1	Initialization	44
9.4.2	Arming	44
9.4.3	Takeoff	44
9.4.4	Mission start	45
9.4.5	Intermediate goal	45
9.4.6	Calculating the interception point	45
9.4.7	Land	48
9.5	Challenges and Problems	50
9.5.1	GPS	50
9.5.2	Waypoints	52
9.5.3	Speed matching	52
9.5.4	Camera control	53
9.5.5	Simulation versus Reality	54
9.6	Testing	54
10	Conclusion	56
10.1	Development	56
10.1.1	Docker	56
10.1.2	Robot Operating System	57
10.1.3	Arducopter versus Px4	57
10.1.4	Simulation	57
10.2	AARD Application	58
10.2.1	AARD Front-End	58
10.2.2	AARD Back-End	58

10.3 Outlook	58
Index	60
Bibliography	64

Acknowledgement

First, we would like to thank everybody who has supported us throughout this diploma thesis.

Most notably, we would like to thank our supervisor MMag. Dr. Michael Stifter, who always had time for us and always gave us valuable advice. Furthermore, we would like to thank Dipl.-Ing. Harald Haberstroh for his amazing support over the last 4 years.

Another notable mention goes to the Austrian Armed Forces for the good cooperation and for allowing us to do this thesis.

Lastly, we would like to thank our families and friends, who always believed in us.

Diplomarbeit Dokumentation

Namen der Verfasser/innen	Konstantin Lampalzer Matthias Grill
Jahrgang Schuljahr	5BHIF 2018 / 19
Thema der Diplomarbeit	Autonomous Aerial Reconnaissance Drone
Kooperationspartner	F-WuTS, BMLV - ARWT

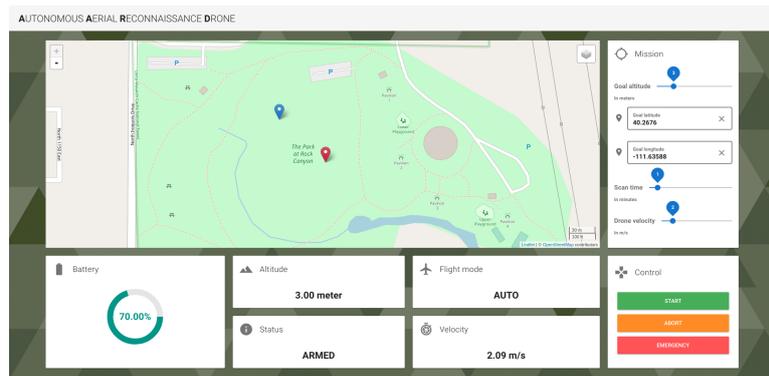
Aufgabenstellung	Erstellung eines Programms, Testung, Vorführung und Dokumentation zur Automatisierung des Flugverfahrens einer gefechtstechnischen Drohne von und zurück zu einer beweglichen Plattform, zu einem dynamischen Ziel, nachfolgendem Schwebeflug und entsprechender Darstellung auf einem Schirm.
------------------	--

Realisierung	Realisierung mithilfe von ROS, Python und Docker im Backend Visualisierung mittels Vue.js im Frontend
--------------	--

Ergebnisse	Funktionsfähiges Backend Funktionsfähiges Frontend
------------	---

Typische Grafik, Foto
etc. (mit Erläuterung)

Benutzeroberfläche der AARD-Software



Teilnahme an
Wettbewerben,
Auszeichnungen

Möglichkeiten der
Einsichtnahme in die
Arbeit

HTBLuVA Wiener Neustadt
Dr.-Eckener-Gasse 2
A 2700 Wiener Neustadt

Approbation

Prüfer

Abteilungsvorstand

(Datum, Unterschrift)

MMag. Dr. Michael Stifter

AV DI Felix Schwab

Diploma Thesis Documentation

Authors	Konstantin Lampalzer Grill Matthias
Form	5BHIF
Academic Year	2018 / 19
Topic	Autonomous Aerial Reconnaissance Drone
Co-operation partners	F-WuTS, BMLV - ARWT

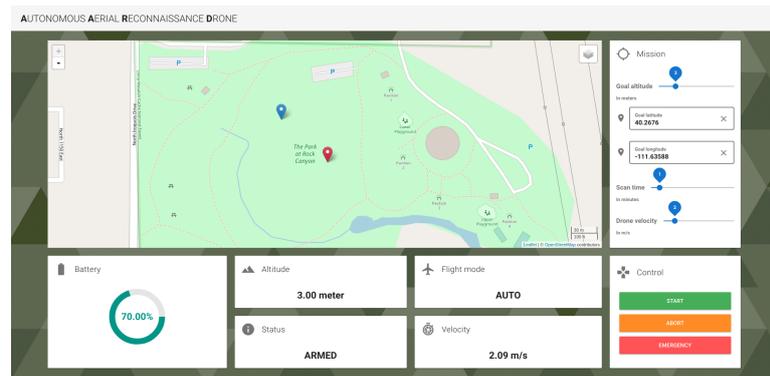
Assignment of tasks	Creation of a program, testing, demonstration and documentation to automate the flight procedure of a combat drone from and back to a moving platform, to a dynamic target, subsequent hover and display on a screen.
---------------------	---

Realization	Realization using ROS, Python and Docker in the Backend Visualisation using Vue.js in the Frontend
-------------	---

Results	Working Backend Beatiful Frontend
---------	--------------------------------------

Illustrative graph, photo
(incl. explanation)

User interface of the AARD-Software



Participation in
competitions,
Awards

Accessibility of diploma
thesis

HTBLuVA Wiener Neustadt
Dr.-Eckener-Gasse 2
A 2700 Wiener Neustadt

Approval

Examiner

Head of Department

(Date, Sign)

MMag. Dr. Michael Stifter

AV DI Felix Schwab

Kurzfassung

In den letzten Jahren haben Drohnen immer mehr an Popularität gewonnen. Ein Grund dafür sind die Einstiegskosten, welche stark gesunken sind. Aufnahmen aus der Luft können dadurch bereits von jedem leicht gemacht werden, dies sieht man zum Beispiel in großen Städten, wo Touristen mit ihren kleinen und kompakten Drohnen herumfliegen, um wunderschöne Aufnahmen zu erstellen. Mit dieser kommenden Technologie war das Ziel der Diplomarbeit, die autonomen Fähigkeiten von solchen Drohnen, im mittleren bis hohen Preisbereich, zu erforschen.

Des Weiteren lag der Fokus nicht nur auf der Erforschung autonomer Drohnen, sondern auch, wie solch eine Drohne autonom auf einem fahrenden Fahrzeug landen kann. Mehrere Proof-of-Concepts, welche Lösungen für dieses Problem bieten, wurden im Rahmen dieser Diplomarbeit programmiert. Zur schnelleren Entwicklung und Testung wurde zusätzlich eine Simulationsumgebung für Drohnen eingerichtet.

Außerdem vergleichen die Autoren die beiden gängigsten Flight Stacks *ArduCopter* und *PX4*, analysieren ihre autonomen Fähigkeiten und zeigen Probleme auf, denen sie während der Arbeit begegnet sind. Um zu verstehen, wie diese fliegenden Fahrzeuge autonom in der Luft navigieren können, wurde zusätzliche Zeit aufgewandt, um die grundlegenden Prinzipien im Bereich der Sensoren zu erforschen, welche in Luftfahrzeuge integriert sind. Da ein wichtiger Teil der Navigation von Drohnen auch die Hindernisvermeidung ist, setzte einer der beiden Autoren den Fokus auf die Pfadplanung mit Drohnen und wie es möglich ist ein Hindernis während des Fluges auszuweichen.

Am Ende wurde all dieses Wissen genutzt, um das AARD-Projekt (Autonomous Aerial Reconnaissance Drone) zu entwickeln, ein Proof-of-Concept Softwarepaket, mit dem Drohnen autonom auf einem fahrenden Fahrzeug landen können. Im Rahmen dieses Projekts wurde eine Vielzahl moderner Technologien eingesetzt, wie beispielsweise das *Robot Operating System (ROS)* und *Docker*. Durch den Aufbau des ROS-Ökosystems und der Verwendung der Docker-Container-Technologie erstellten die Autoren eine Software, die auf einer Vielzahl von Maschinen in verteilten Systemen ausgeführt werden kann. Darüber hinaus wurde mit Hilfe von *Vue.js* und *Rosbridge* eine benutzerfreundliche grafische Oberfläche erstellt, welche zum Steuern der Drohne verwendet wird. Abschließend wurde eine umfassende Dokumentation für die nächste Generation von Schülern bereitgestellt, welche es ihnen ermöglicht, innerhalb kürzester Zeit eine Entwicklungsumgebung mit einem voll funktionsfähigen Simulator, inklusive ROS-Integration, aufzubauen.

Abstract

Aerial drones are becoming more and more popular, as the entry costs sink. Aerial photography can now be done by everyone, as we can see with a rising number of tourists flying drones in the big cities. With this new technology on the uprise, the goal of this project was set to research the autonomous capabilities of such mid- to high-tier drones.

Therefore, a focus was set on researching, how one could enable drones to autonomously land on a moving vehicle. Multiple proof-of-concepts have been programmed as part of this project and a simulation environment was set up for rapid development and testing.

Furthermore, the authors compare the two common flight stacks *ArduCopter* and *PX4*, analyze their autonomous capabilities and show problems they ran into. In order to understand how these flying vehicles can navigate the skies, additional time was spent on researching the fundamental principles in the field of sensors, which are integrated into aerial vehicles. Furthermore, one author put a focus on path planning and how a drone can avoid an obstacle while in-flight.

In the end, all this knowledge is used to develop the Autonomous Aerial Reconnaissance Drone (AARD) project, a proof-of-concept software package to autonomously land a drone on a moving vehicle. As part of this project, a wide variety of modern technologies were used, such as the *Robot Operating System (ROS)* and *Docker*. By building on the ROS ecosystem and by utilizing the Docker container engine, the authors created software, that can be run on a wide variety of machines and in distributed systems. Furthermore, *Vue.js* and *Rosbridge* was utilized to create an easy-to-use user interface, enabling even inexperienced users to control the drone. Finally, extensive documentation has been provided for the next generation of students, which lets them set up a development environment with a fully fledged simulator with ROS integration in no time.



A A R D

Chapter 1

Introduction

Author: Konstantin Lampalzer

Transportation is evolving rapidly. Self driving cars are invading our streets and space rockets are autonomously landing back on our planet. On the contrary to these great inventions, traffic congestion is rising, as the human population increases. Looking at this, many companies are investing vast amounts of money into research regarding alternative transportation methods. For example Hyperloop One¹, with their pods inside a vacuum tube, or the Boring Company² creating tunnels underneath our cities. However, not only passengers need to be transported. Goods play another key role in our economy and therefore companies, like Amazon³ or DHL⁴ are investing into alternative means of hauling goods. They have already shown examples using unmanned aerial vehicles to deliver packets to the customer from a warehouse or car nearby.

These are just few examples of what's awaiting us in the near future, as companies are revolutionizing the way we think about transportation, with the automation of systems and processes playing a major role. Hence, investing into robotics and autonomy is now more important than ever before.

1.1 Goal

The goal of this diploma thesis is to develop a software that enables drones to autonomously navigate in outdoor environments, while avoiding obstacles during flight. The process starts with the user specifying multiple waypoints. This input can be made by either giving GPS coordinates or by simply clicking points on a map shown on the screen. Following setup, the drone takes off and starts to navigate along a calculated path through all waypoints. Using sensors on the aerial vehicle, the software automatically detects obstacles obstructing route and plans a new path avoiding the obstacle. Once the last point is reached, the drone hovers above the target for a predefined duration and then automatically returns to the starting location. Additionally, an interface is provided to dynamically change the waypoint coordinates during flight. This can be used to take off or land on a moving platform. Many challenges arise from this task, as multiple complex systems have to work together in order to achieve this goal.

¹Hyperloop One, *Hyperloop One*.

²The Boring Company, *The Boring Company*.

³Amazon.com, Inc., *Amazon Prime Air*.

⁴Deutsche Post AG, *DHL Parcelcopter*.

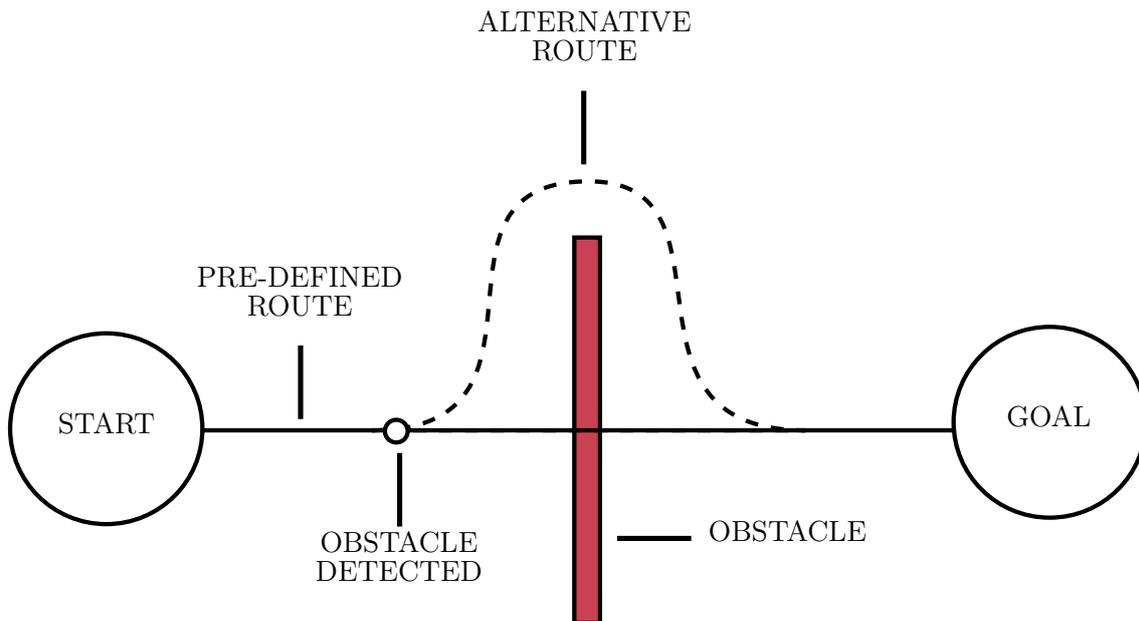


Figure 1.1: Shows an example how a drone flight may look like. It takes off from the start and follows a pre-defined route. During flight it detects an obstacle and calculates an alternative route around the obstacle to reach the goal successfully.

1.2 Motivation

At the time of writing this thesis, the Austrian Armed Forces manually review image data collected by aerial photography and mark potential hazards by hand. Afterwards, an explosive specialist gets dispatched to those locations and needs to verify the safety hazard. As the rate of false positives is quite high, the military actively searches for new ways of minimizing the risk to personnel, accelerating and automating the elucidation of potential dangers.

1.3 Challenges

1.3.1 Fragility of Drones

One major flaw presented by drones is their fragility. Therefore, all systems have to be monitored continuously and backup plans have to be made, in case something goes wrong. For example, when a motor breaks down, the vehicle needs to detect the failure and automatically deploy a parachute to descend safely, in order to keep the payload safe and secure. Another challenge is the big distance, that drones can move away from their control station. This is problematic, as communication between the user and the aerial vehicle is crucial. Therefore, backup strategies have to be in place, when communication breaks off or gets interrupted.

1.3.2 Obstacle Detection and Avoidance

Obstacle detection and avoidance presents another very complicated challenge, because of the the large amounts of data generated by the different types of sensors. Afterwards collection, the data needs to be transferred between a drone and the control station. Furthermore, this vast amount needs to be analyzed and processed with low latency, in order to avoid crashing into an obstacle, because of the high speed of the vehicles. Moreover, recognizing an obstacle during flight also presents a challenge, as some sensors are unable to detect some objects. For example, lasers go trough transparent objects and therefore it's impossible to detect them.

1.3.3 Consistency

Providing the same result every time is another difficult task. After the operator declares the waypoints, the drone has to execute everything autonomously. This is very challenging, as every execution takes place in different conditions. Even tough the route stays the same, the weather conditions might have changed, or the wind speed might have picked up. Therefore, it is nearly impossible to create a fully deterministic robot.

1.4 Autonomy

1.4.1 History of Autonomy

Automation of processes started in the 18th century with the invention of the steam engine by Thomas Newcomen. This point marks the beginning of the industrial revolution, when humanity began to replace human labor with machines. Starting with this event, automa-tion grew continuously, with another mayor event being the first use of welding robots in the car industry. With assembly lines and robots, production was faster and cheaper than ever before. Forwarding to this century, autonomy presents a major discussion point, with subjects like Internet of Things (IoT), Cloud Technology and Big Data changing the present and shaping the future of industry and humanity.

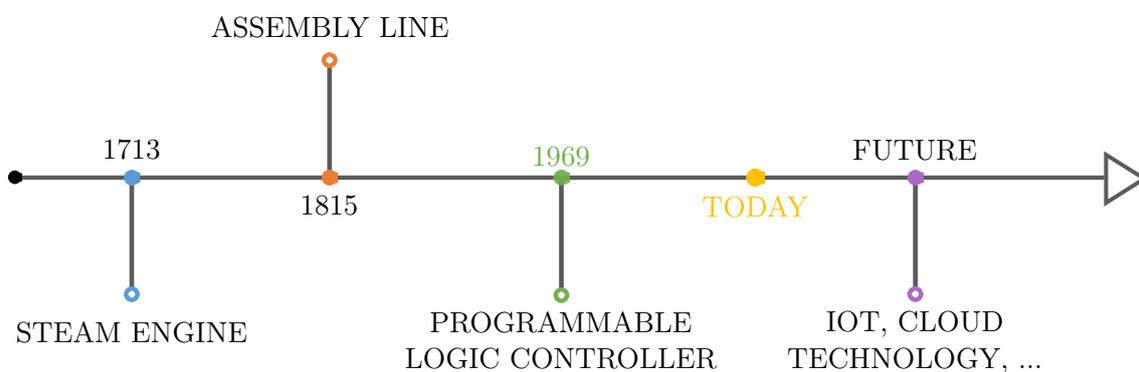


Figure 1.2: Shows the history of autonomy in a time-line. Starting from the industrial age with the invention of the steam engine, continuing with the first assembly line, and finally ending with future predictions.

1.4.2 Evolution of autonomous robots

Autonomy is a versatile subject with applications in many industries. Ranging from basic tasks like vacuuming a house or mowing the lawn to really difficult ones, such as aiding a surgeon in a medical operation. This trend is only possible due to major developments in microprocessors in the last decade, as processors became smaller and more powerful. Without this, robots would not be as compact as today. This growth was predicted in 1965 by Gordon E. Moore⁵, who said, that the amount of transistors on a microprocessor doubles every two years.

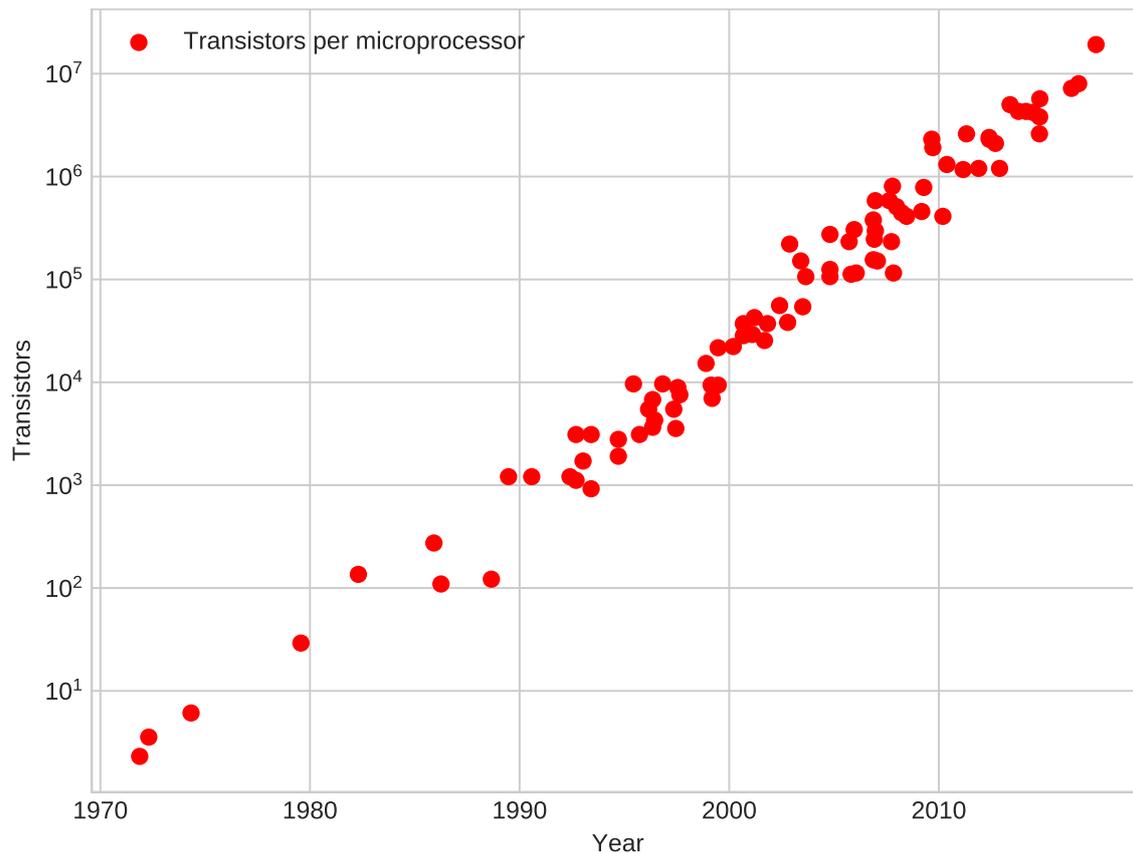


Figure 1.3: Shows the development of transistors per microprocessor. As predicted by Moore's law, the amount of transistors doubles every two years. For better visualization the graph uses a logarithmic scale.⁶

1.4.3 Machine learning in autonomous robots

Another big influence on autonomous robots today, is the increasing usage of machine learning in combination with autonomous robots. Machine learning can be broadly defined as "computational methods using experience to improve performance or to make accurate predictions"⁷. Dynamically making predictions from experience is really important, as it is nearly impossible to create one algorithm for every situation. For example, in self driving cars, you can't create one program that works on every road on the planet.

⁵Moore, *Cramming More Components Onto Integrated Circuits*.

⁶Rupp, *42 Years of Microprocessor Trend Data*.

⁷Mohri, Rostamizadeh, and Talwalkar, *Foundations of Machine Learning*.

Chapter 2

Project Management

Author: Konstantin Lampalzer

2.1 Kanban

Project management is very important in order to efficiently plan, execute and deploy a project in short time. Therefore, we chose a Kanban Board for managing our project. We decided on Kanban, because it is really easy to implement it for small groups, in our case two people. Kanban is Japanese and can roughly be translated as Billboard or Sign. In this chapter we will focus on the following 5 core concepts:

2.1.1 Visuals

Every task or User Story is written down on a visual card (For example a sticky note). These cards then get placed on a board in their respective columns, This helps visualizing the current status of a project and makes it easy to identify bottlenecks. Additionally, we decided to color-code our stickers, in order to show the person assigned to a task¹.

2.1.2 Columns

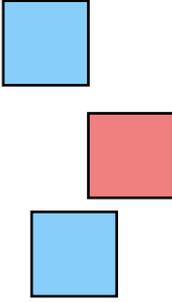
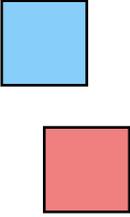
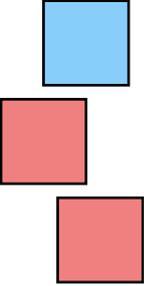
The board is structured in multiple columns, every column represents the current status of a task. This helps the team to easily identify the status of every task. Furthermore, it makes sure that every task passes all columns and therefore everything goes through testing. As Kanban does not specify columns, we chose the structure described in figure 2.1 for our project.

2.1.3 Work In Progress Limits

The work in progress limits² describe the maximum amount of cards, that can be in one column simultaneously. This is important, as it enables the team to prioritize on a specific groups of tasks. Therefore, a team member can focus on his own tasks, instead of getting flooded with work. Furthermore, it helps to understand the weak points in the system. If a column is always full, more team members might be needed, for example.

¹Atlassian, *Kanban - A brief introduction* / Atlassian.

²Radigan, *What are WIP limits?*

	Backlog	Todo	In Progress	Done
				
WIP Limits	/	4	2	/
Description	Initially all tasks get placed here	Tasks in this column will be performed next	Current tasks we are working on	Finished tasks are positioned here

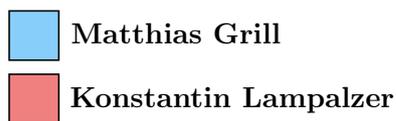


Figure 2.1: This graphic shows an example of our Kanban board. The filled squares represents a task of one team member, depending on the color. Moreover, you can see our WIP limits and the column descriptions of our board.

2.1.4 Commitment point

As explained before, all tasks initially get placed in the backlog. When a team-member moves a task from the backlog to the next column, the team commits to working on this task. This is called a commitment point³. After this, a task can not be altered by the customer anymore. This makes it easier for a team to adapt to changes before starting a task.

2.1.5 Delivery point

The Delivery Point⁴, is the column of the board, where a task gets moved after it has been done. It is called Delivery Point, because the product or feature get delivered to the customer here. One goal of Kanban is to reduce the time it takes, getting one task from the Commitment point to the Delivery point. This duration is called "Lead Time". Another goal is to always have a working software after a task gets past the delivery point.

³Atlassian, *What is a Kanban Board?*

⁴Atlassian, *What is a Kanban Board?*

2.2 History of Kanban

Kanban was invented in the 1940s by Taiichi Ohno, an engineer at Toyota. The company had problems with keeping the right amount of inventory for the current demand. By visualizing everything on a board and communication between the members Toyota was able to solve this problem and created the first version of Kanban. In the year 2007 David J. Andersen and his team developed Kanban, how it is used today. Even now, Kanban is a wide spread agile development method used in software development, because it is really easy to implement it in a team and the benefits are tremendous⁵.

2.3 Meetings with our advisor

We tried to conduct a weekly meeting with our supervisor. First we informed him about the status quo, checked if we accomplished our goals from the week before and planned the next goals for the upcoming week. This made sure that we had continuous progress and additionally helped us to spot problems in the development early on.

2.4 Working hours

Name	Working hours
Konstantin Lampalzer	180
Matthias Grill	180

Table 2.1: Working hours per person

⁵LeanKit Inc., *What is Kanban?*

Chapter 3

Robot Operating System

Author: Konstantin Lampalzer

The Robot Operating System (ROS) is a framework for creating robotics applications. It combines features like a communication infrastructure, conventions and different tools to create a powerful instrument that aids in creating software for many robotics platforms and operating systems.

3.1 Core Concepts

3.1.1 Packages

Every software in ROS is organized in packages¹. A package is the smallest unit of build and release. It aims to only contain enough functionality to be useful. The goal is to combine packages with different functions in order to fulfill the purpose of a robot. Packages are usually released using Debian packages. On the file-system all packages are generally located in one workspace, with one folder for every package. Many packages tend to follow a common structure:

```
packageName
├── package.xml: ..... Contains general information about the package, such as name,
│   │           authors, version, ....
├── CMakeLists.txt: ..... File for the CMake build system used for building software
│   │           packages.
├── msg/: ..... Contains message descriptions.
├── launch/: ..... Contains launch files, used for starting multiple nodes at once.
├── src/: ..... Source folder containing the software code.
└── srv/: ..... Contains service descriptions.
```

3.1.2 Nodes

Nodes² are one of the most essential parts of the Robot Operating System. Nodes can be versatile, they subscribe information from other nodes, process data or publish new information to other nodes. Short, a node takes data, either from a physical sensor, or another node, processes this data and then publishes it or controls actuators. In our case,

¹Open Source Robotics Foundation, Inc., *Packages - ROS Wiki*.

²Open Source Robotics Foundation, Inc., *Nodes - ROS Wiki*.

we implemented multiple nodes. One is responsible for generating a point cloud with sensor information from a depth camera. Then we have some other nodes processing this point cloud and generating a map. Following we have multiple nodes taking this map and creating a path through the environment and finishing it, we have software taking this path and controlling the drone.

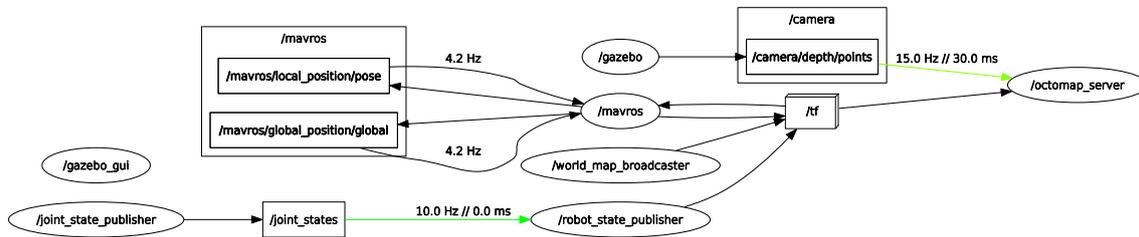


Figure 3.1: Shows the ROS computation graph. Rectangles represent the topics and the ellipses are the nodes.

3.1.3 Topics

Topics³ are used for communicating between nodes. One major feature is the anonymous publishing of information and subscribing to this data from another node. These nodes do not know who they are talking to. This means, that nodes can work under the Fire-and-Forget mentality. If one node needs to know specific information, it can easily subscribe to a topic and doesn't have to worry about where the data is coming from. This enables nodes to be independent of each other, as the data just needs to be present in a serialized format, without caring about who is providing the data.

The Robot Operating System provides the command-line tool *rostopic*⁴ for displaying information about Topics. It provides following functionalities:

```

rostopic
├─ bw [topic] ..... display bandwidth used by topic
├─ echo [topic] ..... print messages of topic to screen
├─ hz [topic] ..... display publishing rate of topic
├─ info [topic] ..... print information about the topic
├─ list ..... display list of active topics
└─ pub [topic] [type] [args] ..... publish data to topic
  
```

3.1.4 Services

In contrast to topics, services⁵ are used for requests and replies between nodes and clients. If a client sends a request message to a node, the call is blocking, meaning that the client will wait till he gets a response from the node. One big advantage of services is the awareness of each other, so one node knows who it is talking to. In contrary to topics, services are based on a Request-Response principle.

The command-line tool *rosservice*⁶ can be used to interact with services:

```

rosservice
├─ call [service] [args] ..... display bandwidth used by topic
  
```

³Open Source Robotics Foundation, Inc., *Topics - ROS Wiki*.

⁴Ken Conley, *rostopic - ROS Wiki*.

⁵Open Source Robotics Foundation, Inc., *Services - ROS Wiki*.

⁶Ken Conley, *rosservice - ROS Wiki*.

```

├ info [service] ..... print information about a service
├ list [topic] ..... list active services

```

3.1.5 Messages

ROS nodes communicate with each other by publishing messages⁷ to a topic. These messages are defined in the `msg` folder of a package with one file for every message type. The Robot Operating System (ROS) provides a message description language used in these files. In order to enable multiple ROS tools to convert these files into different programming languages. Messages are versioned by creating a MD5 hash of the message definition file. A node can only publish or subscribe to a topic, if the MD5 hash and the message type match.

3.1.6 Master

The ROS Master⁸ is responsible for managing the communication between all nodes. He is aware of which nodes are available and which are subscribing or publishing data from topics or services. Moreover, the master is accountable for publishing the recognized information to all other nodes in ROS. Right after the nodes found each other, with the help of the ROS Master, they build up a peer-to-peer connection for communication.

3.1.7 Bags

Rosbags⁹ are comparable with recordings, a bag subscribes on one or more topics in a given time and saves the received information, including a time stamp, in a bag file. Now the data in rosbags can be easily reproduced in ROS for debugging or testing. In our specific case, we have thousands of sensor data in just one test scenario. So if we want to reproduce a certain case at a specific time, for finding the error in our program, we load our bag file with the help of rosbag into ROS. Now we can simply start debugging our software.

3.1.8 Parameters

The Robot Operating System provides a way to dynamically configure nodes using parameters¹⁰. These can be read from a node during runtime and provided by the parameter server. This server is run by the master and can be accessed via `rosparam`¹¹ command line tool. This enables developers to change the settings of a program on the fly. Additionally, parameters can be specified before starting a node using a launch file.

```

1 <launch>
2     <node name="odomProviderArdrone" pkg="libaerial4you" type="odomProviderArdrone"
3         clear_params="true" output="screen">
4         <param name="odomPub" value="ardrone/odometryProcessed" />
5         <param name="odomSub" value="ardrone/odometry" />
6     </node>
7 </launch>

```

⁷Open Source Robotics Foundation, Inc., *Messages - ROS Wiki*.

⁸Open Source Robotics Foundation, Inc., *Master - ROS Wiki*.

⁹Open Source Robotics Foundation, Inc., *Bags - ROS Wiki*.

¹⁰Open Source Robotics Foundation, Inc., *Parameter Server - ROS Wiki*.

¹¹Ken Conley, *rosparam - ROS Wiki*.

3.1.9 Transform

As robots typically have multiple 3D coordinate-systems, a system is needed to keep track of those frames over time. ROS provides this functionality via the transform library¹². It enables the user to transform points, vectors, etc... between multiple coordinate systems at any point in time. This helps developers to easily calculate the relationship between two objects. ROS stores transform in a tree-structure and communication of transforms is implemented via the `/tf` topic.

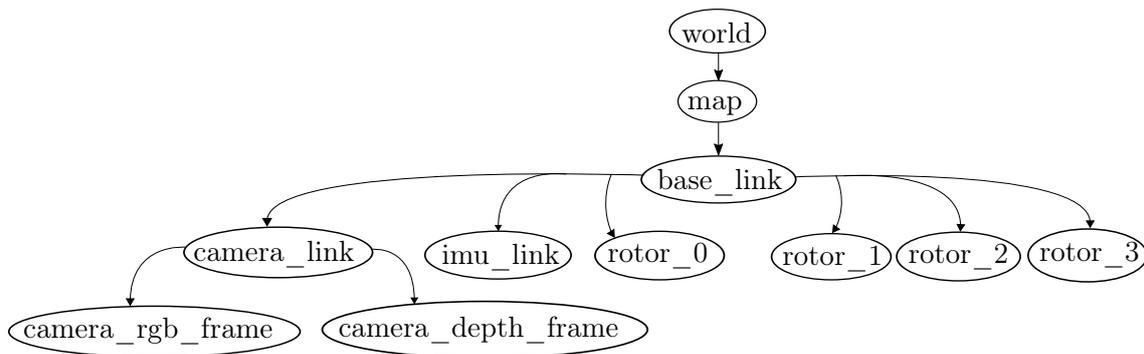


Figure 3.2: Shows the transform tree representing the relationships between the world, the drone and the depth camera.

3.2 Community

One big benefit of using ROS is the large community providing open source software. For example, TU Darmstadt provides a big software-stack called hector with packages such as hector-slam or hector-quadrotor, which is used in controlling UAVs. Furthermore, the ROS industrial consortium is an open source project with the aim of improving manufacturing and automation. Additionally, ROS provides a large list of open-source packages on their Website¹³.

3.3 Docker

We decided to use Ubuntu 18.04¹⁴ as our development operating system, because it's easy to use and it has a large amount of software, that can easily be downloaded. Furthermore, ROS recommends using a Linux-based operating system. This presented some challenges to us, as some packages were not supported under Ubuntu 18.04 with ROS-Melodic. Therefore, we decided to use Docker¹⁵ for virtualizing some packages we used. Docker is a software, that enables so called "containerization". It is used to run "containers", that are isolated from each other, which can communicate through pre-defined channels. This helps us to provide the perfect environment for our ROS nodes and packages, while still having the ability to use different versions of ROS and multiple operating systems at the same time. Additionally, this enables us to develop on multiple computers at the same time, as we only need to set up the correct environment for a package once. To manage all our containers

¹²Tully Foote, Eitan Marder-Eppstein, and Wim Meeussen, *tf2 - ROS Wiki*.

¹³<http://www.ros.org/browse/list.php>

¹⁴Canonical Ltd., *Ubuntu 18.04.1 LTS (Bionic Beaver)*.

¹⁵Docker Inc., *Docker*.

we used Docker Compose¹⁶. Compose is a tool that enables us to start multiple containers with one command. First, we created Dockerfiles¹⁷ for all our packages. Then we made one docker-compose file, that described our environment. Finally, to start all our ROS-Nodes, we just need to run "docker-compose up".

3.4 Simulation

"A simulation is the imitation of the operation of a real-world process or system over time."¹⁸ Simulators are one of the most important tools in the library of a developer. It allows him to rapidly design, develop and test robots. Furthermore, Simulators will be even more important in the future, to train artificial intelligence (AI) models with realistic data. Therefore, ROS provides the gazebo_ros_pkgs¹⁹ interface to communicate with the Gazebo²⁰ simulator. Multiple physics engines, a sophisticated 3D Graphics engine and extensibility through plugins are only some features provided by the simulator. We used the simulator to test our Drone in an realistic environment. Furthermore, to simulate the firmware running on the drone, we used the Software in the Loop (SITL) simulator²¹ provided by ArduPilot. To put our robot into the Gazebo simulator, we first had to create an XML file describing our robot. There we defined all our rotors, sensors and physical properties of our Quadcopter. We added a Inertia Measurement Unit, artificial GPS and a depth camera to our drone, to make it as realistic as possible.

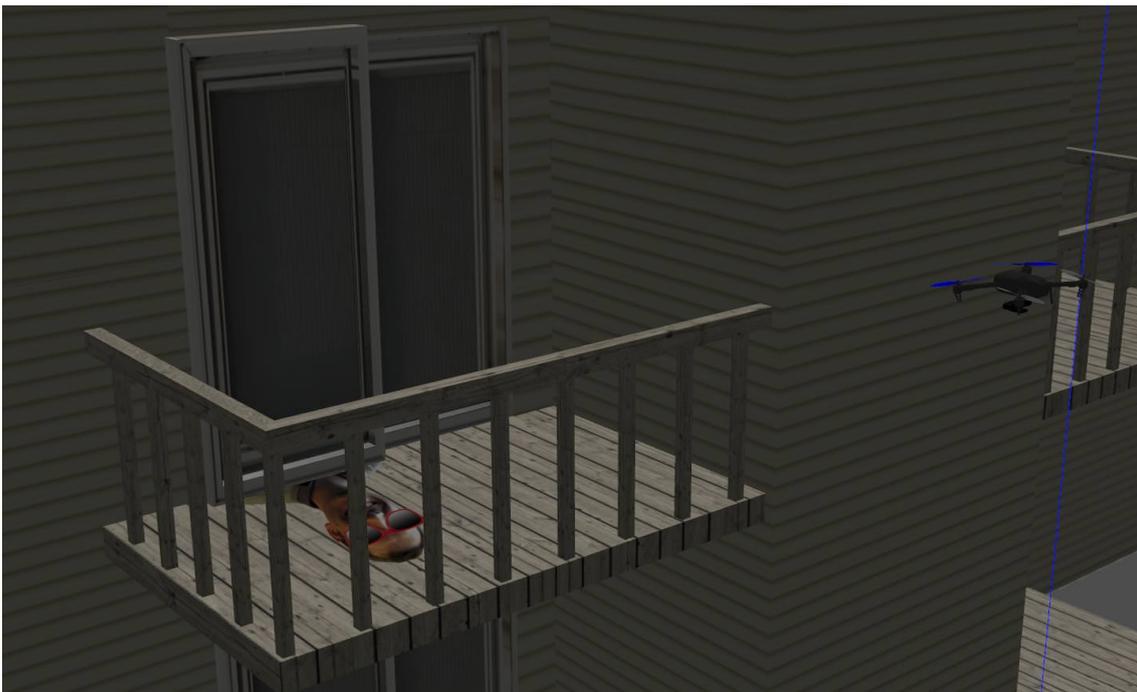


Figure 3.3: Shows the Gazebo simulator with our simulated drone looking at an apartment complex. A depth camera is mounted at the bottom of the vehicle providing data to ROS.

¹⁶Docker Inc., *Docker Compose*.

¹⁷Docker Inc., *Dockerfile reference*.

¹⁸Banks, Carson, and Nelson, *Discrete-event System Simulation*.

¹⁹John Hsu, Nate Koenig, and Dave Coleman, *gazebo_ros_pkgs - ROS Wiki*.

²⁰Open Source Robotics Foundation, Inc., *Gazebo*.

²¹ArduPilot Dev Team, *SITL Simulator (Software in the Loop)*.

3.5 Visualization

Visualization is becoming more important every year. Sectors such as medicine, science, education or robotics use it on a daily basis. Collecting sensor data from a robot and generating graphics from this can be a very useful tool for a developer. Graphics are often used in debugging a software, saving many hours of work in developing an application. Developing on a robot can be very difficult, without knowing exactly, how the robot sees his environment. Debugging, by looking at numbers in a file, can be very difficult, unless you have an intuitive understanding of vectors, coordinate frames and quaternions. Therefore, taking this data and visualizing it in a graph, for example, can cut down on the development time tremendously.

The Robot Operating System help developers in visualizing, by providing two tools out of the box:

3.5.1 RVIZ

RVIZ²² is a 3D visualizing tool for robots enabling us to look at the world through the robot's eyes, whether those eyes are cameras, lasers or other sensors. RVIZ can display the most common types of sensor and state information out of the box by using specialized displays. These are some data types, that can be visualized:

- Point clouds
- Camera data
- Laser scans
- Odometry poses

Additionally, RVIZ gives developers the ability to show primitives, like cubes, arrows and lines in the 3D-World by using visualization markers. This combination of sensor data and custom markers make RVIZ a powerful tool for the development, as you can understand, what the robot is seeing, thinking and doing.

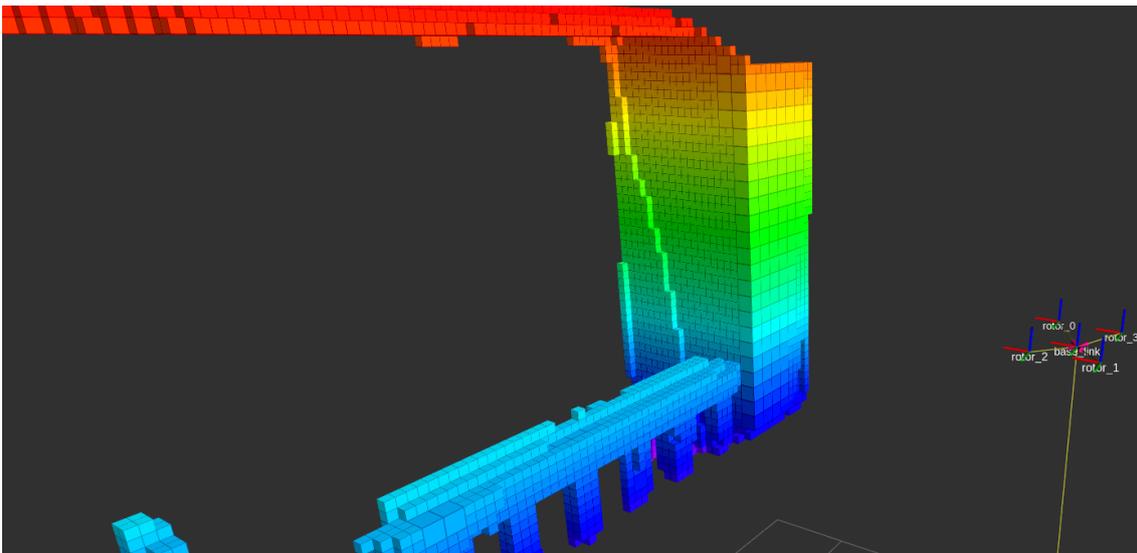


Figure 3.4: Shows an example of a visualization in RVIZ. The drone uses a depth camera, mounted on the bottom side for scanning a house wall.

²²Dave Hershberger, David Gossow, and Josh Faust, *rviz - ROS Wiki*.

3.5.2 rqt

Interaction with a user is another important part of developing a product. Therefore, ROS provides `rqt`²³, a Qt-based framework, for GUI development. A developer can create his own plugins, or use some existing ones to create his GUI. This can be done, by running the plugins as dockable windows within `rqt`. These are some plugins provided by `rqt`, used to visualize information:

```
| rqt_plot ..... Generates a 2D plot of scalar values
├─ rqt_tf_tree ..... GUI plugin for visualizing the ROS TF frame tree
├─ rqt_pose_view ..... Visualizes 3D poses
└─ rqt_image_view ..... Displays images
```

²³Dirk Thomas, Dorian Scholz, and Aaron Blasdel, *rqt - ROS Wiki*.

Chapter 4

ArduPilot

Author: Matthias Grill

ArduPilot¹ is an open source autopilot software which is developed by professionals and enthusiasts. One big advantage of the ArduPilot firmware is the compatibility with many vehicle systems, no matter if conventional airplanes, hexacopters or submarines. This makes ArduPilot one of the leading software-stacks used in developing remotely controlled robots. Currently, the ArduPilot firmware supports 4 different platforms:

- ArduCopter (Multicopters)
- ArduPlane (Planes)
- ArduRover (Ground vehicles and boats)
- ArduSub (Underwater vehicles)

ArduPilot needs 3 key components to transform almost any machine into an autonomous vehicle:

4.1 Hardware

In order to send signals to motors or servos, ArduPilot supports a large variety of sensors, which can be used to calculate the current status of the vehicle. This includes, but is not limited to, Accelerometer, Gyrometers, Barometers, GPS and many more. All this data gets collected by a controller, which runs the firmware. To suit everyone, many open- and closed-source options are available.

4.2 Firmware

The firmware collects all the data provided by the sensors and a user in order to make a calculated decision on how to control the outputs such as a motor or servos. This firmware is compatible with a variety of controllers and provides a unified software for different robots. It uses various algorithms, such as an extended Kalman filter to predict the state of a vehicle and act accordingly. ArduPilot is installed in over 1 000 000 vehicles world-wide and provides different tools to log data and simulate a vehicle. Additionally, as ArduPilot is open-sourced on Github, mistakes can easily be spotted and corrected by community members. Furthermore, it can be audited to ensure compliance with security requirements.

¹ArduPilot, *ArduPilot Open Source Autopilot*.

4.3 Ground Control Software

A good ground control software (GCS) is the counterpart to the ArduPilot firmware. It offers necessary features for the user, like setting up, configuring, testing or tuning an Unmanned Aerial Vehicle (UAV). With an advanced GCS you can create autonomous missions, which is essential for this thesis. There are many GCS's available on the market, the most common ones are Mission Planner and QGroundControl.

4.3.1 Mission Planner

Mission Planner² is an open-source application. It is one of the GCS's, as it is maintained by ArduPilot and officially recommended for use development environments. The biggest disadvantage of this software is its compatibility with Windows only. We are using Linux for developing our project because, it is highly recommended to use ROS in an Linux environment. Consequently, if we want to use Mission Planner, we have to reboot our system. Obviously, this stops us from using Mission Planner and ROS at the same time.

4.3.2 QGroundControl

Another ground control software is QGroundControl³, which provides almost the same features as Mission Planner. In contrary to Mission Planner, QGroundControl supports many platforms, such as Android, Linux or Windows. Additionally, mission planning is more user-friendly and supports a large variety of options, which can be used to autonomously fly a multi-copter.

4.4 MAVLink

MAVLink⁴ is a messaging protocol designed for communication between different drones and ground-control-stations and on-board hardware. It uses the publish-subscribe and point-to-point design patterns. Therefore, data is transmitted via topics, while configuration information is communicated point-to-point. MAVLink provides a reference message set⁵, which is implemented in most ground control stations. These message sets are defined in XML files, which are used to generate MAVLink libraries for all supported programming languages. These are some key-features of MAVLink:

1. Adaptability: MAVLink supports a large amount of programming languages and micro-controllers.
2. Efficiency: As MAVLink only needs a 8 byte of overhead, it can be used in environments with limited bandwidth.
3. Reliability: MAVLink has been used in a wide variety of applications. Therefore, it is field-tested and can be relied upon. Furthermore, it includes systems to detect dropped or corrupted packets.

²ArduPilot Dev Team, *Mission Planner Home*.

³QGroundControl, *QGroundControl*.

⁴Dronecode Project, *MAVLink*.

⁵Dronecode Project, Inc., *MAVLINK Common Message Set*.

4.4.1 Telemetry

Telemetry is one of the most important systems in an unmanned aerial vehicle (UAV). It gives the user the ability to communicate with his drone and remotely operate it. In our thesis we decided to use a 3DR Telemetry Radio v2⁶ together with the Pixhawk 1 flight controller⁷. We chose this radio, because it is a relatively cheap option with a maximum range of 500 meters, which was enough for us. This range can change, depending on your surroundings. One important part in choosing a radio, is checking the regional laws, as most countries regulate the allowed radio frequencies⁸. In our case we are using the low power device 433 MHz (LPD433) ultra high frequency (UHF) band which allows a license free communication in Europe. LPD433 provides 69 channels and a minimum frequency of 433.075 MHz to a maximum frequency of 434.775Mhz.

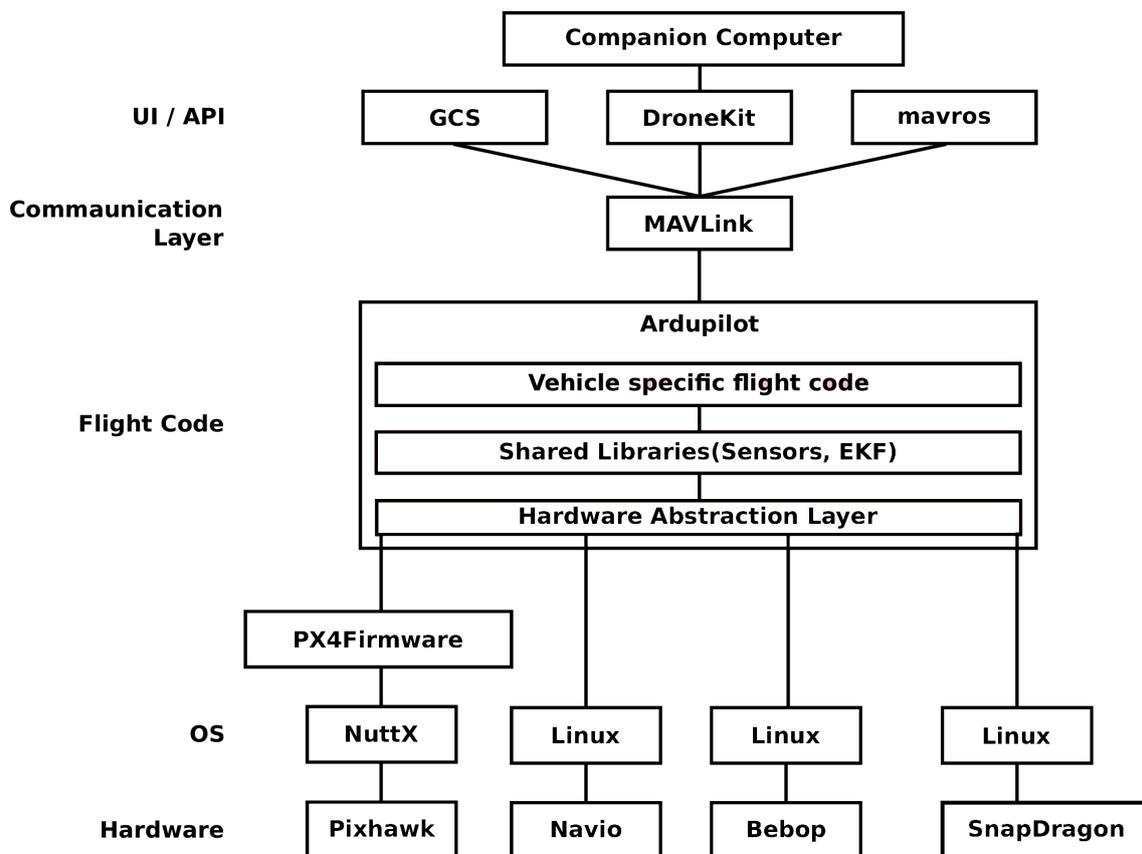


Figure 4.1: Describes the structure of an ArduPilot controlled drone. At the bottom, are the flight controllers with their Operating System. They communicate to ArduPilot via the Hardware Abstraction Layer (HAL). The HAL creates an interface between ArduPilot and the board-specific features of a flight controller. Afterwards, all data from the controller gets put into the shared libraries. Those libraries are shared amongst the four vehicle types supported by ArduPilot and include sensor drivers, estimation and control code. Finally, the vehicle-specific code is the firmware for the four types of vehicles.

⁶ArduPilot Dev Team, *SiK Telemetry Radio*.

⁷PX4 Dev Team, *Pixhawk 1 · PX4 User Guide*.

⁸ArduPilot Dev Team, *Telemetry Radio Regional Regulations*.

Chapter 5

PX4

Author: Matthias Grill

PX4¹ is an opensource autopilot software for unmanned aerial vehicles. It provides many different tools to develop drone applications. PX4 was first developed in the PIXHAWK project at the ETH Zurich, which is the Swiss Federal Institute of Technology in Zurich, Switzerland². Today this project has more than 300 global contributors and is one of the most popular control software in the drone space. Furthermore, PX4 is part of Dronecode³ which is a project hosted under the Linux Foundation. Further well-known projects which are hosted by Dronecode are MAVLink⁴, QGroundControl⁵ and SDK⁶.

5.1 Flight Modes

The flight mode of an UAV defines the drones behaviour. For each task there is usually a specific flight mode. PX4 categorises all flight modes in three groups⁷:

5.1.1 Manual Flight Modes

As the name implies, if the drone is in a manual flight mode the user can manually control the drone. Mostly this happens via a RC control. But there are small differences between theses flight modes. More experienced people will use a flight mode with direct passthrough to the actuators, while beginners should choose a mode which is less responsive on fast stick-position changes. Following manual flight modes are available in PX4 for multirotors:

- Manual / Stabilized
- Acro
- Rattitude

5.1.2 Assisted Flight Modes

If the drone is in a assisted flight mode the user also has to control the drone via a RC control. The big difference to the manual flight mode is that you will have some automatic

¹Dronecode Project, Inc., *Open Source for Drones*.

²Meier, *The History of Pixhawk*.

³Dronecode Project, Inc., *Dronecode SDK*.

⁴Dronecode Project, Inc., *Introduction · MAVLink Developer Guide*.

⁵Dronecode Project, Inc., *QGROUNDCONTROL*.

⁶Dronecode Project, Inc., *Open source Drone end-to-end solutions*.

⁷Dronecode Project, Inc., *Flight Modes · PX4 Developer Guide*.

assistance. Which leads to better control of your drone. For example, if the UAV has to fly in strong wind. PX4 provides following assisted flight modes:

- AltCtl (Altitude control)
- PosCtl (Position control)

5.1.3 Auto Flight Modes

The last category are the auto flight modes. These modes need no user input so the drone will fly autonomously. Auto flight modes are commonly used for flight missions.

- Auto_Loiter (Loiters around the current position)
- Auto_RTL (Returns to the home position and loiters)
- Auto_Mission (Executes predefined mission)

5.2 QGroundControl

QGroundControl is another popular project of Dronecode. It is a ground control station and can be used to flash the PX4 software onto a flight controller. Moreover, you are able to setup and configure your vehicle. In contrast to the ArduPilot firmware the configuration of a drone is much easier and more beginner friendly. Furthermore, in QGroundControl you see real-time information about the drone, for example the altitude, the speed or the current location which is marked on a map. One of the most important feature of this ground control station is the creation of drone missions. QGroundControl provides a simple user interface (UI) where it is possible to create, plan and save your drone missions.

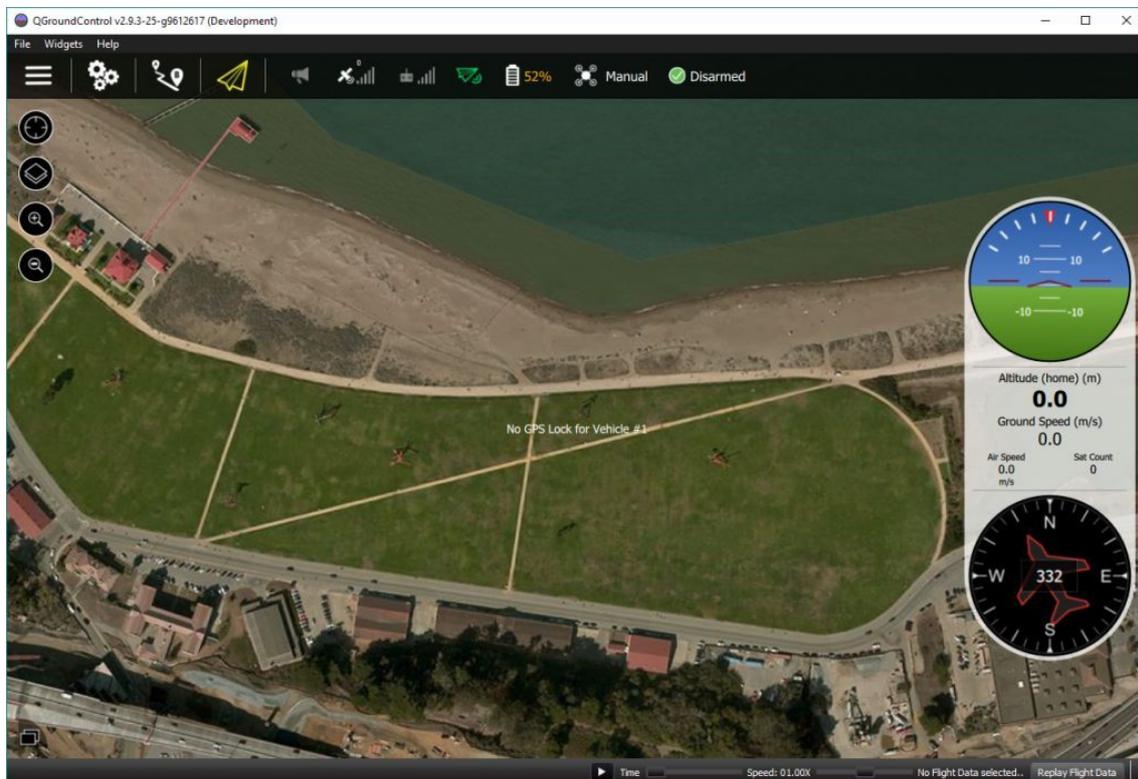


Figure 5.1: This graphic shows an example of QGroundControl. On the right side and on the top bar you can see some real-time information about the drone.

5.3 Simulation

PX4 also offers a good support for simulating a drone⁸. Simulation is a safe and easy way to test your new code instead of immediately fly outdoor in the real world. Two different types of simulation are supported by PX4:

- Software In The Loop (SITL) simulation
- Hardware In The Loop (HITL) Simulation

The SITL simulator enables the possibility to simulate your drone without any hardware, so you can develop a drone without having any flight controller. In contrast to SITL, the HITL simulator simulates everything on the real flight controller. Simulating a drone in a simulation is not simple, but PX4 provides a really good documentation which includes different examples of how to simulate a drone.

5.3.1 Supported Simulators

PX4 supports four different simulators. Nevertheless, using the Gazebo⁹ simulator is highly recommended for the most purposes. Gazebo makes it possible to simply simulate one or more robots, in our case multiple drones, in an simulation environment. Furthermore, PX4 provides much documentation of how to run PX4 in combination with Gazebo, which makes it really beginner-friendly.

JMAVSim¹⁰ is another simulator supported by PX4. This simulation is not as powerful as Gazebo but still good enough for simple tasks like take off or landing. Furthermore, jMAVSim can be used for either SITL or HITL simulation.

PX4 also supports the AirSim¹¹ simulator which is an open-source, cross platform simulator developed by Microsoft AI & Research¹². This simulator is built on the Unreal Engine¹³ and Unity¹⁴ this is one reason why this simulator is very resource intensive and requires a powerful PC. Like the other simulators, AirSim also supports the SITL and HITL simulation.

The last supported simulator is XPlane¹⁵. In contrast to the others, this simulator does not support multirotors, just planes. Moreover, only HITL simulation with XPlane is supported by PX4. Nevertheless, it is a powerful simulator which offers very realistic flights.

⁸Dronecode Project, Inc., *Simulation*.

⁹Open Source Robotics Foundation, *Gazebo*.

¹⁰PX4, *Simple multirotor simulator with MAVLink protocol support*.

¹¹Microsoft, *Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research*.

¹²Microsoft, *Microsoft Research AI (MSR AI)*.

¹³*Unreal Engine*.

¹⁴Technologies, *Unity*.

¹⁵*X-Plane 11 Flight Simulator | More Powerful. Made Usable*.

Chapter 6

Sensors

Author: Konstantin Lampalzer

Sensor equipment is one of the most important part in a robotics system. With sensors the robot is able to perceive his environment and act according to changes. Mapping the surroundings and calculating paths around obstacles is an essential part in this thesis. Therefore, we decided to describe some sensor systems we used and present the challenges we had during the development phase.

6.1 Altimeter

An altitude meter (short: altimeter) is a sensor used to determine the altitude of an object relative to a fixed height. The sensor does this by measuring the atmospheric pressure at the current height which can then be used to calculate the altitude. One way of measuring the pressure at the current height is a piezoresistive pressure sensor. It uses piezoresistive material in combination with a diaphragm. This diaphragm changes shape depending to the external pressure. This change can then be measured with the piezoresistive sensors.

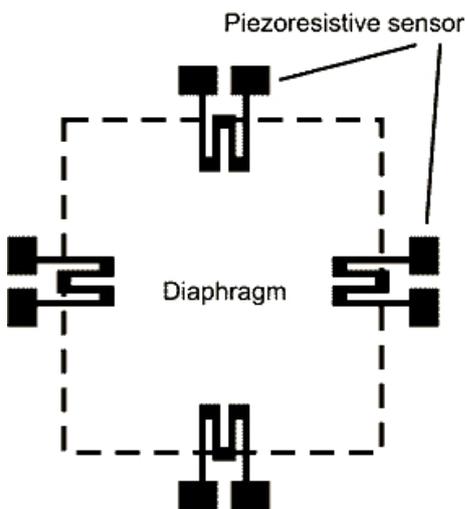


Figure 6.1 shows the structure of an altimeter looking down on the top of the altimeter. In the middle of the sensor is a sheet of a flexible material anchored at its periphery. This material moves up or down depending on the pressure. At the sides of this material are piezoresistive sensors measuring the change of resistance in the piezoresistor.¹

Figure 6.1

¹<https://www.radiolocman.com/review/article.html?di=148185>

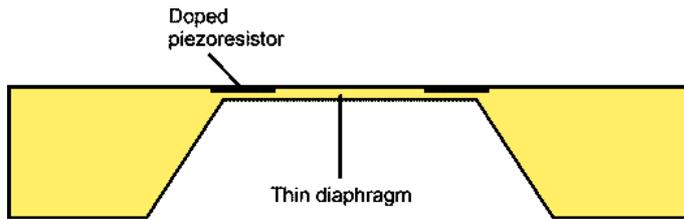


Figure 6.2

Figure 6.2 shows the cross-section of an altimeter. The pressure above and below the diaphragm are the same. Therefore, the material does not change its shape.²

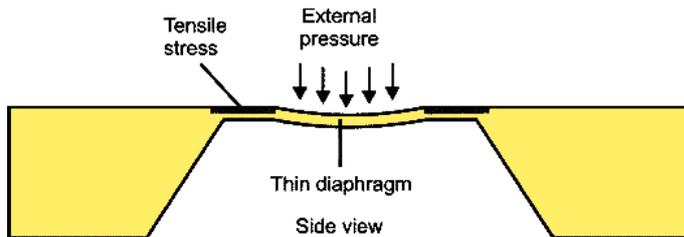


Figure 6.3

Figure 6.1 displays the cross-section of an altimeter with pressure being applied from above. Therefore, the diaphragm moves down slightly, applying tensile stress to the piezoresistors.³

Using pressure sensors in order to calculate the current height can be effective in outdoor use, but several problems occur when flying over uneven terrain. Altimeters get calibrated reference pressure set at ground level. Therefore, it only calculates the height difference to this reference. This means, that it does not provide the height above ground. This can lead to drones crashing into the ground if it is flying in uneven terrain. Another problem can occur if the altimeter is used indoors. As most modern buildings have some form of ventilation system, the pressure inside a room can fluctuate often. This can also occur, if someone opens a window or a door. Therefore, the use of altimeters relying on pressure is not recommended.

6.2 Accelerometer

Accelerometers are sensors, that measure acceleration, meaning the change of velocity of an object in reference to a free fall. Therefore, an accelerometer at rest will measure the acceleration g upwards, where g denotes the constant of gravitation. These sensors are usually developed as micro-electromechanical systems (MEMS), meaning they are really small in size and have moving parts in them. There are different types of accelerometers but generally speaking they are made using capacitive plates. Some of those plates are fixed, while others are moving, attached to springs. If forces are applied to the sensor, these plates start moving, changing the distance between the plates. These changes in distance result in a change of capacitance between plates, which can be measured to determine the acceleration applied to the sensor.

²<https://www.radiolocman.com/review/article.html?di=148185>

³<https://www.radiolocman.com/review/article.html?di=148185>

⁴<http://www.instrumentationtoday.com/mems-accelerometer/2011/08/>

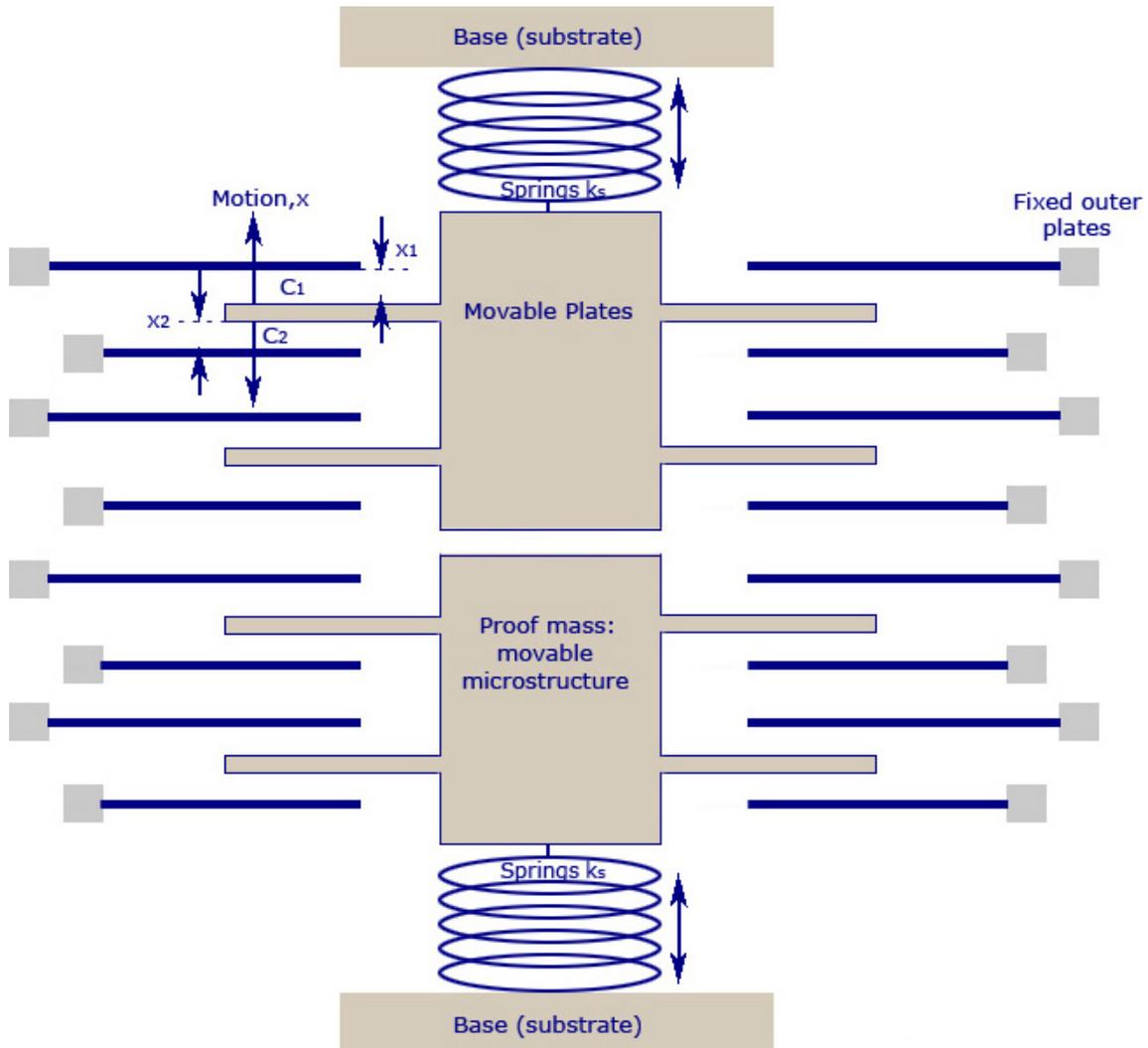


Figure 6.4: Structure of an accelerometer. If force is applied to the sensor, the distances $C1$ and $C2$ change. These changes in distance result in a change of capacitance, between the movable mass and the fixed plates. This change in capacitance can then be used to determine the acceleration in one axis. ⁴

6.3 Satellite Navigation Systems

Satellite navigation systems give a user the ability to calculate their current location (longitude, latitude, and altitude/elevation). There are multiple components to such systems: The sender (a satellite) and the receiver. The satellites continuously transmit navigation signals, which then get received by a receiver. These signals contain the necessary information to compute the travelling time from satellite to receiver and the position of the satellite. If the user receives this information from four or more different satellites, he can use it to calculate his own location. There are currently multiple systems run by different countries, which can be publicly used by anyone.

Each of these systems use different signals and wavelengths but they are all based on the same principle.

1. The GPS receiver detects the signal from several satellites. Each transmission is time-tagged and contains the satellites position. This time-tag is called time of transmis-

Name	Owner
BeiDou	China
Galileo	European Union
GLONASS	Russia
GPS	United States
NAVIC	India
QZSS	Japan

Table 6.1: Publicly available satellite navigation systems

sion. Additionally the receiver saves the time of arrival of every transmission.

2. The time of arrival is then compared to the time of transmission to calculate the time difference.
3. It is then multiplied by the speed of light to obtain the range between the sender and the receiver.
4. Each range puts the user on a sphere around the satellite.
5. Intersecting several spheres can then be used to finally calculate the user position.

In order to receive a signal from a satellite the receiver needs a line of sight to the sender. This results in several problems, that can reduce the accuracy of satellite navigation systems. One of the biggest problems is the use of these systems in urban areas. If the user is near a high building or even under a tree, the line of sight could be blocked. Furthermore, there can be problems caused by reflections created from the signals bouncing off of different buildings.

6.4 Gyroscope

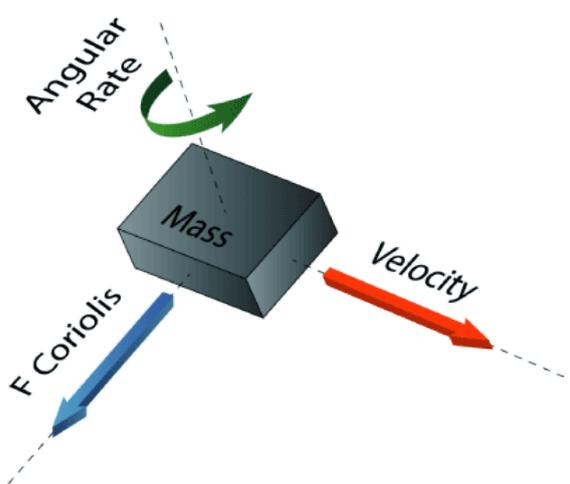


Figure 6.5

A gyroscope sensor measures the angular rate of an object. To do this it uses the Coriolis Effect. First, an alternating current is used to oscillate a mass. If an external angular rate gets applied to this mass, a force will occur, displacing the mass. This force, marked in blue in figure 6.5, is perpendicular to the oscillating direction, marked in red, and the angular rate, marked in green. Similar to the accelerometer this movement causes a change in capacitance between the movable mass and fixed plates around the moving mass. This can then be measured and used to calculate the corresponding angular rate.

6.5 Filtering

In order to produce useful measurement values from a sensor, we have to reduce out the noise generated by those sensors. For example, this noise can arise, when two cables

are interfering with each other. Noise can also occur in the measurement. If we put our magnetometer right next to the power cable on the drone, this could also generate incorrect signals. To filter this out, we decided to compare some algorithms.

6.5.1 Low-pass filter

The simplest form of filters is a Low-pass filter. It is designed to let all frequencies below a certain threshold pass, while dampening everything exceeding the maximum allowed range. It is really easy to implement this filter in python, as a vast amount of libraries provide some form of low-pass filter. One example can be seen in 1

Listing 1: Example of an low-pass filter implemented in python. The filter is already provided by the scipy library⁵

```

1 def butersworth(y):
2     # First, design the Buterworth filter
3     n = 2     # Filter order
4     Wn = 0.4 # Cutoff frequency
5     B, A = signal.butter(n, Wn, output='ba')
6
7     # Second, apply the filter
8     return signal.filtfilt(B,A, y)

```

6.5.2 Moving Average Filter⁶

The moving average filter is one of the most common filters used in signal processing. Mainly, because it can be implemented and understood easily. As the name already implies, the moving average filter takes a number of points from the input and then averages them. This number of points is defined by the window size. For example, if we choose a window size of 5, we take two values before and two after our current measurement and calculate the average of them. This means, we choose the points symmetrically around our current value. Another option would be to only take one side and average them. For our symmetrically moving window, we would have the following formula:

$$y_i = \frac{1}{M} * \sum_{j=-\frac{M-1}{2}}^{\frac{M-1}{2}} x_{i+j}$$

Figure 6.6: Formula for the moving average algorithm

Variable	Description
y	filter output
x	sensor output
M	window size

Figure 6.7: Variables used in the calculation of the moving average filter.

6.5.3 Moving Median Filter

Another form of filtering is the moving median algorithm. Like the moving average described before, it has a window that is moved through the sensor data, but instead of

⁵SciPy developers, *SciPy.org*.

⁶Smith, *Digital Signal Processing*.

calculating the average, we take the median. This means, we take all elements in the window, sort them and then take the element in the middle. If the amount of elements is even, we would just average the two elements in the middle. In contrary to the moving average, the moving median is less sensitive to short spikes in the data. Although this might seem like an advantage, if the spike is longer than the window size, the output will be effected more than in the moving average. Another disadvantage of the median filter is the need to sort all the data. This makes it computationally more intensive than the average, especially with large window sizes.

6.5.4 Exponential Filter⁷

Another really simple filter is the exponential filter. It is intended to work like a low-pass filter removing high frequency noise. Comparing it to the last two filters, it has the similar effect, that it introduces a lag into the system. Another advantage of the system is it's low memory requirement. It only needs to store the last predicted value.

$$y_i = a * y_{i-1} + (1 - a) * x_i$$

Figure 6.8: Formula for the exponential filter

Variable	Description
y	filter output
x	sensor output
a	smoothing constant

Figure 6.9: Variables used in the calculation of the exponential filter.

⁷Standards, (U.S.), and SEMATECH, *NIST-International Sematech E-handbook: NIST Handbook 151. Dataplot : NIST Handbook 148.*

6.5.5 Spike Filter

The idea of the spike filter is to remove values exceeding a predefined threshold, while keeping everything in the range as it is. The values outside the range only get accepted, once they have occurred often enough. If someone would use this in a series of filters, it is recommended to place it at the first position, as other filters would dampen the spike. The filters following in the path would not get affected, as the range is normally not exceeded under normal operation. This filter could be implemented as follows:

```

1 for i in range(1, N):
2     if abs(xi - yi-1) > M and c < n:
3         c = c + 1
4         yi = np.average(np.array(y[i-c:i]))
5     else:
6         c = 0
7         yi = xi

```

Variable	Description
y	filter output
x	sensor output
M	maximum change
n	maximum number of extreme changes

Listing 2: Example code on how to implement a spike filter.

Figure 6.10: Variables used in the calculation of the spike filter.

6.5.6 Summarizing

To compare all the filters to each other, we used them all on the data set represented in figure 6.11. By looking at the different curves, we can see that the moving mean and moving average look pretty similar to each other. They could both successfully eliminate most of the noise with only leaving some spikes in the data. Looking at the exponential filtering, we can see that it pretty much retained the original, but damped it a bit. Lastly, the spike filter did its job pretty well, leaving most of the original data, while removing some spikes.

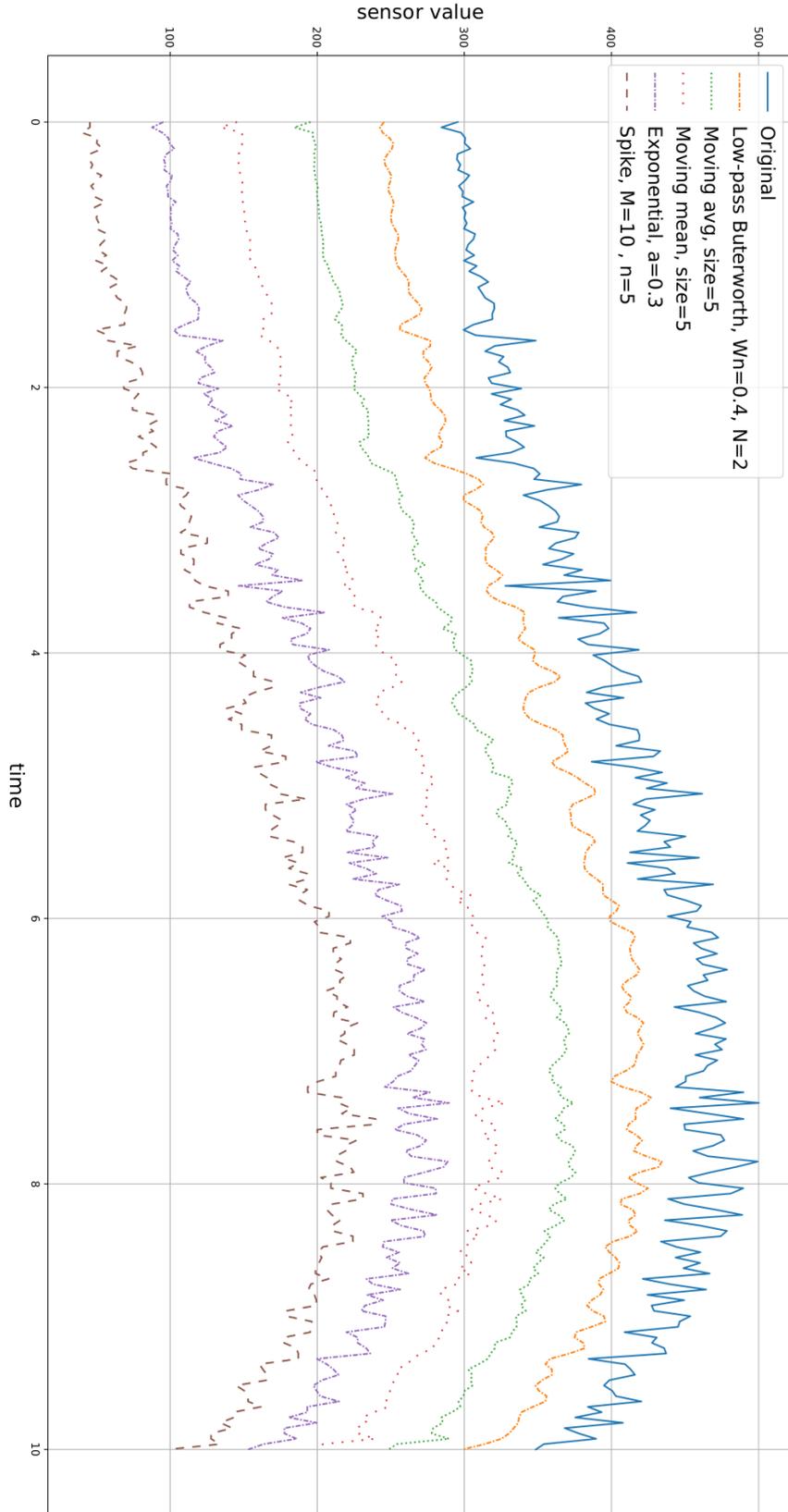


Figure 6.11: All filter algorithms described in this section applied to the same data set. Each curve is offset + 50 units to the last algorithm on the y-axis.

Chapter 7

Pathfinding

Author: Matthias Grill

Pathfinding with unmanned aerial vehicles (UAVs) is a major part of this diploma thesis. As a short reminder, our drone has to fly a predefined route. During the flight, the drone scans its environment using sensors, generating a map. Additionally, it detects obstacles on its way. Afterwards, if the UAV found an obstacle, it has to generate a new path avoiding the obstruction in its route, preventing a crash. We use Pathfinding algorithms, in order to find the shortest route around obstacles.

7.1 Concept

Pathfinding is an autonomous process for finding a path from one given start point to another end point on a map. This map contains multiple obstacles which have to be avoided. Moreover, as different routes exist, from start to finish, the goal is to find the shortest one.

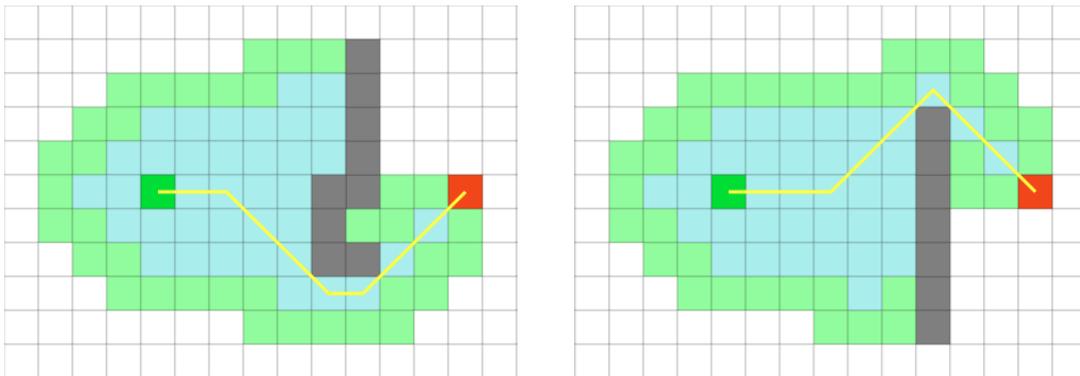


Figure 7.1: These figures show two examples of pathfinding. The green dots are representing the start points, the red ones, the end points and the gray dots, the walls. As you can see, these pathfinding algorithms found the shortest way from the start to the end, successfully avoiding the obstacles. These figures were created using a web tool¹.

¹<https://qiao.github.io/PathFinding.js/visual>

7.2 Applications of Pathfinding

7.2.1 Pathfinding in Computer Games

Nowadays, most computer games are making use of Artificial Intelligence (AI). Therefore, movement of AIs need to be as realistic as possible to provide the best gaming experience. These AIs are often called NPCs or non-player characters and need pathfinding algorithms to efficiently navigate around the world. The next figures represent the most common ways to simplify the world for navigation.²

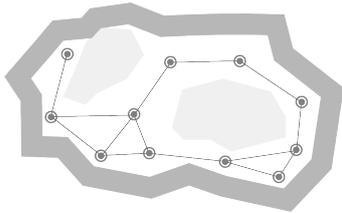


Figure 7.2: Waypoints³

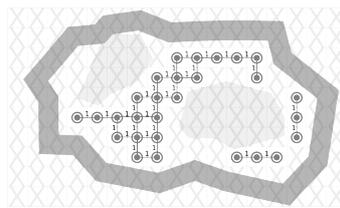


Figure 7.3: Grid⁴

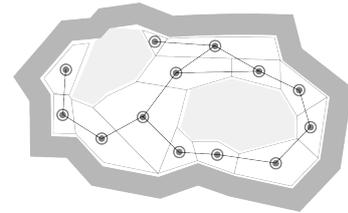


Figure 7.4: Navigation Mesh⁵

Moreover, certain paths in the game world are not as fast as other ones. For example, driving on a motorway is much faster than on a gravel path. Therefore, the pathfinding algorithm needs to have some sort of weighting implemented, to prioritize some routes before others. The most commonly used algorithm in computer games is A*, which is explained in detail later in this chapter⁶.

7.2.2 Pathfinding in Exploration

Another important application of pathfinding is the exploration of new territory. Often robots with specialized algorithms are used, instead of humans. For example, NASA⁷ explores Mars with the help of rovers. There are many reasons why NASA sends robots for exploring, rather than humans. First of all, there is the minimized risk endangering a person, as most planets provide harsh climate conditions. Furthermore, sending robots to planets is much cheaper, as they can withstand higher forces of acceleration and can be transported more easily than humans. Lastly, if a robot mission is completed, NASA does not have to organize an expensive return from Mars to Earth⁸.

Aside from other planets, there are difficult places on earth aswell. For example, drones or ground-based-robots are used in mapping a whole cave for finding minerals or underwater robots which explore undersea caves^{9,10}. All these projects are making use of pathfinding algorithms, for instance avoiding obstacles and planning a new path around them or just to find the shortest way from point A to B.

²Graham, Ross; McCabe, Hugh; and Sheridan, Stephen, *Pathfinding in Computer Games*.

³<http://jceipek.com/Olin-Coding-Tutorials/pathing.html>

⁴<http://jceipek.com/Olin-Coding-Tutorials/pathing.html>

⁵<http://jceipek.com/Olin-Coding-Tutorials/pathing.html>

⁶Graham, Ross; McCabe, Hugh; and Sheridan, Stephen, *Pathfinding in Computer Games*.

⁷Kristen Erickson, *The Mars Rovers :: NASA Space Place*.

⁸Kristen Erickson, *Why do we send robots to space? :: NASA Space Place*.

⁹Sreeja Banerjee, *A COMPARATIVE STUDY OF UNDERWATER ROBOT PATH PLANNING ALGORITHMS FOR ADAPTIVE SAMPLING IN A NETWORK OF SENSORS*.

¹⁰Himangshu Kalita; Steven Morad; Aaditya Ravindran; Jekan Thangavelautham, *Path Planning and Navigation Inside Off-World Lava Tubes and Caves*.

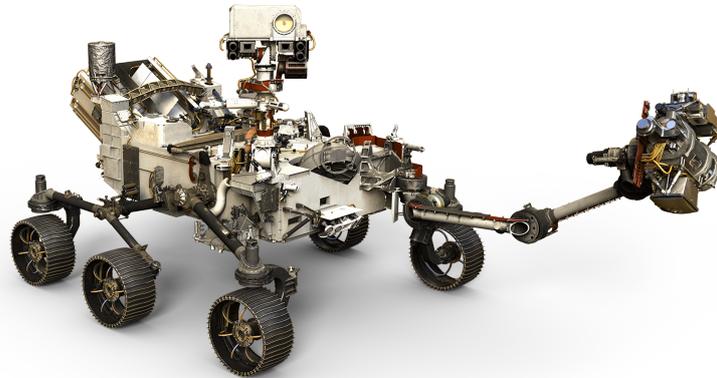


Figure 7.5: This figure shows the Mars2020 rover from NASA. The robot will be in use in 2021 on the Mars. Goal of this mission is finding more information about the past life on the planet.¹¹

7.2.3 Pathfinding in industrial applications

Robots carrying goods in warehouses are already reality. For instance, Amazon Robotics¹² makes use of this. Automated Guided Vehicles (AGV) search for the ordered products in the warehouse and bring them to the packing station autonomously, where humans put the goods in a packet. Amazon Robotics uses pathfinding algorithms¹³ for planning the most efficient paths and without crashing in another AGV. Moreover, another important application in industry are robotic arms. These are used in different sectors for example in assembly line production where they have to accomplish various tasks. Depending on the task the arm can be built versatile. For example a gripper with two, three or five fingers. MoveIt!¹⁴ is a motion planning framework which runs on top of the Robot Operating System (ROS). It allows us to plan, manipulate and visualize the motions of a robot arm in a 3D environment. Here pathfinding comes into use. If you want to move the arm from shelf A to B, MoveIt! calculates the shortest path to it and also takes account to obstacles. The framework has many advantages, one of it is the wide range of different provided functions. Additionally, because it is an open source framework, MoveIt! has a large community and many projects¹⁵. This improves quality and security because bugs can be found more easily. However, there are also disadvantages and one of the biggest is that it is not beginner-friendly because of the large number of features and the complexity. This does not mean that there is no latest available documentation. On the contrary, there is much of it on their website, but you will need much time to install and understand MoveIt!.

¹¹<https://mars.nasa.gov/mars2020/mission/rover>

¹²Amazon Robotics, *Amazon Robotics*.

¹³Jun-tao Li; Hong-jian Liu, *Design Optimization of Amazon Robotics*.

¹⁴*MoveIt! Motion Planning Framework*.

¹⁵*Related Projects | MoveIt!*

7.3 Pathfinding Algorithms

Depending on the task various pathfinding algorithms are available today. The most used and known ones are the Breadth First Search, the Dijkstra's Algorithm and the A* (A star) search algorithm. The most simply search algorithm is the Breadth First Search. Adding some features to it will turn it into a complex A* search algorithm. All these algorithms are graph search algorithms. This means that you need a graph as input.

7.3.1 Breadth First Search

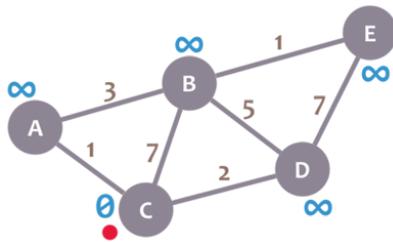
A Breadth First Search algorithm explores the graph in every direction, beginning from the start point. The main purpose of this algorithm is to find the shortest path from start to end, but this algorithm can be used for generating a map as well. Its main feature is the frontier, which is an expanding ring. It is implemented with a simple queue. At the beginning the frontier starts with the starting point. Afterwards, it picks a location from the frontier queue and removes it. Now you have to look at the neighbors which are not already visited. Add the new locations to the frontier and mark them as visited. Repeat this process till the frontier queue is empty. This process is also called flood fill.

Listing 3: Example code of a Breadth First Search implemented in Python.

```
1 frontier = Queue()
2 frontier.put(start)
3 visited = {}
4 visited[start] = True
5
6 while not frontier.empty():
7     current = frontier.get()
8     for next in graph.neighbours(current):
9         if next not in visited:
10            frontier.put(next)
11            visited[next] = True
```

7.3.2 Dijkstra's Algorithm

As the name implies, the algorithm was published by Edsger W. Dijkstra (/daɪkstrə/) in 1959. Goal of the Dijkstra's Algorithm is to find the shortest path between one node and every other node in a graph. Difference between Breadth First Search and Dijkstra's is that the Dijkstra's algorithm takes account to movement costs. For example connection A is twice as fast as B.

Figure 7.6: ¹⁶

First of all you have to initialize the relevant data structure. We need a start node which is represented in figure 7.6 as a red dot, this is also the current node at the beginning. Moreover, we set the distance from node C to C to 0 because it is our starting point. During the runtime we will receive the minimum distances between Node C and every other Node, so we can find the shortest way. For the beginning we will set the distance from the rest of the nodes to infinity because they are currently unknown.

Afterwards, we have to calculate the minimum distance from every neighbour. For example the distance from C to B is calculated by adding 0, which is the minimum distance of C, with 7, which the weight of the edge connecting our current node with B, to obtain a minimum distance from C to B of 7. This process must be carried out for each additional neighbour. After calculating every distance the current node has to be marked as visited. Afterwards, pick another current node and repeat the whole process again and again until there are no unvisited nodes left.

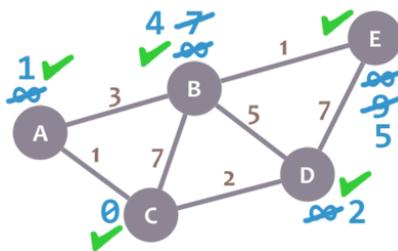
Figure 7.7: ¹⁷

Figure 7.7 shows the final result of the example. As you can see every minimum distance is calculated and every node is marked as visited, represented with a green check mark. Now you are aware of the shortest ways and you can easily find the most efficient route from one point to another.

7.3.3 A* search algorithm

A* search algorithm (A*) is build on top of the Breadth First Search and Dijkstra algorithm. Moreover, A* is only used for finding the shortest path to one specific point and does not explore all possible ones. In contrast to Dijkstra's Algorithm, A* accounts for the actual distance from start and the estimated distance to goal. Consequently, A* does not waste time on exploring directions which are not promising. This algorithm is wide spread in the gaming industry to ensure a better gaming experience.

Listing 4: Example code of a A* search algorithm implemented in Python.

```

1 frontier = PriorityQueue()
2 frontier.put(start, 0)
3 came_from = {}
4 cost_so_far = {}
5 came_from[start] = None

```

¹⁶<https://www.codingame.com/playgrounds/1608/shortest-paths-with-dijkstras-algorithm/dijkstras-algorithm>

¹⁷<https://www.codingame.com/playgrounds/1608/shortest-paths-with-dijkstras-algorithm/dijkstras-algorithm>

```

6 cost_so_far[start] = 0
7
8 while not frontier.empty():
9     current = frontier.get()
10
11     if current == goal:
12         break
13
14     for next in graph.neighbors(current):
15         new_cost = cost_so_far[current] + graph.cost(current, next)
16         if next not in cost_so_far or new_cost < cost_so_far[next]:
17             cost_so_far[next] = new_cost
18             priority = new_cost + heuristic(goal, next)
19             frontier.put(next, priority)
20             came_from[next] = current

```

7.3.4 Pathfinding in 3D environments

As our drone is flying in a 3-Dimensional environment, we need to make sure all our algorithms are compatible with 3D. Because it is possible to move in an extra dimension much more data can occur. Therefore, more information has to be processed by the algorithm and more computing power is needed. Instead of using a 2D graph for abstraction, we now need to use 3D graphs. Consequently x^2 nodes are available, which does not make a big difference for the algorithms, but it leads to higher memory use and a need for more computing power.

7.3.5 Efficiency of pathfinding algorithms

One of the most important parts of an algorithm its efficiency, which can be expressed using the Big-O notation. This notation defines the worst-case scenario and describes the necessary execution time of an algorithm¹⁸.

Algorithm	Worst-case performance
Breadth First Search	$O(V + E) = O(b^d)$
Dijkstra's Algorithm	$O(E + V \log V)$
A* search algorithm	$O(E) = O(b^d)$

Table 7.1: Big-O notations of pathfinding algorithms. Where E is the number of edges and V is the number of nodes.

7.4 Conclusion

Overall, we underestimated the effort for this part of the diploma thesis. Programming and implementing a 3D pathfinding algorithm for a drone is not as easy as we anticipated. Moreover, the GPS accuracy was not sufficient for our goals. With our current budget and time-limits we were not able to reliably track the position of a moving drone on the calculated map. In order to improve this thesis on the subject of path planning, more

¹⁸Rob, *A beginner's guide to Big O notation* - Rob Bell.

time and a substantial amount of effort would be needed. Instead, we decided to focus on researching the foundation elements of aerial navigation, in order to provide our successors with the knowledge they need to implement more complex matters, such as path planning.

Chapter 8

Graphical User Interface

Author: Matthias Grill

8.1 Introduction

Another requirement of this diploma thesis was the creation of a user interface (UI). The front end was required to be a lightweight and intuitive way to control the drone in critical situations. This ensured, that every soldier of the Austrian Armed Forces could control the drone without the need for a long training. Reliability was another major requirement set by the client. This was needed as errors in the control application could lead to loss of equipment or in the worst case endangering a soldier.

8.2 Choice of platform

Because no platform for the UI was specified we had to choose one. Therefore, we decided to develop a web application, because it's system independent and can be easily modified to run on a mobile device. Furthermore, we gathered much experience in web development during the past years so it was not as difficult as a desktop application would have been. In the end, we chose A single-page application (SPA).

It is a web application that as the name suggests only contains one HTML page. In most cases this file is named "index.html". Depending on the users actions that page will be dynamically updated to fulfill the users request. There are several advantages why SPAs are used. They enhance the user experience because everything necessary to display the user interface is pre-fetched. This leads to swift response times and imitates the user experience of a desktop application.¹

8.3 Vue.js

Vue is a javascript framework which is used for building single-page applications². It was used for this diploma thesis because it is simple to use and beginner friendly. In comparison to other frameworks for example Angular or Ember it is lightweight. Additionally, Vue.js provides a good documentation and user support. It is also an open source software licensed

¹Angular University, *Angular SPA*.

²Vue, *Introduction — Vue.js*.

under the MIT license (MIT)³ which allows to use it for private and commercial projects⁴.

8.4 Rosbridge

For the connection between the front end and the AARD back end we made use of Rosbridge⁵. It provides a JSON API for interacting with the Robot Operating System (ROS). Therefore, we are able to communicate with the ROS master and we can easily subscribe and publish to all topics. Subscribing to a topic is necessary for the GUI to get it' information from the network. The UI contains a whole section which is used just for monitoring the drone. Each value is a subscription from a mavros topic for example the altitude of the drone or the current flight mode. Furthermore, Rosbridge allows us to rapidly add new UI elements, as all the information necessary is already present or can be easily added with a subscription to a topic.

Listing 5: Example of subscribing a topic with Rosbridge

```

1  var ros = new ROSLIB.Ros({
2      url: "ws://localhost:9090"
3  });
4
5  var altitudeListener = new ROSLIB.Topic({
6      ros: ros,
7      name: "/mavros/global_position/rel_alt",
8      messageType: "std_msgs/Float64"
9  });
10
11 altitudeListener.subscribe(function(message) {
12     self.data.altitude = message.data.toFixed(2);
13 });

```

Furthermore, the user is able to plan and adjust the mission in the UI. For example, he can select the coordinates of the drone's destination, set the drone's height or control its maximum speed. All this information needs to be sent to the back end system. This has also been realized by using the rosbridge to communicate from the front end to the mavros node. After the user chooses his preferred settings, he can submit his information and trigger a drone start by clicking the "start"-button which will lead to all necessary information being published to a suitable topic (in our case to the "/drone/start" topic).

Listing 6: Publishing all chosen settings to the "/drone/start" topic with Rosbridge

```

1  startTopic: new ROSLIB.Topic({
2      ros: new ROSLIB.Ros({
3          url: "ws://localhost:9090"
4      }),
5      name: "/drone/start",

```

³OpenSource.org, *The MIT License | Open Source Initiative.*

⁴Vue, *The official documentation site for Vue.js. Contribute to vuejs/vuejs.org development by creating an account on GitHub.*

⁵rosbridge_suite - ROS Wiki.

```

6     messageType: "std_msgs/String"
7 });
8
9 startEvent() {
10     var message = {
11         goalAltitude: this.state.goalAltitude,
12         goalLatitude: this.state.lat,
13         goalLongitude: this.state.lng,
14         scanTime: this.state.scanTime,
15         droneVelocity: this.state.droneVelocity
16     };
17
18     var stringMessage = new ROSLIB.Message({
19         data: JSON.stringify(message)
20     });
21
22     this.startTopic.publish(stringMessage);
23 };

```

Code listing 6 shows the necessary steps to publish a message with rosbridge to a specified topic. After the "startEvent" method is called a new message with the chosen settings will be created. Before the new ROS message can be prepared it is necessary to "stringify" the JSON object. "Stringify" is a method which converts the JSON object to a JSON string so it is possible to exchange this message. This process is called serialization. Subsequently, the previously created ROS message is ready to be published to the start-topic. The backend system will now receive and process this information.

8.5 AARD GUI

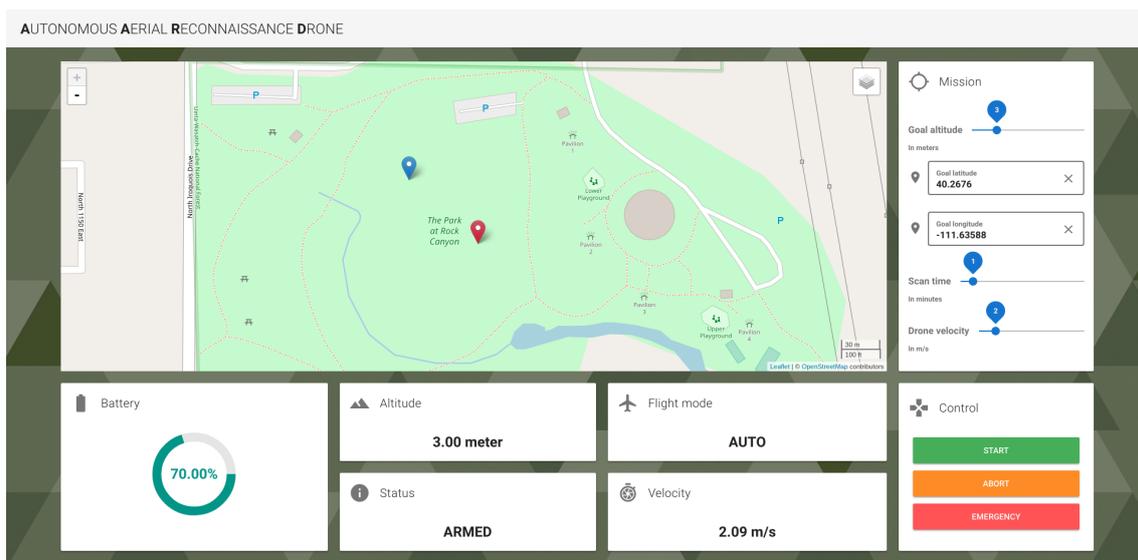


Figure 8.1: Graphical user interface of the AARD application

Figure 8.1 shows the web application which was developed for this diploma thesis. The

user interface (UI) is divided into four different sections to keep it intuitive and simple. Each one has its own purpose:

1. Map: Shows a map which contains markers for the current location of the drone and the selected destination
2. Mission: Settings for the mission which can be configured by the user
3. Monitoring: Shows the current status of the drone consisting of:
 - Battery
 - Altitude
 - Armed or Disarmed
 - Flight mode
 - Velocity
4. Control: Control panel to start or stop the AARD application

An advantage of this simple design is that it is easy to customize for the current needs.

8.6 Docker

We decided to run our AARD frontend software in two Docker⁶ containers. One container contains a web-server which serves the AARD-GUI and the other one for running rosbridge. One major reason why we use docker is the portability across machines. We can assume that our software is running on every other PC which runs docker.

⁶Docker Inc., *Docker*.

Chapter 9

AARD Application

Author: Matthias Grill

9.1 Introduction

Goal of this diploma thesis is to develop the AARD Application for the Austrian Armed Forces. To realize the software we decided to create automated missions, which are already provided by ArduPilot¹. There are a number of benefits associated with using the provided missions. It is open source and a lot of functions are made available, which were already tested. In addition, Ardupilot provides a lot of documentation on their website and offers good support for their users.

9.2 Mission Planning

There are several ways how missions can be created and started. In most cases the Mission Planner 9.1, a Windows-based Ground Control Station (GCS), is used. As Mission Planner is a graphical user interface, developed for humans, we could not use the software to automate our drone. Therefore, we decided to go to a deeper abstraction layer and used the MAVLink² protocol. It is used by the different GCSs to communicate between the PC and the drone.

¹ArduPilot Dev Team, *Mission Commands — Mission Planner documentation*.

²Dronecode Project, Inc., *Introduction · MAVLink Developer Guide*.

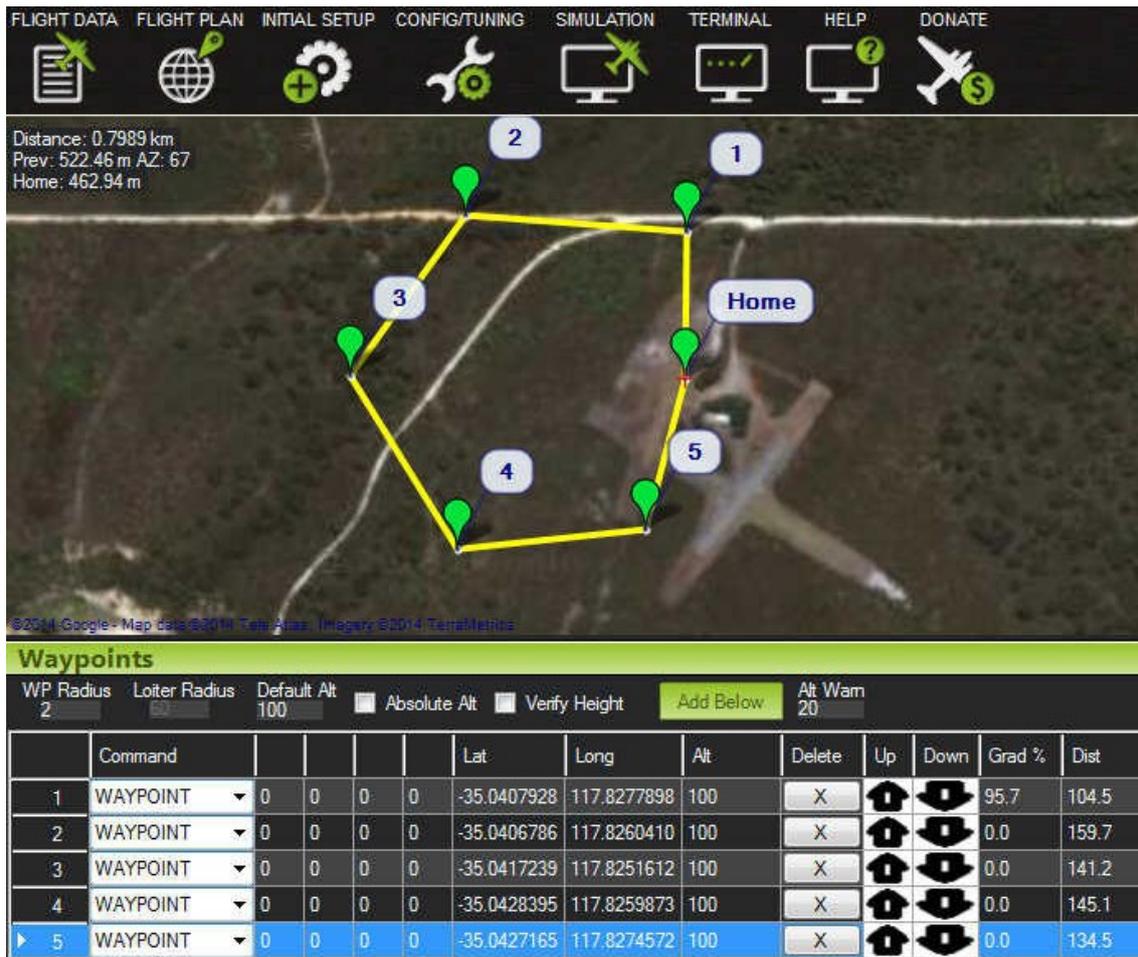


Figure 9.1: The Ground Control Station Mission Planner³. In the top window the user clicks on some points on the map to create waypoints for the drone. In the bottom all points are listed with their corresponding GPS coordinates and altitude.

9.2.1 Waypoints

For us, waypoints are the most important feature of a mission. There is no limit on the amount of waypoints that can be sent to the drone. For example, they can be used to set the home position, initialize a takeoff or land at a specific point. That are just a few examples, but there are much more which can be found in the MAVLink documentation⁴. However, in our case the most used feature was to navigate to specified points.

Listing 7: Example code, creating a new waypoint and sending it to the flight controller.

```

1 self.waypoints.append(Waypoint(frame=3,
2     command=16,           # Command type (16 is move to)
3     autocontinue=True,
4     is_current=True,
5     param1=self.scanTime, # Hold time in decimal seconds
6     param2=0.2,          # Acceptance radius in meters

```

⁴common.xml · MAVLink Developer Guide.

```

7         x_lat=self.goalLatitude,    # Waypoint latitude
8         y_long=self.goalLongitude,  # Waypoint longitude
9         z_alt=self.goalAltitude))   # Waypoint altitude
10
11 self.waypointPushService(start_index=0, waypoints=self.waypoints)

```

As you can see in code listing 7 there are many parameters which can be customized for your own requirements. All the different command types and their associated parameters can be found in the MAVLink documentation⁵.

9.2.2 Commands

The MAVLink protocol provides a large number of waypoint command types. Three different types of commands are defined:

- Navigation commands
- Condition commands
- DO commands

Navigation commands are used to control the drone. For example: takeoff, move to or return to launch (RTL). These commands have the highest priority at runtime, therefore every DO or condition command will be skipped till the navigation command is executed. Conditional commands are used for waiting till a specific condition occurs. For example, to reach a certain height or to fly a distance of five meters before the next command will be performed. Lastly, the DO commands are used to execute one command immediately. For instance to set the drone speed, drone mode or the home position⁶. A long list of all commands and how to use them is available in the MAVLink documentation⁷.

9.3 Flight modes

Arducopter provides 20 different flight modes and ten of them are normally used. Every mode accomplishes another purpose. Switching from one mode to another can be controlled by using a radio, a ground control station or the MAVLink protocol. In the AARD Application we are using a ROS service to switch through the different modes. In the background, this service uses the MAVLink protocol.

Listing 8: Example on how to create the setMode service in ROS.

```

1 self.modeService = rospy.ServiceProxy("mavros/set_mode", SetMode)

```

In code listing 8 we create a ROS service that calls mavros/set_mode. This interface is provided to us by the mavros⁸ ROS package described later in this chapter.

We are using following flight modes in our software:

As you can see, some commands require a GPS location to be used, while others don't. This is a requirement set by the ArduCopter firmware, as these modes need GPS to function properly. For example, RTL can not return to the starting location, if it does not know the

⁵ *common.xml · MAVLink Developer Guide.*

⁶ *ArduPilot Dev Team, Mission Planning — Copter documentation.*

⁷ *common.xml · MAVLink Developer Guide.*

⁸ *Vladimir Ermakov, mavros - ROS Wiki.*

Flightmode	Purpose	GPS required
Alt Hold	Holds altitude and self-levels the roll and pitch	
Auto	Executes pre-defined mission	x
Guided	Navigates to single points commanded by GCS	x
RTL	Returns above takeoff location, may also include landing	x

Table 9.1: List of used flight modes⁹

current location of the drone. If a mode switch command is issued to the flight controller, while GPS is not present, the controller will not switch and instead return a message, that switching was unsuccessful.

9.4 AARD Mission

9.4.1 Initialization

It is required to execute an initialization in order to detect any errors, either from the user or the drone, before takeoff. We do this to make sure that the drone or the person supervising doesn't get hurt in a malfunction of the program. Furthermore, it is necessary to make calculations which will then be used later in the flight phase. In the AARD Application following steps are carried out before launch:

1. Check and set arguments from the user input
2. Create services
3. Create subscribers
4. Create publishers
5. Set parameters for the drone's behavior

After these steps were successfully executed we begin with the arming process.

9.4.2 Arming

Another important part of starting the mission is arming the drone. This can be done by sending the correct message to the flight controller via the MAVLink protocol. During the arming phase, the firmware running on the drone performs its pre-arm safety checks¹⁰. This is necessary, to ensure all sensors are calibrated correctly and no anomalies are present in the sensors. Barometer, compass, GPS or battery checks are a few examples of such safety checks. If any of the pre-arm checks fail, the flight controller will prevent the vehicle from arming. Furthermore, the safety button feature can be enabled, that requires the user to press a button on the drone before arming. This makes sure, that the drone doesn't take off, if someone accidentally presses the wrong button.

9.4.3 Takeoff

Takeoff is the process where the drone leaves the ground and begins to fly. There are several methods to launch a drone. For example the hand launch where the drone takes off from your hand or the catapult launch where you catapult your drone to the sky to achieve a greater level of acceleration¹¹. However, we are using the commonly used method to start

¹⁰ArduPilot Dev Team, *Pre-Arm Safety Check — Copter documentation*.

¹¹ArduPilot Dev Team, *Automatic Takeoff — Plane documentation*.

from the ground and fly straight up. Nevertheless, we still have to do small tasks before takeoff. Firstly, we change the flight mode to guided. Afterwards we have to wait for the GPS signal because, as you can see in table 9.1, guided mode requires GPS. Finally, we can arm our drone and start the launch.

Listing 9: Example code, demonstrating how the drone can be armed from a python program.

```

1 self.SetMode(88, "GUIDED")
2 self.Arm(True)
3 self.TakeOff(altitude)

```

9.4.4 Mission start

After we reach the desired altitude, which the user entered previously, we automatically start calculating the mission. The reason why we do not determine the waypoints before launch is because the drone starts on a moving platform, which has a velocity up to 30 kilometers per hour. For example if we would calculate the waypoints on the ground in a duration of 0.1 seconds and the platform has a velocity of 30 kilometers per hour, we would have an inaccuracy of up to 0.8 meters on the home position, which would lead to some consequential errors. If we have calculated our waypoints successfully we switch the flight mode to auto and the mission starts automatically.

9.4.5 Intermediate goal

First goal of the mission is to reach the first waypoint in the list, which is the intermediate goal. Afterwards, the drone has to wait a specific time in seconds, which is set by the user previously in the graphical user interface (GUI). After waiting, the next calculation will be done. The interception position of the car and the drone must be calculated because the UAV has to fly back to the moving platform. After waiting following information is available for us:

- Current position of the drone (Intermediate goal)
- Velocity of the drone
- Current position of the car (Car velocity multiplied by passed time)
- Velocity of the car

With this information and vector algebra we now can calculate the waypoint for the interception point of the car and the drone.

9.4.6 Calculating the interception point

One direction, constant velocity

Let's consider our car being at point C with a local vector of \vec{c} moving with a constant speed vector \vec{v}_C . Furthermore, let's consider our drone being at point D with a local vector of \vec{d} and a constant speed vector \vec{v}_D . As we do not know this speed vector \vec{v}_D , we additionally define $|\vec{v}_D|$ as the current absolute speed of the drone. Our goal is to calculate the interception point I , at which the vehicle and drone meet. This is done in equation 9.1 In order to calculate I , we additionally need to add t , defining the time needed for both vehicles to travel to the interception point.

$$I = \vec{c} + t * \vec{v}_C \text{ and } I = \vec{d} + t * \vec{v}_D \quad (9.1)$$

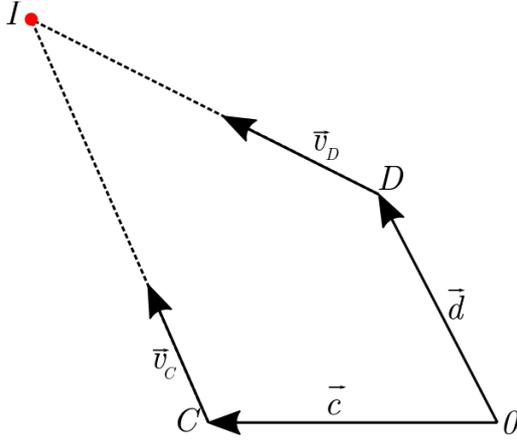


Figure 9.2: Shows an example of how the environment might look like. We used this sketch to come up with our formula to calculate the interception point between the car and the drone.

Variable	Description	Dimension
C	car position	1×2
D	drone position	1×2
\vec{v}_C	car velocity	1×2
$ \vec{v}_C $	absolute car velocity	1×2
\vec{v}_D	drone velocity	1×2
$ \vec{v}_D $	absolute drone velocity	1×2
I	interception point	1×1

Figure 9.3: All variables known to the system.

At first glance, one might think, setting both equations equal, as shown in 9.2, could return a viable result. This is not the case, as \vec{v}_D is unknown. Therefore, we decided on following another approach to the problem.

$$\vec{c} + t * \vec{v}_C = \vec{d} + t * \vec{v}_D \quad (9.2)$$

Our second idea was calculating the time t needed to get from our current point to the interception point I . Since our drone needs the time t to get from it's starting point D to the interception point I , we can divide this distance by our known speed $|\vec{v}_D|$ to get t . Again, we can not use this equation, as we would need to know the absolute speed of the drone. Instead, putting all know variables into one equation we get 9.3. This can be used to calculate the time, but gives us the next problem, of t being at both sides of the equation.

$$t = \frac{|\vec{c} + \vec{v}_C * t - \vec{d}|}{|\vec{v}_D|} \quad (9.3)$$

Before we try to solve this, we can simplify the formula by removing the fraction. This gives us the equation defined in 9.4. Furthermore, we can add the helper variable \vec{o} , representing the offset between \vec{c} and \vec{d} , as described in 9.5. Putting this all together, it results in 9.6. Our next step is removing the length operator from the equation.

$$|\vec{v}_D| * t = |\vec{c} + \vec{v}_C * t - \vec{d}| \quad (9.4)$$

$$\vec{o} := \vec{c} - \vec{d} \quad (9.5)$$

$$|\vec{v}_D| * t = |\vec{o} + \vec{v}_C * t| \quad (9.6)$$

By using the Pythagorean theorem, we know that the length of a vector is the length of a vector is the square root of the sum of the squares of its components. This results in 9.7. In the next step we remove the square root, giving us $\vec{v}_D^2 * t^2 = (o_x + v_{Cx} * t)^2 + (o_y + v_{Cy} * t)^2$. As a last step we multiply out the squares, resulting in 9.8, which is a quadratic equation that can be solved by us.

$$s * t = \sqrt{(\vec{o}_x + \vec{v}_{Cx} * t)^2 + (\vec{o}_y + \vec{v}_{Cy} * t)^2} \quad (9.7)$$

$$t = t^2 * (v_{Cx}^2 + v_{Cy}^2 - |\vec{v}_D|^2) + 2t * (o_x * v_{Cx} + o_y * v_{Cy}) + o_x^2 + o_y^2 \quad (9.8)$$

Although, we could solve this right now, we simplify this even further and replace all the parts with known variables with helper variables defined in 9.9, 9.10 and 9.11.

$$h_1 := v_{Cx}^2 + v_{Cy}^2 - |\vec{v}_D|^2 \quad (9.9)$$

$$h_2 := o_x * v_{Cx} + o_y * v_{Cy} \quad (9.10)$$

$$h_3 := o_x^2 + o_y^2 \quad (9.11)$$

We can now solve this quadratic equation putting it into the quadratic formula $t_{1,2} = \frac{-2 * h_2 \pm \sqrt{2 * h_2^2 - 4 * h_1 * h_3}}{2 * h_1}$, which can be simplified, giving us two, one or none solutions to the problem.

$$t_{1,2} = -\frac{h_2}{h_1} \pm \sqrt{\left(\frac{h_2}{h_1}\right)^2 - \frac{h_3}{h_1}} \quad (9.12)$$

Although we solved the equation, we still got to look at the edge cases. First, if the expression under the square root is negative, the equation has no solution, meaning the car and the drone will never intercept each other. The other edge case is present, if h_1 is zero, giving us the problem, that we can't divide by zero. Lucky for us, this case is the simpler case to solve. Looking back at $t^2 * h_1 + 2t * h_2 + h_3 = 0$ we can see, that if we set h_1 to 0 the equation becomes $2t * h_2 + h_3 = 0$ meaning we can simply solve it, represented in 9.13.

$$t = -\frac{h_3}{2 * h_2}. \quad (9.13)$$

After we now put down all the equations needed, we can easily implement this resulting in the following code:

Listing 10: Code showing how we used our formulas to calculate the interception point between the car, driving at a constant velocity, and the drone.

```

1 def calculateInterceptionPoint(startCar, velCar, startDrone, velDrone):
2     o = startCar - startDrone
3
4     h1 = np.inner(velCar, velCar) - velDrone ** 2
5     h2 = np.inner(o, velCar)
6     h3 = np.inner(o, o)
7
8     if h1 == 0.0:
9         t = -h3 / (2.0 * h2)
10
11    else:
12        discriminant = (h2 / h1) ** 2 - h3 / h1
13        if (discriminant < 0.0):
14            return np.array([0.0, 0.0])
15
16        t1 = (h2 / h1) + sqrt(discriminant)
17        t2 = (h2 / h1) - sqrt(discriminant)
18        tMin = min(t1, t2)

```

```
18         tMax = max(t1, t2)
19         t = tMin if tMin < tMax and tMin > 0.0 else tMax
20
21     return startCar + t * velCar
```

One direction, varying velocity

Sadly, the last example, one direction, constant velocity, is only present in a simulated environment. In reality, the environment changes constantly. Winds could affect the vehicle or friction could slow it down. Furthermore, a car needs time to accelerate or decelerate. Therefore, we decided to try another approach to intercept the vehicle. Of course, one direction is still somewhat unrealistic, but we tried to break our problem down, solving the simple ones first. In order to cope with all the changes, we mounted another flight controller to the moving vehicle. This flight controller is connected to a Raspberry Pi 3 Model B+¹² that connects to the flight controller via mavros¹³. Both, the drone and the controller on the vehicle, are connected to the same WLAN, so they can communicate via ros. We can then use this to get the current velocity and the gps position of the landing platform and the drone. Having all this data, we can use the methods described in the last example, but instead of the start position and velocity, we use the current position and velocity of the car. We then periodically run this function to update the interception point.

Multiple directions, varying velocity

Lastly, the biggest challenge is presented, when a vehicle is moving in any direction with changing velocity. This can be the case, when a car is moving around in a city, changing lanes or turning around a corner. In order to land on such a moving object, algorithms need to be in place, that predict the route. One way to simplify this, would be to define the route the vehicle will take in advance. A tool could then use the speed limits, or just any publicly available route planning API to calculate an interception point.

9.4.7 Land

The landing phase is the last phase of this diploma thesis. The goal is to land on a platform mounted on a vehicle, while the vehicle is moving. This presents several challenges and their possible solutions, which are documented in the next section:

Vehicle Detection

In order to land on the vehicle, we first need to detect the vehicle and then calculate the relative position between the car and the drone. This is as trivial as it sounds, as we have a constantly changing environment. This also includes multiple conditions in which the drone still needs to be able to detect the vehicle. For example, this might include a flight in the dark, during rain or during a snowstorm. As wanted to focus more on the land process itself, we decided to detect the platform using visual information. This was relatively easy to implement, as we only needed to put a ArUco tag onto the platform and define the dimensions of this tag in the software. Such tag can be seen in 9.6. We then feed the image from a camera, mounted on the drone to a software processing the information.

¹²RASPBERRY PI FOUNDATION, *Raspberry Pi 3 Model B+*.

¹³Vladimir Ermakov, *mavros - ROS Wiki*.

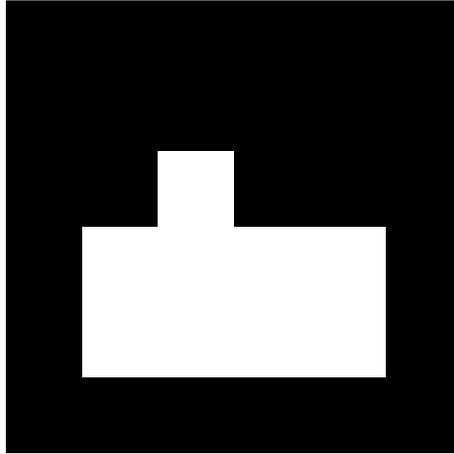


Figure 9.4: Shows an aruco tag. Every tag has an ID encoded embedded as information. Software can use this to identify the tag and calculate a relative position to the tag, if the dimensions of the tag are predefined

We decided to use the ros fiducials package¹⁴. It allows a robot to determine its position and orientation just by looking at fiducial tags.

Drone setup

In order to get a steady video feed from the drone's camera, we decided to use a gimbal mounted at the bottom of a drone facing downwards. We then connected the video camera to a companion computer mounted on the drone. Additionally, we decided to put a Raspberry Pi 3B+ onto the Quadcopter, powered by a power bank. This onboard computer is connected to the camera on the gimbal with a usb cable, which then streams the live video to the companion computer on the drone.

Vehicle setup

In order to intercept the vehicle, we first need to calculate an interception point. To do this we required the gps position and velocity of the vehicle transporting the landing platform. Therefore, we decided to place another flight controller on the landing target, communicating with the drone. This communication presented another challenge, as we needed enough bandwidth to transfer a continuous stream of position and velocity. Additionally, we needed a medium that was able to transfer TCP/IP packets, because we did not have enough time to implement communication via radio. In order to have a fast working prototype, we chose a simple 802.11 Wireless Network with a battery powered router on the vehicle and a WiFi dongle on the drone's companion computer.

Speed matching

After the Drone is positioned at the interception point, it waits for the vehicle to pass the bottom camera. When this occurs, the marker board placed onto the landing platform is

¹⁴Vaughan, *fiducials - ROS Wiki*.

¹⁵Erle Robotics, *Erle-Copter*.

¹⁶PX4 Dev Team, *Pixhawk 1 · PX4 User Guide*.

¹⁷Erle Robotics, *Erle-Brain*.



Figure 9.5: Image of the Drone used for the real world tests. It is equipped with 4 rotors, a flight controller placed in the middle of the frame and a gimbal mounted on the bottom connected to a companion computer used for processing images on the drone. The drone itself is a heavily modified Erle Copter¹⁵ with a Pixhawk¹⁶ flight controller.

detected and the aerial vehicles starts to match the target's ground speed. This velocity is provided via the flight controller placed onto the vehicle. The companion computer on the drone then takes this velocity and sends a command to match it, to the drone's flight controller. This then results in the drone and the vehicle moving at roughly the same speed, giving the drone the ability to go into the final landing phase.

Correcting the relative position and touching down

Before the quadcopter can touch down on the vehicle, it needs to place itself over the moving object. This can only be done, after the vehicle and drone match their speed. Afterwards, the bottom camera is used to track the marker board placed onto the vehicle, providing the drone with a relative position between the landing platform and the drone. The difference in the X and Y axis, between the two objects is then seen as the error, which we try to minimize using a simple P controller. The output of the controller is then added to the speed of the ground vehicle, in order to fly to the board. Finally, after the drone is right above the landing platform, a signal is given to reduce the relative height between the drone and the vehicle's surface. This is done, till a certain threshold is reached, which sends the landing command, finally touching down on the target.

9.5 Challenges and Problems

During development of this thesis project we encounter some problems and challenges we did not expect at the beginning. We tried to solve all of them in the best possible way.

9.5.1 GPS

First of all, calculations with GPS coordinates caused some problems. If the user selects the destination for the drone we receive these coordinates in the backend. For the first approach where we had to calculate the distance between the current car position and



Figure 9.6: Image of the flight controller placed on the landing platform. It is equipped with a GPS module on top and a IMU to estimate the current velocity. The controller used is a Erle Brain 3¹⁷, which consists of a sensor shield mounted onto a raspberry pi. The whole package is battery powered and publishes all data onto a ROS network. The GPS module is placed onto handkerchiefs, in order to vibrations, affecting the compass integrated into the same module.

the selected destination it was not easy to get calculate this distance. Subtracting the destination coordinates and the current car position is not working here. So we had to create our own method which calculates the distance between to given coordinates.

Listing 11: Method to calculate distance between to given coordinates

```

1 def DistanceBetweenCoordinates(self, lat1, lon1, lat2, lon2):
2     degreesToRadians = (pi / 180.0)
3
4     dlong = (lon2 - lon1) * degreesToRadians
5     dlat = (lat2 - lat1) * degreesToRadians
6
7     tempA = pow(sin(dlat / 2.0), 2.0) + cos(lat1 * degreesToRadians) *
8             cos(lat2 * degreesToRadians) * pow(sin(dlong / 2.0), 2.0)
9     tempC = 2.0 * atan2(sqrt(tempA), sqrt(1.0 - tempA))
10    tempD = 6367.0 * tempC # approximate radius of earth in km
11    return 1000.0 * tempD

```

Moreover, we also need a formula to add meters to given coordinates. As before, it is also not possible to simply add a distance to latitude and longitude because you have to convert both in the same unit.

Listing 12: Method for adding meter to GPS coordinates

```

1 def OffsetGps(self, inputLatitude, inputLongitude, meterX, meterY):
2     dLatitude = meterX / 6378137.0 # Radius of earth in meter
3     dLongitude = meterY / (6378137.0 * cos(pi * inputLatitude / 180.0))
4
5     newLatitude = inputLatitude + (dLatitude * (180.0 / pi))
6     newLongitude = inputLongitude + (dLongitude * (180.0 / pi))
7     return newLatitude, newLongitude

```

Overall, we found a solution for every problem we had with GPS coordinates calculations. Nevertheless, these calculations are not 100% accurate but sufficient for us.

9.5.2 Waypoints

Creating and using waypoints for our mission also leads to some problems. Debugging missions is really time consuming because it is only possible with trial and error. The first problem with waypoints came up when we realized that our drone skips the first point in the waypoint array, which is uploaded to the drone. We tried to find a solution to this problem, but we gave up after wasting multiple hours. Instead, we chose to just ignore the problem and send a placeholder waypoint in the first place. Even in the documentation is no available solution for this bug.

Secondly, we wanted to launch the drone's takeoff procedure by using a waypoint. This has led us to the next problem with waypoints, because it just does not work when you create a waypoint with the takeoff command. It will be ignored and the drone stays on the ground. Like in the previous problem, there is no solution for this problem in the documentation. Therefore, we have to takeoff in the guided mode and afterwards start the mission by switching to the auto flight mode.

The last problem we are having with waypoints is the callback function. If the drone reaches a new point it should automatically send a message to this function. This message contains information about the reached waypoint. However, this function does not work as expected. Instead of triggering when a waypoint is reached, it called the callback randomly with often incorrect sequence numbers.

Listing 13: Waypoint reached callback function

```

1 def WaypointsCb(self, data):
2     rospy.loginfo("Waypoint reached: " + str(data.current_seq))
3     self.currWp = data.current_seq

```

As you can see in code listing 13, every time the drone reaches a new waypoint it will be logged and the current waypoint variable will be set. Sometimes it works but sometimes it logs random values and another time it logs the same waypoint 20 times consecutively. Following from this reasons, there is not enough reliance on this function. Therefore, we have to check the position of the drone on our own if we want to know where the drone is and if it already reached a way point or not.

9.5.3 Speed matching

Before the drone can land on the moving vehicle, it first needs to match its speed. At first, we decided to use the relative velocity between the vehicle and the drone, to accelerate the drone accordingly. We know the relative velocity, as we can get the position of the tag, if it is spotted by the camera. By integrating this list of relative positions and timestamps, we

can easily calculate the velocity. After we did all those calculations, we looked at sending the corresponding commands to the drone via the MAVLink protocol. Luckily for us, so we thought at first, there is a message called `SET_POSITION_TARGET_LOCAL_NED`¹⁸ defined in the common message definitions¹⁹ for ArduCopter. Among other things, the message includes 3 fields for acceleration in the X, Y, and Z axis. Easy done, just implement the message and off we go, flying a drone. Sadly, this was not the case, as nothing happened, when we sent the correct message to the drone running the ArduCopter flight stack. The message does exist, as said in the documentation, but hidden on the second page of google, we found out, that the 3 fields we wanted to use are currently not implemented in software. For us, this resulted in a big amount of more work, setting back our planning. Instead of working with the relative velocity, we now had to put sensors onto the vehicle, to get its absolute velocity.

9.5.4 Camera control

Another challenge we didn't expect occurred, when we tried to use a gimbal mounted on, and controlled by the drone. Our idea was to use a camera to detect the tag underneath the moving quadcopter. For this, we mounted a camera onto a computer-controlled gimbal, which would then be aimed straight down. Sadly, as a default mode, our Tarot gimbal²⁰ is aimed straight ahead. Therefore, we needed to use the PWM camera control functionality of the PX4 firmware to control the two servos on the gimbal. Easier said than done, as the flight controller is typically set up to control the camera via two RC channels received on the radio. A quick look at the PX4 documentation revealed, that this behavior could be changed by some parameters. Sadly, similar to other problems we had with the firmware, the documentation and reality vary drastically. Setting the parameters resulted in no visible changes of the gimbal orientation, not even after a controller restart and after updating to the newest flight controller firmware. This is a major problem of both, the PX4 firmware and the ArduCopter flight stack, as the documentation is often not kept up to date. Luckily for us, after hours of tinkering, we could solve this problem by overwriting the default mixer behavior of the Pixhawk. Mixing is used to translate force commands to actuator commands. To do this, we just had to create the correct file, as shown in 14, and upload it to the SD-Card. The controller then reads this file and changes the pitch accordingly.

Listing 14: Mixer file used to point a gimbal downwards.

```

1 # roll
2 M: 1
3 O:      10000  10000      0 -10000  10000
4 S: 2 0  10000  10000      0 -10000  10000
5
6 # pitch
7 M: 1
8 O:      10000  10000      0 -10000  10000
9 S: 2 1  10000  10000      0 -10000  10000
10
11 # yaw
```

¹⁸https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_LOCAL_NED

¹⁹*common.xml · MAVLink Developer Guide.*

²⁰WENZHOU TAROT AVIATION TECHNOLOGY CO.,LTD, *TAROT T4-3D 3-AXIS BRUSHLESS GIMBAL FOR GOPRO TL3D01.*

```

12 M: 1
13 O: 10000 10000 0 -10000 10000
14 S: 2 2 10000 10000 0 -10000 10000

```

9.5.5 Simulation versus Reality

For test purposes we created a simulation environment for the drone and the car. The major reason why we decided to develop our software in a simulation is because it speeds up the project process. We are able to develop at home, on the way to school or even when the weather is bad. Another reason for a simulation is the functionality, if it works in the simulation it also should work in reality when we fly outdoor. The functionality of the simulation is the biggest problem. We spent much time in simulating the drone but for inexplicable reasons it does not work as good outdoor as in the simulation.

9.6 Testing

Testing a software is one major part of developing an application. When we developed the AARD Application we spent a lot of hours just in testing and debugging. Testing a program on a drone is only possible with trial and error which consumes much time of the developer, especially if we need to test in a realistic environment. Before every testing session outside we had to do the following steps, consuming a lot of time and energy:

1. Check if the batteries are charged
2. Go outside with following items:
 - (a) Notebooks
 - (b) Drone + batteries
 - (c) Radio transmitter and receiver
3. Connect notebook to drone
4. Calibrate all sensors:
 - (a) Compass
 - (b) Accelerometer
5. Start testing

Just the preparations for testing outside took about 15 to 30 minutes. Moreover, when the battery level got too low during testing we had to stop and wait for multiple hours till the batteries were fully charged. Another big problem of testing the drone outside was the weather and the wind. If the environmental conditions do not allow it, for examples when it snows, rains or the wind is too strong, we could not test and had to wait till the conditions changed. This can lead to major time delays, as we could not calculate the weather into our time management.

All these problems were a incentive for us to rather focus on getting a reliable simulation working, instead of testing outside. We decided to test everything in Gazebo which is a software for simulation in ROS. It includes a 3D model of our drone and an ArUco marker, which represents the moving platform. The effort required to get this simulation working was another part we underestimated. In the end, we spent about two weeks to get from a clean system install to a fully set up computer running the firmware and the simulation required to develop without any flight controller hardware.

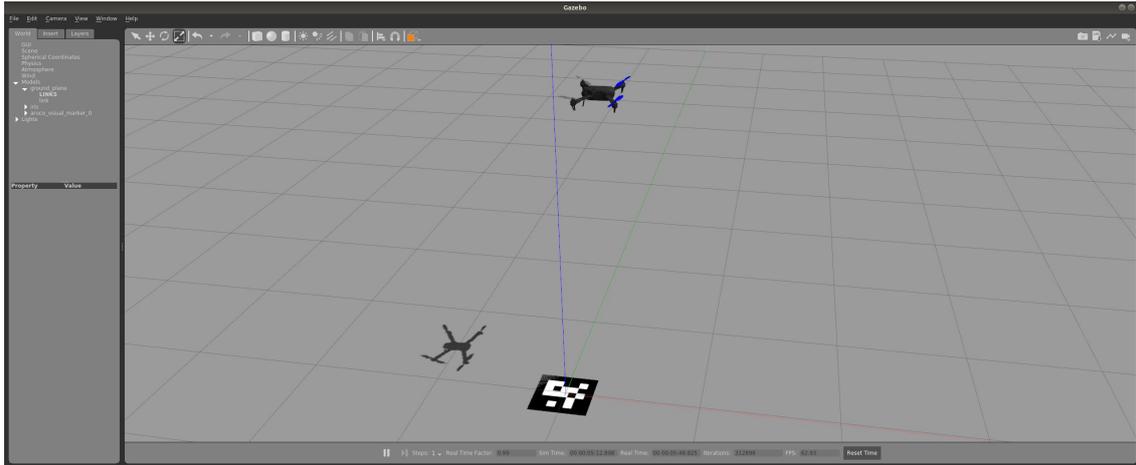


Figure 9.7: Shows the basic structure of our simulation.

However, simulating a drone also creates disadvantages and difficulties, which one might not expect. First of all, simulated environments do not represent all the factors present in reality. Hence, it can not be expected that the drone works in reality if it works in the simulation. Of course simulations provide a good approximation of the real conditions, therefore often only small changes are needed to get the program working with real hardware.

Moreover, trying to simulate an realistic environment takes a lot of memory and computational effort. This provides another problem for us, as we typically work on notebooks, without large amounts of processing power or memory capacity. We could not easily solve this problem, as we typically developed while we were in school, making it impossible for us to use a big tower pc. One possible solution, would be using a cloud-based technology to have all the power in some datacenter and stream the visual outputs to a small workstation via the internet.

Chapter 10

Conclusion

The Autonomous Aerial Reconnaissance Drone (AARD) project is a first step towards a fully autonomous aerial drone landing on a constantly moving vehicle. Multiple prototypes have been constructed and programmed and a vast amount of work has been done to help the next generation of students in understanding the ArduCopter and PX4 flight stacks. Furthermore, a git repository with a fully functioning development environment has been created, which enables other students to instantly start working on other projects.

10.1 Development

A big amount of time working on this thesis was spent on creating a functioning development environment, including operating system independence, easy to use communication and a simulation environment. This was needed, as no similar work has been done before at this educational institution, giving the authors no base to work on.

10.1.1 Docker

One technology playing a key role in this thesis is the docker container engine¹, which provides an easy way to set up all the necessary software needed to run this project on any computer. If used together with Docker Compose², a tool for running multiple containers at once, every service needed to run the application can be started with only one command. Furthermore, all containers have been set up to provide a graphical user interface, if the correct NVIDIA drivers are installed. Another key advantage of using this bundle of software is the easy dependency management. To update all the software, only small changes need to be made to the Dockerfiles, giving the user a hassle-free experience in updating the dependencies.

¹Docker Inc., *Docker*.

²Docker Inc., *Docker*.

10.1.2 Robot Operating System

The second technology playing a key role is the Robot Operating System (ROS). Although it is called "operating system", ROS should rather be classified as a software stack, which provides a unified way for multiple services to communicate between each other. These ROS nodes can exchange information, as long as they share the same network, making it easy to transfer data between our ground station and drone. Furthermore, the open source community is providing a massive amount of free-to-use software compatible with ROS, giving developers the ability to rapidly create prototypes.

10.1.3 Arducopter versus Px4

The third and last component required to run the software created in this thesis is a flight controller firmware. This can be ArduCopter, PX4 or any other MAVLink compatible software. The firmware runs on the flight controller placed on the drone, controlling the motion of the vehicle. The two major stacks tested as part of this thesis were ArduCopter and PX4.

ArduCopter is an open source project created by volunteers in their free time. One big advantage of this stack is its compatibility with a variety of hardware controllers, making it one of the most used systems. This results in a large amount of developers working on different parts of the software at once, providing many features and customizing options for the user. Although this might seem like an advantage at first, the documentation on the project is often left untouched, even if the software changes.

The PX4 flight control software is another way to control the drone. Like ArduCopter, PX4 is open-source and developed by a large community. The system was first created at the ETH Zürich aimed to be used in computer vision projects. Differentiating it from the other stacks, PX4 only supports a rather small amount of controllers, all based on the Pixhawk series. This rather small list of controllers makes it easy for the user to set up the firmware, without running into any problems. Another difference noticed by the authors, was the good and extensive documentation, making it easy to fix any problems that might occur.

Both stacks are compatible with the QGroundControl software, which enables the user to configure the drone before flight. Additionally, autonomous missions can be created, executed and saved to a file, which helped the authors in debugging their own software.

10.1.4 Simulation

In order to rapidly develop and test software, a virtual simulation environment has been set up by the authors. Both flight Stacks, ArduCopter and PX4, include simulators based on the gazebo engine, with the flight controller either running directly on the hardware (Hardware in the Loop) or on the host computer (Software in the Loop). Both of these environments include an integration into ROS, making it easy to spawn new drones. Furthermore, the source code is provided as open source, giving the authors the ability to change the flight characteristics of the quadcopters. Although these simulators are a good way to develop, they do not represent a real test. Nevertheless, these environments provide a good way to test the software beforehand.

10.2 AARD Application

Next to the written part of this thesis, as software prototype was developed, enabling the user to create and execute autonomous missions. In order to maintain some form of structure, everything was split into two parts.

10.2.1 AARD Front-End

The front-end is responsible for displaying the user-friendly interface. It can be accessed via any common, up-to-date web browser. In this interface, a goal location can be specified, which the drone needs to explode. Additionally, the current location of the car and the aerial vehicle are displayed. After all the required input is done, the mission will be executed and additional information, such as battery status, will be shown. To enable communication between the drone and the web-server rosbridge, a javascript API for ROS, was utilized, making it easy to subscribe or publish on ROS topics.

10.2.2 AARD Back-End

In order to translate the user input into commands for the drone, a back-end was created. It gathers all necessary information from the drone and the front-end and then creates the control commands for the drone. Additionally, it analyzes the image data generated by the bottom facing camera and then calculates the velocities needed to successfully land on the vehicle. All this information is then taken and processed via a python software running in a docker container. Afterwards, the commands are encoded into a MAVLink message, using mavros and sent to the flight controller, either via radio communication or by a direct cable connection.

Together, the front-end and the back-end provide an easy way for future students to get started with developing an autonomous drone. Multiple ways to control the drone, such as autonomous missions and velocity setpoints are included as examples and an easy integration into an intuitive ui was created. All in all, the software is far from complete, but it presents a good proof-of-concept or prototype. With its easy expandability through ROS, a ready to use development environment and the simulation everything has been set up for future teams to get started on development.

10.3 Outlook

One big problem encountered within this thesis was the tracking of a moving vehicle from the drone. In future works, the camera could be replaced with specialized sensors mounted on the vehicle and the drone, in order to calculate the relative position from the vehicle to the drone. Another part that could be improved is the route between the start, the goal and back. Currently, the drone flies on a straight line, without any obstacle detection. This could be improved, by mounting a depth camera onto drone, looking forward and creating a map of the environment. If something in the path is detected, the drone should autonomously avoid the obstacle. Similar behavior could be integrated into the landing phase, where the drone needs to place itself above the drone. If the user for some reason decides to make a sharp 90 degree turn in an urban environment, the drone could crash into obstacles. This could be avoided, if the route of the vehicle would be entered beforehand.

Index

Authors Index

- (U.S.), Technology, iii, 27
- Aaron Blasdel, 15
- Amazon Robotics, 32
- Amazon.com, Inc., 2
- Angular University, 37
- ArduPilot, 16
- ArduPilot Dev Team, 13, 17, 18, 41, 43, 44
- Atlassian, 6, 7
- Banks, J., 13
- Canonical Ltd., 12
- Carson, J.S., 13
- Dave Coleman, 13
- Dave Hershberger, 14
- David Gossow, 14
- Deutsche Post AG, 2
- Dirk Thomas, 15
- Docker Inc., 12, 13, 40, 56
- Dorian Scholz, 15
- Dronecode Project, Inc., 17, 19, 21, 41
- Eitan Marder-Eppstein, 12
- Erle Robotics, 49
- Graham, Ross; McCabe, Hugh; and Sheridan, Stephen, 31
- Himangshu Kalita; Steven Morad; Aaditya Ravindran; Jekan Thangavelautham, 31
- Hyperloop One, 2
- John Hsu, 13
- Josh Faust, 14
- Jun-tao Li; Hong-jian Liu, 32
- Ken Conley, 10, 11
- Kristen Erickson, 31
- LeanKit Inc., 8
- Meier, Lorenz, 19
- Microsoft, 21
- Mohri, Mehryar, 5
- Moore, Gordon E., 5
- Nate Koenig, 13
- Nelson, B.L., 13
- Open Source Robotics Foundation, 21
- Open Source Robotics Foundation, Inc., 9–11, 13
- Opensource.org, 38
- PX4, 21
- PX4 Dev Team, 18, 49
- QGroundControl, 17
- Radigan, Dan, 6
- RASPBERRY PI FOUNDATION, 48
- Rob, Bell, 35
- Rostamizadeh, Afshin, 5
- Rupp, Karl, 5
- SciPy developers, 26
- SEMATECH, International, iii, 27
- Smith, Steven, iii, 26
- Sreeja Banerjee, 31
- Standards, National Institute of, iii, 27
- Talwalkar, Ameet, 5
- Technologies, Unity, 21
- The Boring Company, 2
- Tully Foote, 12
- Vaughan, Jim, 49
- Vladimir Ermakov, 43, 48

Vue, 37, 38

TECHNOLOGY CO.,LTD, 53

WENZHOU TAROT AVIATION

Wim Meeussen, 12

Literature Index

ArduPilot Open Source Autopilot, 16

DHL Parcelcopter, 2

MAVLINK Common Message Set, 17

MAVLink, 17

MoveIt

MoveIt

Motion Planning Framework, 32

NIST-International Sematech E-handbook: NIST Handbook 151. Dataplot : NIST

Handbook 148, iii, 27

QGC, 19

QGroundControl, 17

SITL Simulator (Software in the Loop), 13

SciPy.org, 26

SiK Telemetry Radio, 18

TAROT T4-3D 3-AXIS BRUSHLESS GIMBAL FOR GOPRO TL3D01, 53

42 Years of Microprocessor Trend Data, 5

A beginner's guide to Big O notation - Rob Bell, 35

*A COMPARATIVE STUDY OF UNDERWATER ROBOT PATH PLANNING
ALGORITHMS FOR ADAPTIVE SAMPLING IN A NETWORK OF
SENSORS*, 31

Amazon Prime Air, 2

Amazon Robotics, 32

Angular SPA: Why Single Page Applications?, 37

Automatic Takeoff — Plane documentation, 44

Bags - ROS Wiki, 11

common.xml · MAVLink Developer Guide, 42, 43, 53

Cramming More Components Onto Integrated Circuits, 5

Design Optimization of Amazon Robotics, 32

Digital Signal Processing: A Practical Guide for Engineers and Scientists, iii, 26

Discrete-event System Simulation, 13

Docker, 12, 56

Docker Compose, 13

Dockerfile reference, 13

Dronecode - The Open Source UAV Platform, 19

Dronecode SDK, 19

Enterprise Application Container Platform, 40, 56

Erle-Brain, 49

Erle-Copter, 49

fiducials - ROS Wiki, 49

Flight Modes · PX4 Developer Guide, 19

- Foundations of Machine Learning*, 5
- Gazebo*, 13, 21
gazebo_ros_pkgs - ROS Wiki, 13
- Hyperloop One*, 2
- Introduction — Vue.js*, 37
Introduction · MAVLink Developer Guide, 19, 41
- Master* - ROS Wiki, 11
mavros - ROS Wiki, 43, 48
Messages - ROS Wiki, 11
Microsoft Research AI (MSR AI), 21
Mission Commands — Mission Planner documentation, 41
Mission Planner Home, 17
Mission Planning — Copter documentation, 43
- Nodes* - ROS Wiki, 9
- Open Source for Drones*, 19
Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research: Microsoft/AirSim, 21
- Packages* - ROS Wiki, 9
Parameter Server - ROS Wiki, 11
Path Planning and Navigation Inside Off-World Lava Tubes and Caves, 71
Pathfinding in Computer Games, 31
Pixhawk 1 · PX4 User Guide, 18, 49
Pre-Arm Safety Check — Copter documentation, 44
- Raspberry Pi 3 Model B+*, 48
rosbridge_suite - ROS Wiki, 38
roscpp - ROS Wiki, 11
rosservice - ROS Wiki, 10
rostopic - ROS Wiki, 10
rqt - ROS Wiki, 15
rviz - ROS Wiki, 14
- Services* - ROS Wiki, 10
Simple multirotor simulator with MAVLink protocol support: PX4/jMAVSim, 21
Simulation · PX4 Developer Guide, 21
- Telemetry Radio Regional Regulations*, 18
tf2 - ROS Wiki, 12
The Boring Company, 2
The History of Pixhawk, 19
The Mars Rovers :: NASA Space Place, 31
The official documentation site for Vue.js. Contribute to vuejs/vuejs.org development by creating an account on GitHub, 38
Topics - ROS Wiki, 10
- Ubuntu 18.04.1 LTS (Bionic Beaver)*, 12

Unity, 21

Unreal Engine, 21

What are WIP limits?, 6

What is a Kanban Board?, 7

What is Kanban?, 8

Why do we send robots to space? :: NASA Space Place, 31

Bibliography

- Amazon Robotics. *Amazon Robotics*. Amazon Robotics. URL: <https://www.amazonrobotics.com/#/vision> (visited on 11/02/2018).
- Amazon.com, Inc. *Amazon Prime Air*. URL: <https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011> (visited on 10/04/2018).
- Angular University. *Angular SPA: Why Single Page Applications?* Angular University. Apr. 21, 2017. URL: <https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/> (visited on 02/05/2019).
- ArduPilot. *ArduPilot Open Source Autopilot*. URL: <http://ardupilot.org/> (visited on 10/29/2018).
- ArduPilot Dev Team. *Automatic Takeoff — Plane documentation*. Automatic Takeoff. 2016. URL: <http://ardupilot.org/plane/docs/automatic-takeoff.html> (visited on 01/04/2019).
- *Mission Commands — Mission Planner documentation*. Mission Commands. 2016. URL: http://ardupilot.org/planner/docs/common-mavlink-mission-command-messages-mav_cmd.html (visited on 01/03/2019).
- *Mission Planner Home*. URL: <http://ardupilot.org/planner/> (visited on 10/29/2018).
- *Mission Planning — Copter documentation*. Mission Planning. 2016. URL: <http://ardupilot.org/copter/docs/common-mission-planning.html> (visited on 01/03/2019).
- *Pre-Arm Safety Check — Copter documentation*. URL: http://ardupilot.org/copter/docs/prearm_safety_check.html#prearm-safety-check (visited on 01/04/2019).
- *SiK Telemetry Radio*. URL: <http://ardupilot.org/copter/docs/common-sik-telemetry-radio.html#common-sik-telemetry-radio> (visited on 10/30/2018).
- *SITL Simulator (Software in the Loop)*. URL: <http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html> (visited on 10/31/2018).
- *Telemetry Radio Regional Regulations*. URL: <http://ardupilot.org/copter/docs/common-telemetry-radio-regional-regulations.html#common-telemetry-radio-regional-regulations> (visited on 10/30/2018).
- Atlassian. *Kanban - A brief introduction | Atlassian*. Kanban. URL: <https://www.atlassian.com/agile/kanban> (visited on 10/31/2018).
- *What is a Kanban Board?* Atlassian. URL: <https://www.atlassian.com/agile/kanban/boards> (visited on 10/31/2018).
- Banks, J., J.S. Carson, and B.L. Nelson. *Discrete-event System Simulation*. Prentice Hall, 2010. ISBN: 978-0-13-606212-7. URL: <https://books.google.at/books?id=cqSNnmrqqbQC>.
- Canonical Ltd. *Ubuntu 18.04.1 LTS (Bionic Beaver)*. URL: <http://releases.ubuntu.com/18.04/> (visited on 10/30/2018).
- common.xml · MAVLink Developer Guide*. MAVLINK Common Message Set. URL: https://mavlink.io/en/messages/common.html#MAV_AUTOPILOT_GENERIC_WAYPOINTS_AND_SIMPLE_NAVIGATION_ONLY (visited on 01/04/2019).
- Dave Hershberger, David Gossow, and Josh Faust. *rviz - ROS Wiki*. rviz. Aug. 14, 2015. URL: <http://wiki.ros.org/rviz> (visited on 10/31/2018).

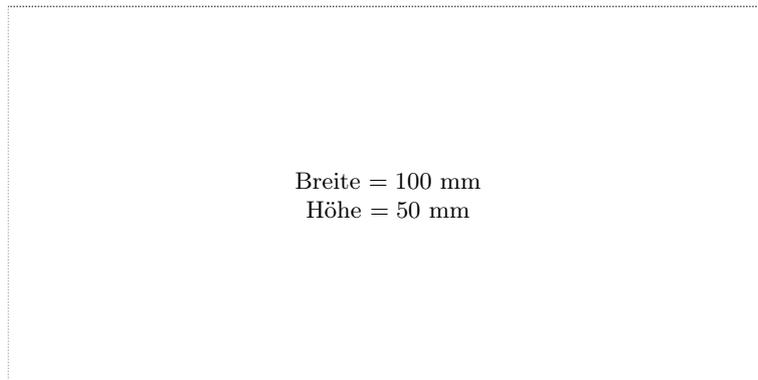
- Deutsche Post AG. *DHL Parcelcopter*. DHL Parcelcopter. In collab. with Alexander Edenhofer. URL: <https://www.dpdhl.com/en/media-relations/specials/dhl-parcelcopter.html> (visited on 10/04/2018).
- Dirk Thomas, Dorian Scholz, and Aaron Blasdel. *rqt - ROS Wiki*. URL: <http://wiki.ros.org/rqt> (visited on 11/01/2018).
- Docker Inc. *Docker*. Docker. URL: <https://www.docker.com/> (visited on 10/30/2018).
- *Docker Compose*. Docker Compose. URL: <https://docs.docker.com/compose/> (visited on 10/30/2018).
- *Dockerfile reference*. Dockerfile reference. URL: <https://docs.docker.com/engine/reference/builder/> (visited on 10/30/2018).
- *Enterprise Application Container Platform*. Docker. URL: <https://www.docker.com/> (visited on 03/05/2019).
- Dronecode Project, Inc. *Dronecode - The Open Source UAV Platform*. Dronecode. URL: <https://www.dronecode.org/> (visited on 03/19/2019).
- *Dronecode SDK*. Dronecode. URL: <https://www.dronecode.org/sdk/> (visited on 03/20/2019).
- *Flight Modes · PX4 Developer Guide*. Flight Modes. URL: https://dev.px4.io/en/concept/flight_modes.html (visited on 03/19/2019).
- *Introduction · MAVLink Developer Guide*. MAVLink Developer Guide. URL: <https://mavlink.io/en/> (visited on 03/20/2019).
- *Introduction · MAVLink Developer Guide*. MAVLink Developer Guide. URL: <https://mavlink.io/en/> (visited on 01/03/2019).
- *MAVLINK Common Message Set*. URL: <https://mavlink.io/en/messages/common.html> (visited on 10/30/2018).
- *Open Source for Drones*. PX4 Open Source Autopilot. URL: <https://px4.io/> (visited on 03/19/2019).
- *QGC*. QGroundControl - Drone Control. URL: <http://qgroundcontrol.com/> (visited on 03/20/2019).
- *Simulation · PX4 Developer Guide*. Simulation. URL: <https://dev.px4.io/en/simulation/> (visited on 03/21/2019).
- Dronecode Project, Inc. *MAVLink*. URL: <https://mavlink.io/en/> (visited on 10/30/2018).
- Erle Robotics. *Erle-Brain*. URL: http://docs.erlerobotics.com/brains/discontinued/erle_brain (visited on 03/14/2019).
- *Erle-Copter*. URL: http://docs.erlerobotics.com/erle_robots/erle_copter (visited on 03/14/2019).
- Graham, Ross; McCabe, Hugh; and Sheridan, Stephen. *Pathfinding in Computer Games*. Vol. 4. 2003. URL: https://arrow.dit.ie/itbj/vol4/iss2/6?utm_source=arrow.dit.ie%2Fitbj%2Fvol4%2Fiss2%2F6&utm_medium=PDF&utm_campaign=PDFCoverPages.
- Himangshu Kalita; Steven Morad; Aaditya Ravindran; Jekan Thangavelautham. *Path Planning and Navigation Inside Off-World Lava Tubes and Caves*. URL: <https://arxiv.org/pdf/1803.02818.pdf>.
- Hyperloop One. *Hyperloop One*. URL: <https://hyperloop-one.com/> (visited on 10/20/2018).
- John Hsu, Nate Koenig, and Dave Coleman. *gazebo_ros_pkgs - ROS Wiki*. URL: http://wiki.ros.org/gazebo_ros_pkgs (visited on 10/31/2018).
- Jun-tao Li; Hong-jian Liu. *Design Optimization of Amazon Robotics*. May 30, 2016. URL: <http://article.sciencepublishinggroup.com/pdf/10.11648.j.acis.20160402.17.pdf>.
- Ken Conley. *rosparam - ROS Wiki*. URL: <http://wiki.ros.org/rosparam> (visited on 10/16/2018).
- *rosservice - ROS Wiki*. URL: <http://wiki.ros.org/rosservice> (visited on 10/29/2018).
- *rostopic - ROS Wiki*. URL: <http://wiki.ros.org/rostopic> (visited on 10/29/2018).

- Kristen Erickson. *The Mars Rovers :: NASA Space Place*. The Mars Rovers. URL: <https://spaceplace.nasa.gov/mars-rovers/en/> (visited on 11/02/2018).
- *Why do we send robots to space? :: NASA Space Place*. Why do we send robots to space? URL: <https://spaceplace.nasa.gov/space-robots/en/> (visited on 11/02/2018).
- LeanKit Inc. *What is Kanban?* LeanKit. Oct. 30, 2018. URL: <https://leankit.com/learn/kanban/what-is-kanban/> (visited on 10/31/2018).
- Meier, Lorenz. *The History of Pixhawk*. Auterion. URL: <https://auterion.com/the-history-of-pixhawk/> (visited on 03/19/2019).
- Microsoft. *Microsoft Research AI (MSR AI)*. Microsoft Research. URL: <https://www.microsoft.com/en-us/research/lab/microsoft-research-ai/> (visited on 03/21/2019).
- *Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research: Microsoft/AirSim*. original-date: 2017-02-14T00:52:29Z. Mar. 21, 2019. URL: <https://github.com/Microsoft/AirSim> (visited on 03/21/2019).
- Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, Aug. 17, 2012. 427 pp. ISBN: 978-0-262-30473-3.
- Moore, Gordon E. *Cramming More Components Onto Integrated Circuits*. McGraw-Hill, 1965. 4 pp.
- MoveIt! Motion Planning Framework*. MoveIt! URL: <https://moveit.ros.org/> (visited on 01/01/2019).
- Open Source Robotics Foundation. *Gazebo*. Gazebo. URL: <http://gazebo.org/> (visited on 03/21/2019).
- Open Source Robotics Foundation, Inc. *Bags - ROS Wiki*. URL: <http://wiki.ros.org/Bags> (visited on 10/16/2018).
- *Gazebo*. URL: <http://gazebo.org/> (visited on 10/31/2018).
- *Master - ROS Wiki*. URL: <http://wiki.ros.org/Master> (visited on 10/16/2018).
- *Messages - ROS Wiki*. URL: <http://wiki.ros.org/Messages> (visited on 10/16/2018).
- *Nodes - ROS Wiki*. URL: <http://wiki.ros.org/Nodes> (visited on 10/16/2018).
- *Packages - ROS Wiki*. URL: <http://wiki.ros.org/Packages> (visited on 10/16/2018).
- *Parameter Server - ROS Wiki*. URL: <http://wiki.ros.org/Parameter%20Server> (visited on 10/16/2018).
- *Services - ROS Wiki*. URL: <http://wiki.ros.org/Services> (visited on 10/16/2018).
- *Topics - ROS Wiki*. URL: <http://wiki.ros.org/Topics> (visited on 10/16/2018).
- Opensource.org. *The MIT License | Open Source Initiative*. The MIT License. URL: <https://opensource.org/licenses/MIT> (visited on 02/05/2019).
- PX4. *Simple multirotor simulator with MAVLink protocol support: PX4/jMAVSim*. jMAVSim. original-date: 2015-06-13T23:11:25Z. Mar. 21, 2019. URL: <https://github.com/PX4/jMAVSim> (visited on 03/21/2019).
- PX4 Dev Team. *Pixhawk 1 · PX4 User Guide*. Pixhawk 1 Flight Controller. Oct. 30, 2018. URL: https://docs.px4.io/en/flight_controller/pixhawk.html (visited on 10/31/2018).
- QGroundControl. *QGroundControl*. QGroundControl. URL: <http://qgroundcontrol.com/> (visited on 10/29/2018).
- Radigan, Dan. *What are WIP limits?* Atlassian. URL: <https://www.atlassian.com/agile/kanban/wip-limits> (visited on 10/31/2018).
- RASPBERRY PI FOUNDATION. *Raspberry Pi 3 Model B+*. Raspberry Pi. URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/> (visited on 02/04/2019).
- Related Projects | MoveIt!* MoveIt! Related Projects. URL: https://moveit.ros.org/documentation/related_projects/ (visited on 01/01/2019).

- Rob, Bell. *A beginner's guide to Big O notation - Rob Bell*. A beginner's guide to Big O notation. 2008. URL: <https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/> (visited on 01/03/2019).
- rosbridge_suite* - *ROS Wiki*. *rosbridge_suite*. Oct. 23, 2017. URL: http://wiki.ros.org/rosbridge_suite (visited on 02/05/2019).
- Rupp, Karl. *42 Years of Microprocessor Trend Data*. Feb. 15, 2018. URL: <https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/> (visited on 10/25/2018).
- SciPy developers. *SciPy.org*. URL: <https://www.scipy.org/> (visited on 03/14/2019).
- Smith, Steven. *Digital Signal Processing: A Practical Guide for Engineers and Scientists*. Google-Books-ID: CdtQBAAQBAJ. Elsevier, Oct. 22, 2013. 666 pp. ISBN: 978-0-08-047732-9.
- Sreeja Banerjee. *A COMPARATIVE STUDY OF UNDERWATER ROBOT PATH PLANNING ALGORITHMS FOR ADAPTIVE SAMPLING IN A NETWORK OF SENSORS*. Aug. 2014. URL: <https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1099&context=computerscidiss>.
- Standards, National Institute of, Technology (U.S.), and International SEMATECH. *NIST-International Sematech E-handbook: NIST Handbook 151. Dataplot : NIST Handbook 148*. National Institute of Standards and Technology, 2003. URL: <https://books.google.at/books?id=ER-YDAEACAAJ>.
- Technologies, Unity. *Unity*. Unity. URL: <https://unity.com/frontpage> (visited on 03/21/2019).
- The Boring Company. *The Boring Company*. URL: <https://www.boringcompany.com/> (visited on 10/20/2018).
- Tully Foote, Eitan Marder-Eppstein, and Wim Meeussen. *tf2* - *ROS Wiki*. URL: <http://wiki.ros.org/tf2> (visited on 10/29/2018).
- Unreal Engine*. Unreal Engine. URL: <https://www.unrealengine.com/en-US/what-is-unreal-engine-4> (visited on 03/21/2019).
- Vaughan, Jim. *fiducials* - *ROS Wiki*. URL: <http://wiki.ros.org/fiducials> (visited on 03/07/2019).
- Vladimir Ermakov. *mavros* - *ROS Wiki*. URL: <http://wiki.ros.org/mavros> (visited on 01/04/2019).
- Vue. *Introduction* — *Vue.js*. Introduction. URL: <https://vuejs.org/v2/guide/#What-is-Vue-js> (visited on 02/05/2019).
- *The official documentation site for Vue.js. Contribute to vuejs/vuejs.org development by creating an account on GitHub*. original-date: 2014-01-05T03:11:14Z. Feb. 5, 2019. URL: <https://github.com/vuejs/vuejs.org> (visited on 02/05/2019).
- WENZHOU TAROT AVIATION TECHNOLOGY CO.,LTD. *TAROT T4-3D 3-AXIS BRUSH-LESS GIMBAL FOR GOPRO TL3D01*. URL: <http://www.tarotrc.com/Product/Detail.aspx?Lang=en&Id=6dbcf6be-be09-4429-87c5-ca41c426aae0> (visited on 03/20/2019).
- X-Plane 11 Flight Simulator | More Powerful. Made Usable*. X-Plane. URL: <https://www.x-plane.com/> (visited on 03/21/2019).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —