HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

Autonomous universal Mapping and Navigation

Ausgeführt im Schuljahr 2021/22 von:

System Design, Sensory Inputs, Mapping
Lukas Leskovar — 5BHIF

Motion Planning, Exploration, Collision Detection
Fabian Kleinrad — 5BHIF

Betreuer / Betreuerin:

MMag. Dr. Michael Stifter

Wiener Neustadt, am 4. April 2022

Abgabevermerk: Übernommen von:

Chapter 1

Eidestattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegeben Quellen und Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Wiener Neustadt am 4. April 2022

Verfasser / Verfasserinnen:

Lukas Leskovar

Fabian Kleinrad

Contents

Chapter 2

Acknowledgement

First and foremost, the authors would like to thank F-WuTS and robo4you for providing the equipment and facilities used during the development of the Autumn Project. Furthermore, we want to thank our supervisor MMag. Dr. Michael Stifter, who's support, motivation and advice were of integral importance for the authors and this thesis.

Author: Fabian Kleinrad
My gratitude is due to all the people who have helped me with topic or non-topic specific questions. This made realizing this project possible and directly affected the quality of the outcome.
Author: Lukas Leskovar
I would also like to thank everyone who read this thesis and contributed insight from a non-technical standpoint, improving its overall quality and readability.
Most notably, I am very grateful for my family and friends and their support throughout the past five years.

Chapter 3

Kurzfassung

Heutzutage unterstützen uns Computer mehr als je zuvor in allen Bereichen unserer Arbeit. Von digital gestürzter Gebäudekonstruktion, bis hin zu Computer basierter Waldwachsbeurteilung. Autumn bietet hierbei eine Lösung, diese Vorhaben durch eine autonome und universell einsetzbare Drohne zu vereinfachen. Die Drohne erstellt ein realitätsnahes 3-D Modell der Umgebung. Dies kann beispielsweise genutzt werden, um Maße zu entnehmen oder eine Momentaufnahme eines Vorhabens festzuhalten. Aufgrund der verwendeten Technologien ist die Drohne unabhängig von äußeren Lokalisierungsmechanismen und kann dadurch in abgelegenen Geländen eingesetzt werden. Realisiert wird dies durch den Einsatz eines SLAM-Algorithmus, um das Mapping zu ermöglichen, und einen RRT* Algorithmus für die Pfadfindung.

Chapter 4

Abstract

Computers nowadays are integrated into a wide variety of processes. Ranging from computer-aided construction to software-based monitoring of forests. Autumn proposes a solution of using an autonomous and universally deployable drone to simplify these tasks. The drone generates a realistic model of the environment, which can be used to extract measurements or capture the momentary progress. Furthermore, Autumn uses technologies that enable the drone to work independently of any external localisation mechanism, which allows for deployment in secluded areas. This characteristic can be realised by using a SLAM algorithm, needed to map the environment in combination with an RRT* algorithm, which handles the path-planning.

Chapter 5

Introduction

Author: Lukas Leskovar

5.1 The Evolution of Robotics

Robotic research has always utilized concepts, processes, and methods of different scientific disciplines such as physics, mathematics, and biology to improve application and aid human needs. Because of this, industrial, medical, and even agricultural sectors have used technologies and products developed by researchers to improve workflows and alleviate employees from performing exhausting tasks. This relationship ranges back to the early ages of information technology in the 1950s and 1960s, in which many developments on production robots and Artificial Intelligence (AI) have been made. Between 1970 and 1990, the public interest in automation and AI has decreased, forcing the industry into the so-called AI winter. Despite this recession, research has been continued, and the building blocks for another robot boom during the 1990s have been set. Since then, the usage of robotic applications has broadened, and the industry has proven itself to be a vital aspect of today's economy.

5.2 Robots with human interaction

Nowadays, the utilization of robots in workplaces has broadened to almost every branch and is accepted by employees and workers. In countries like Japan, robots are no longer seen as a threat to jobs. Industrial robots are no longer used as simple construction tools. Instead, their safety and accuracy have improved so that collaborative robots (Cobots) are capable of working in close cooperation with humans. Surgery robots used in the medical sector allow for much more accurate procedures and enable remote specialists to work on patients without having to be in the same hospital. In developed countries, educational robots are used at school or home to teach children topics playfully and interestingly.

[1]Malone, *George Devol: A Life Devoted to Invention, and Robots*

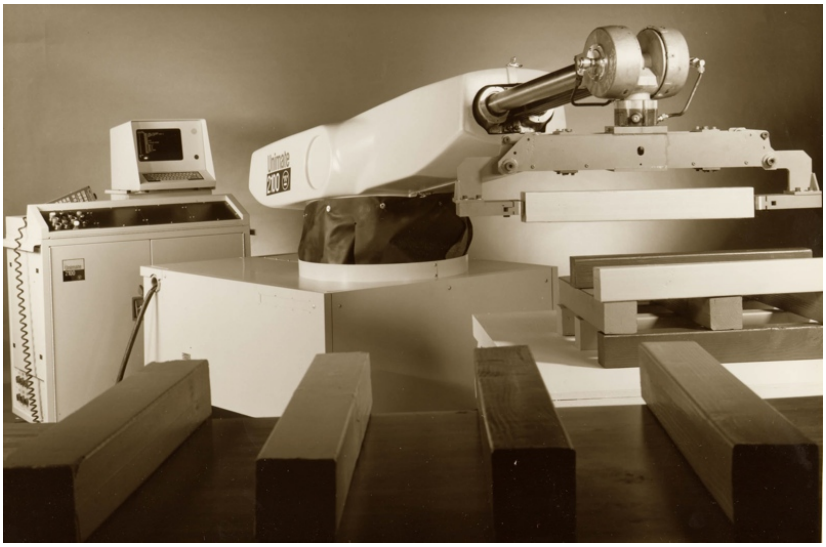


Figure 5.1: Picture of the first industrial robot. The Unimate developed by George Devol and Joseph Engelbert in 1961, was first used for hot die-casting and welding applications.[1]

5.3 Robots in hazardous environments

Robots do not only serve a purpose in a close-to-user work environment, but they also ensure human safety by performing dangerous tasks in unsafe surroundings. Remotely controlled robots or drones can inspect mine shafts, collapsed buildings, pipelines, or overhead power poles. Other applications of such robots are bomb or mine defusion, fire extinction, or avalanche rescue.

5.4 Autonomous robots

Implementing an autonomous robot system is an intricate task that proposes many challenging problems for research or development teams. Autonomy requires a system to continuously work in a dynamic environment without external controlling inputs and utilize perceived information about its surroundings to adapt to environmental change.[2] Despite their complexity in development, autonomous systems, mobile or stationary, immensely facilitate the execution of a job for the human user. Such robots can navigate and organize warehouses, construct parts in an assembly line, or map large areas for comprehensive calculations.

5.5 Autonomous 3D Mapping

While 3D mapping is a well-established and growing economy, most applications require human interaction at some point during the mapping process. Products such as the Emesent

[2]Bekey, *Autonomous robots: from biological inspiration to implementation and control*, Pages 1-2.

Figure 5.2: The Emesent Hovermap mapping a dangerous area inside a mineshaft.[5]

Hovermap[3] or Exyn Aero[4] utilize the spatial flexibility of drones and high-end light detection and ranging (LiDAR) sensors to facilitate autonomous mapping in GPS-denied areas without being within sight of the drone pilot. These solutions are used to autonomously map building or explore hazardous environments such as mineshafts in a human-safe manner, as seen in Figure 5.2.

5.6 Goals

The project associated with this thesis aims to implement a 3D Mapping system similar to the solutions mentioned earlier with limited financial, personnel, and temporal resources. This goal forces the project team to maintain an open-source approach during development primarily.
This thesis aims to document the challenges encountered and experiences gained by the project team during development to demonstrate how highly sophisticated industrial problems can be solved in a low-budget fashion.

5.7 Requirements

In order to declare the project as successful, the following criteria have to be implemented:

- The system is capable of generating a 3D map of its surroundings.
- All required hardware (e.g. sensors, computing boards, etc.) is mounted onto a drone-platform.

[3]Emesent Hovermap, *Autonomy Level 2 for Emesent Hovermap*.
[4]Exyn Aero, *Exyn Aero - Aerial Mapping Drone*.
[5]Emesent, *Emesent_Hovermap.JPG (JPEG Image, 2018 × 910 pixels)*

- The drone utilizes the map to orientate in its surroundings and navigate one or multiple waypoints.
- The path planning is capable of avoiding obstacles as it is moving through an unknown environment.

The following criteria can be implemented but have no direct correlation to the success of the project:

- A user interface facilitating the usage of the system and enabling the user to create. waypoints, monitor the drone and mapping algorithm as well as evaluate the resulting point-cloud
- The system is replicated within the gazebo simulator (see Section 7.8.2) to simplify future development without direct access to its hardware.

Chapter 6

System Architecture

Author: Lukas Leskovar
This chapter provides a thorough overview of Autumns' system architecture by describing its construction and logical and computational structure. To this end, the difficulties faced during development and decisions made that affected the overall project are described.

6.1 First Prototype

The first version of the Autumn Drone consisted of three main components:

- A DJI Matrice 100 functioning as the system's base powering all external components.
- To perform 3D mapping, a Stereolabs ZED 1 was mounted onto the lower front part of the drone.
- As the main component used to compute 3D mapping and control the drone a NVIDIA Jetson TX2 was mounted onto the drones expansion bay.

This prototype quickly demonstrated which part of the system needed improvement, whereas the main issues were the lack of computational power and usage of non-optimal hardware. A detailed explanation and a multitude of solutions to this problem are discussed in the following sections.

6.2 Processing and power management

In the past decades, research has made vast improvements concerning the performance of processors, whether they are used as stand-alone microprocessors, microcontrollers, embedded processors or digital signal processors. However, these improvements come at the cost of higher power requirements. This trade-off is significant in many robotic applications powered by batteries or restricted to low power inputs. Since Autumn is powered only by a drone battery, this issue was groundbreaking during the development of this diploma thesis.

The system's central component is an NVIDIA Jetson TX2 board equipped with a 2GHz NVIDIA Denver2 dual-core and a 2GHz Arm Cortex-A57 quad-core processor.[1]
Using NVIDIA tegrastats[2] the average power consumption of the system at an idle state was measured at 2.7W. However, while performing non-optimized 3D mapping and navigation algorithms (Chapter 10) with both processors fully utilized and Max-N power mode activated, the average power consumption reached 7.9W.[3] Operating the system at such high stress does not only quickly drain the drone's battery but also impairs the quality of the resulting 3D map and operating the drone.

6.3 Solving processing limitations

When dealing with heavy load computations, one way to solve quality issues is to use higher performance hardware. However, this approach is not suitable for this project due to the aforementioned power constraints and drone payload requirements. Another possible solution is to lower the processors' computational load, therefore improving result quality and lowering power consumption. This approach was tested in the following two forms:

- Distributing high power computations to a remote host. This approach lowers the amount of computation on the drone but requires using a high-performance computer on site to communicate with the drone wirelessly. Furthermore, the wifi range becomes another limiting factor since the latency grows with increasing range, thus impairing the resulting quality.
- The second approach was to disable visual odometry using feature extraction and pattern matching as the most complex part of the algorithm and to provide odometry using a much more performant visual-inertial odometry algorithm. To this end, the drone was equipped with a Stereolabs ZED 2i stereo-camera which is benchmarked against other sensors in Section 9.

6.4 Final Product

Following the aforementioned approach the main causes of performance issues where eliminated by replacing non-optimized hardware components and distributing non-critical computations such as user interaction or path planning to a remote host as impairing these aspects of the system by latency would not pose less of a problem. With these adjustments the Autumn Drones final version consisted of the following altered components:

- As already mentioned, the Stereolabs ZED 1 was replaced by its successor, the Stereolabs ZED 2i. With its additional sensors and out of the box sensor fusion available, it significantly reduced the amount of computations performed on the NVIDIA Jetson TX2.

[1] NVIDIA Corporation, *NVIDIA Jetson Hardware Page*.
[2] NVIDIA Corporation, *tegrastats Utility*.
[3] NVIDIA Corporation, *Power Management for Jetson TX2 Series Devices - Supported Modes and Power Efficiency*.

Figure 6.1: The Autumn Drone with the NVIDIA Jetson TX2 on top and the Stereolabs ZED 2i mounted onto the drone's Gimbal mounting plate using a custom 3D printed frame.

- In order to establish a connection to a remote host and perform computations the Dual Band 2.4GHz and 5GHz Antennas of the NVIDIA Jetson TX2 where utilized.
- A laptop serving as the remote host was added to the system performing non-critical computations.

6.5 Autumn Lifecycle

Author: Fabian Kleinrad
This section is focusing on how different modules in the Autumn Project interoperate. Hereby an overview will be given, that will illustrate how the Autumn Project works. The theoretical approach in this project is closely represented by the techniques explained in detail in Section 8.3. In contrast, this section concerns itself with the practical realization of these concepts

6.5.1 Lifecycle Modules

Autumn contains four modules delimited from one another. Like the name suggests these fundamental building blocks of the project are modular. By that means, it is made possible

to change or improve certain parts without the need to integrate the changes into all other aspects present in autumn.

Drone

This component represents the physical drone and sensory equipment installed on the Autumn Drone, described in Section 6.4.

SLAM

The SLAM component converts the sensor data, provided from the drone into a digital representation. Chapter 10 goes into detail how this is realized.

Navigation

The module navigation is used to plan routes through the abstracted environment explained in chapter 11. Hereby the functionality of this module is divided into two components.

- Explorer - decides next waypoint.
- Path-Planning - Plans path to new waypoint, explained in Chapter 12.

Drone Control

Drone control acts as the intermediary between the physical drone and virtual instruction stemming from the navigation component. Drone control can be realized autonomous or semi-autonomous. Semi-autonomous hereby refers to, the user providing flight instructions depended on the navigation output.

6.5.2 Module cooperation

From a process oriented standpoint the Autumn Lifecycle begins with the drone perceiving its environment. The data captured by the sensor equipment the drone is equipped, are stereo images and data from an inertial measurement unit. This data in succession allows an SLAM algorithm to perform its localization and mapping. Upon finishing SLAM provides a digital representation of the space the drone has been able to observe. These abstractions and how such a SLAM algorithm works are covered in Chapter 10 and 11. Using this representation of the environment, an path planning algorithm is used, along with an algorithm determining where the drone should move next. This choosing of the next waypoint is realized in Autumn via the explorer. For Autumn the user provides the point for the explorer. The final step in the navigation component is using the a path planning algorithm, described in Chapter 12, to calculate a path to the goal point. Based on this path drone control is able to send flight instructions using the dji-sdk to control the movements of the drone.

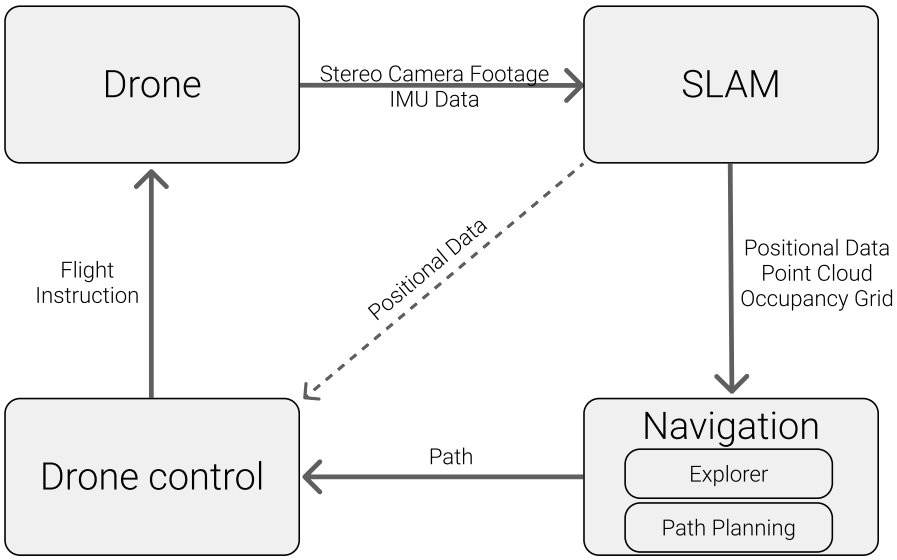


Figure 6.2: Depiction of the Autumn Lifecycle, with its four components and the communication between them.

Chapter 7

Robot Operating System

Author: Lukas Leskovar
This chapter's objective is to describe the basic concepts of the Robot Operating System (ROS) utilised by Autumn. The ROS, despite its name, is a meta-operating system or middleware providing the utility and services often found in robotics frameworks. It enables the composition of distributed systems by utilising publisher-subscriber communication between different programs of such systems. Furthermore, ROS provides a comprehensive set of tools enabling the compilation, operation, testing, visualisation, and debugging of robotic systems. ROS facilitates the development of robotic applications without reimplementing standardised technology with its vast amount of libraries and a huge open-source community providing useful functionality.[1]

7.1 Conceptual Overview

The Robot Operating System can be divided into three conceptual levels, each contributing an integral part to the utility of ROS. These different levels are described in the following sections.[2]

File System

The File System Level mainly provides constraints and best practices for creating and structuring packages and their components. ROS provides appropriate tools to facilitate file-system operations with and within packages.

Computational Graph

The Computational Graph provides crucial functionality to ROS as it refers to the peer-to-peer mesh network of processes (nodes), each providing data to be utilised within the graph by publishing and subscribing to topics. The concepts and technologies powering the computational graph are described later in this chapter.

[1] Gerkey, *Definition - ROS Answers*.
[2] Open Source Robotics Foundation, *Concepts - ROS Wiki*.

Community

The community preserves the usability of ROS as new and useful packages, and tools are created as well as existing functionality is being maintained.

7.2 Naming

To aid the organisation of programs, processes, and resources, ROS provides two naming schemes described in the following sections.[3]

Graph Resource Names

Graph Resource Names utilise a hierarchical structure to organise nodes, services, topics or anything else within the computational graph. ROS defines four different types of names:

Package Resource Names

Package Resource Names aim to facilitate the search process of resources at File System Level. These names usually consist of the package's name and the path to the desired resource within the package.

7.3 Packages

Software in ROS is organised in packages containing nodes, libraries or any other piece of software providing functionality.
Since packages are the atomic unit of build and release, they aim to be as slim as possible by implementing only a limited set of features. In other words, packages should be implemented to provide minimal usability without being too large-scaled.[4] This means that each package is developed to work together with other packages to deliver utility as a connected system.
At file-system level, packages refer to directories. While most subfolders and files within a package depend on their purpose, every package has to contain a package.xml and a CMakeLists.txt providing meta and build information. Packages can be build by utilizing rosbuild or catkin.[5]

Metapackages

Metapackages are specialized packages only containing a package.xml that logically links multiple related packages.[6] They can be used to conveniently install a group of packages simultaneously.

[3] Open Source Robotics Foundation, *Concepts - ROS Wiki*.
[4] Open Source Robotics Foundation, *Packages - ROS Wiki*.
[5] Open Source Robotics Foundation, *Building Packages - ROS Wiki*.
[6] Open Source Robotics Foundation, *Metapackages - ROS Wiki*.

7.4 Nodes

The goal of ROS is to promote code reusability and decoupling of functionality to aid the versatility and usability of the system. Following this guideline, every robotic system utilising ROS consists of a fine-grained graph of processes called nodes. Each node provides computation on a single feature utilising a ROS client library to communicate with others over a mesh-like peer-to-peer network.[7]
Exemplary for such a system would be one node running a LiDAR sensor, one responsible for localisation, one performing motion planning, one controlling motor drivers and motors, and one node running the robot's main control loop.
This architecture allows for much more fault safe and less complex applications in comparison to monolithic systems.[8] This means that development and debugging are facilitated since errors can be contained within a singular slim node rather than a more extensive program.
Each node has a node type consisting of the package name it is located and the nodes executable.

7.5 Communication

7.5.1 Messages

Messages are the medium of communication used in topics or services to transport data between nodes.

Message Description

A message is a simple data structure consisting of multiple type fields. These fields can be primitives, arrays, custom types as well as other message types.[9]
The message description language can be used to structure custom messages in *.msg* files contained in the *msg* directory of a package.

Message Types

Message types refer to package resource names consisting of the package's name as well as the name of the messages *.msg* file.

7.5.2 Topics

The core component of communication in ROS are topics. They are unidirectional message streams enabling data transmission by utilising the publisher-subscriber model Furthermore, the decoupling of functionality is facilitated by anonymously connecting nodes as producers and consumers of data. This means neither publishers nor subscribers of the topic need to know each other. While ROS does not limit the number of publishers and subscribers

[7] Open Source Robotics Foundation, *Nodes - ROS Wiki*.
[8] Stephens, *Beginning Software Engineering*, Page 94.
[9] Open Source Robotics Foundation, *Messages - ROS Wiki*.

connected to a topic, it strictly enforces the usage of the exact message type specified for the topic's communication to work properly.

7.5.3 Services

The communication architecture in ROS utilising the publisher-subscriber model is advantageous in most use-cases, however most distributed systems require remote procedure calls (RPC) which are not supported by default.
With RPCs, a client sends a request to a server specifying the procedure to be called and its parameters. While the server executes the procedure, the client awaits a reply. Once the procedure's results are computed and sent to the client, its workflow can be resumed.[10]
Services enable communication over RPC by defining a pair of messages, one for requests and one for replies. Such service can then be attached to a node and called by a client using the service name.[11]

7.6 Master

One of the most important components of ROS is the Master. It tracks publishers and subscribers of topics and services and provides registration and name resolution to nodes. This means whenever a node wants to publish or subscribe to a specific topic or service, it contacts the Master first using XML-RPC. When a topic has at least one subscriber and publisher, the Master negotiates between the nodes to establish a peer-to-peer connection using a Slave API provided by the nodes XML-RPC Server.[12] A simplified version of this procedure can be seen in Figure 7.1
Besides registration and name resolution, the ROS Master also provides a Parameter Server used for globally storing static system parameters.[13]

7.7 Transform Library

A complex robotic system consists of multiple parts such as sensors, cameras and manipulators, each represented as a coordinate frame, where each frame is connected to another frame using joints. When trying to move a specific part or coordinate frame, the transform of that single frame and the composite transform of each frame in relation to the target have to be calculated. This is especially important when moving a robotic arm based on sensor readings. In this example a transform between the position of the sensor and the arm needs to be calculated so the motion performed by the arm the motion perceived by the sensor match. These complex calculations can be facilitated using the ROS Transform Library (tf). To this end, tf keeps track of each coordinate frame in an acyclic relationship tree where tf broadcasters then publish relative pose information, and listeners query transforms between

[10] Srinivasan, *RPC: Remote Procedure Call Protocol Specification Version 2*, Page 3.
[11] Open Source Robotics Foundation, *Services - ROS Wiki*.
[12] Open Source Robotics Foundation, *Master - ROS Wiki*.
[13] Open Source Robotics Foundation, *Parameter Server - ROS Wiki*.

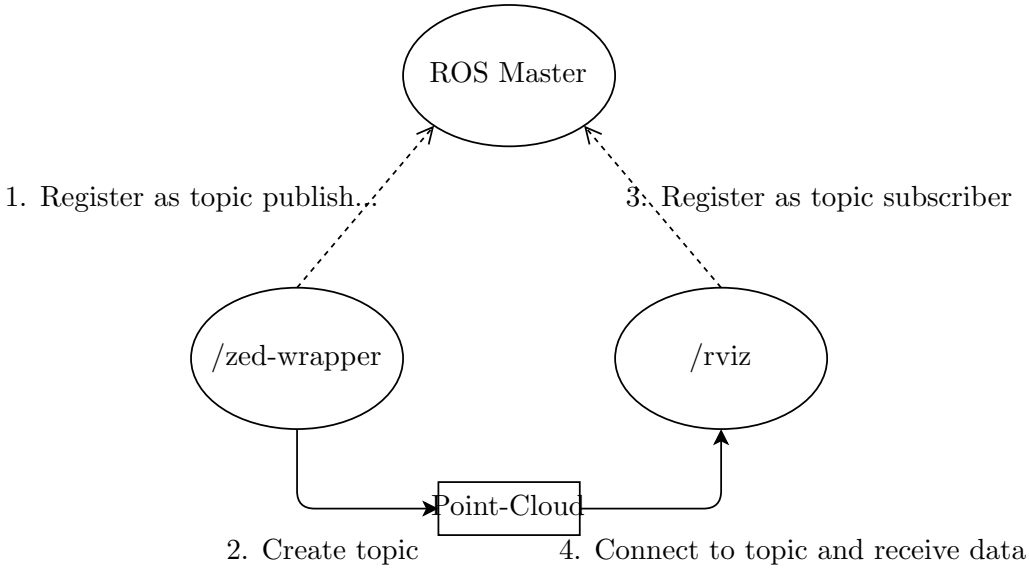


Figure 7.1: Diagram of the topic registration process in ROS at which the publishing node registers its topic before the subscriber tells the master its interest in the point-cloud topic. However a node can be registered as a subscriber to a specific topic without the topic existing yet.

two coordinate frames. Because not all pose information in a robotic system is instantly accessible, the tf saves this information for each frame over time. This means that transforms can be queried not just spatially but also temporally.

7.8 Simulation

Debugging and testing robot applications can be a repetitive and tedious task, especially when a test environment needs to be reset at every test cycle. In order to facilitate this part of development, future implementations of the Autumn Drone ought to utilize the representation and simulation technologies described in the following sections.

7.8.1 URDF

In order to perform simulations or compute coordinate frame transforms, a robot needs to be described in some way. One of the more popular description formats is the Unified Robot Description Format (URDF) which provides an XML format for representing a robot and its components, as well as a *C++* parser and tools to convert and verify and visualise these models.[14] The *check_urdf* tool parses a URDF-File and returns the robot's kinematic chain if successful. To visualize the robots frames and joints in a graphviz[15] tree the *urdf_to_graphiz*

[14]Open Source Robotics Foundation, *URDF- ROS Wiki*.
[15]The Graphviz Authors, *About - Graphviz*.

tool can be used. The resulting tree corresponds to the relationship tree tf uses to calculate transforms.

7.8.2 Gazebo Simulator

Gazebo is a 3D physics simulator often used in close relation to ROS projects. Therefore it provides tooling for model and world design and generation and comprehensive interfaces for controlling a simulated robot through ROS. Further advantages of Gazebo are its accurate sensor and sensor noise generation and its large community providing countless models of robots and sensors.[16]

[16] Open Source Robotics Foundation, *Gazebo - Overview*.

Chapter 8

Autonomous Navigation

Author: Lukas Leskovar Autonomy, or the capability of a robot to perform specific tasks without human supervision or intervention, is a central topic in any robotic application. While stationary systems performing repetitive tasks can be automated relatively effortlessly, mobile robots introduce many challenges that need to be overcome to achieve autonomy. This chapter deals with the classification and explanation of such robots to illustrate the applicability of Autumn as an autonomous navigation system.

8.1 Autonomous Navigation

To navigate a robot through its environment autonomously, it is necessary to develop an algorithm that employs it to move without any external controls. Regardless of how this algorithm is implemented, any autonomous mobile vehicle consists of two fundamental abilities - the ability to perceive its surroundings using one or many sensors and to relocate itself using actuators.

8.1.1 Reactive Approach

One way to autonomously control a robot is to define simple behavioural rules that directly act upon sensory input without needing a sophisticated model of its environment. This approach can be associated with bionic robotics as its concepts are inspired by relatively simple life forms performing intelligent behaviour without having a brain. A control cycle following this approach can be seen in Figure 8.1.

Figure 8.1: A reactive control cycle, matching sensor inputs directly to actions.

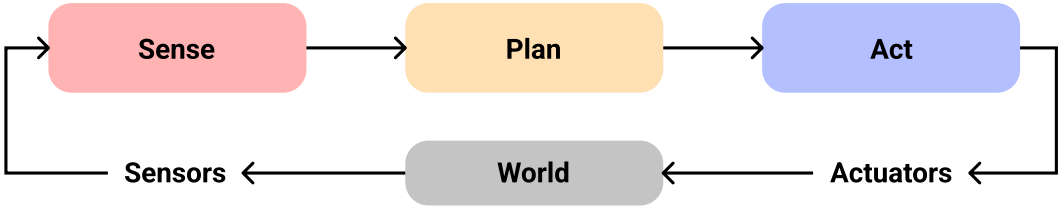


Figure 8.2: A hierarchical control system executing the sense, plan, act cycle sequentially.

These behaviours match sensory input to movement and can be categorized into three levels of complexity:

- Reflexes - pre-programmed direct connections between stimuli and actions.
- Reactions - learned behaviours that execute without the need of complex logic.
- Consciousness - sequences of reactive behaviours ruled by a logical architecture.

Within a reactive system, different behaviours are stacked without any knowledge of one another. This allows for layers to build upon functionality implemented by lower layers while maintaining decoupling of modules, thus promoting task dissection and facilitating independent testing[1].

8.1.2 Hierarchical Approach

Unlike reactive systems, robots representing the hierarchical approach require a much more complex implementation as world modelling and sophisticated reasoning are needed. Any such system, as seen in Figure 8.2, can be divided into three components that are executed sequentially:

- Sense - the robot perceives its environment and creates an abstracted model of it (e.g. create an occupancy grid using a LiDAR sensor)
- Plan - using a model of the environment (e.g. find the shortest path to a given waypoint)
- Act - transform the plan into motion by controlling the robots manipulators

While this approach can introduce many difficulties concerning real-world representation or computational complexity this approach facilitates development of intelligent semi- or fully-autonomous vehicles[2][3].

8.1.3 Hybrid Approach

One way to combine the reactive paradigms simplicity with the intricate prevision of tasks as seen in hierarchical systems can be implemented using a hybrid approach, which is depicted

[1] Jan Faigl, *Robotic Paradigms and Control Architectures*.
[2] Ibid.
[3] Wolfram Burgard, *Introduction to Mobile Robotics - Robot Control Paradigms*.

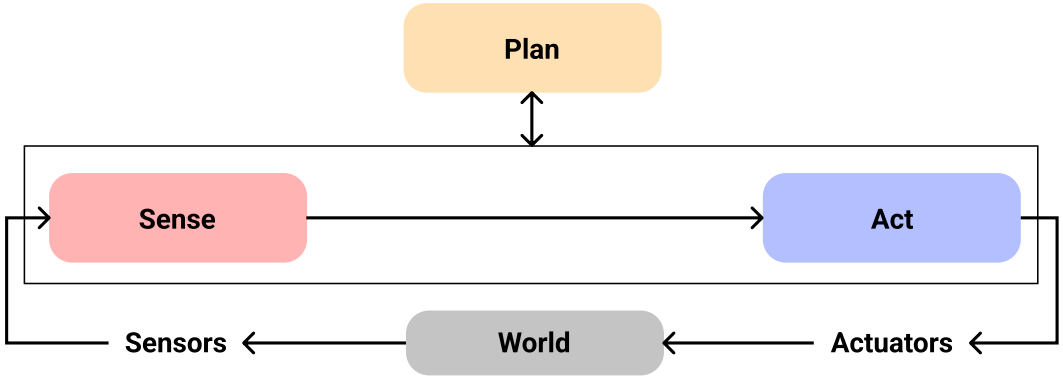


Figure 8.3: A hybrid control system with a planning module supervising a reactive structure.

in Figure 8.3. This concept uses deliberate planning algorithms to determine which reactive behaviour should be executed using a global world model, all while monitoring the success of each behaviour to determine if it is beneficial to achieve a set goal (e.g. find out if the robot moves to a waypoint or is stuck)[4].

8.2 Difficulties

While, in theory, many approaches are applicable to a robot, achieving autonomy can still be a challenging task. Difficulties may arise due to the following reasons:

- Sensing Inaccuracy - Sensors are inherently inaccurate, which can significantly affect how the robot observes its environment and the thereby resulting model. Means of improving sensory measurements usually employ data or sensor fusion methods[5].
- Dynamic Environments - It is relatively easy for a robot to locate itself within a static environment as its pose is the only variable. However, many difficulties may arise when irregularities such as people, movable objects and doors are introduced. These dynamic properties may be treated as noise and thus be filtered[6].
- Predictability of Motion - The actuators robots are equipped with do not exactly perform motion as in any predicted model. This may be due to inaccuracies in actuator fabrication, uneven terrain or changing wind conditions. To counteract such tendencies, motion control paradigms can be applied[7].

8.3 Autonomous Navigation in Autumn

The preceding sections described how a mobile robot can achieve autonomous navigation and which difficulties may hinder the development of such a system. Autumn and its use-

[4] Jan Faigl, *Robotic Paradigms and Control Architectures*.
[5] Siciliano et al., *Springer handbook of robotics*, Page 585.
[6] Thrun, *Probabilistic robotics*, Pages 159 - 162.
[7] Siciliano et al., *Springer handbook of robotics*, Page 133.

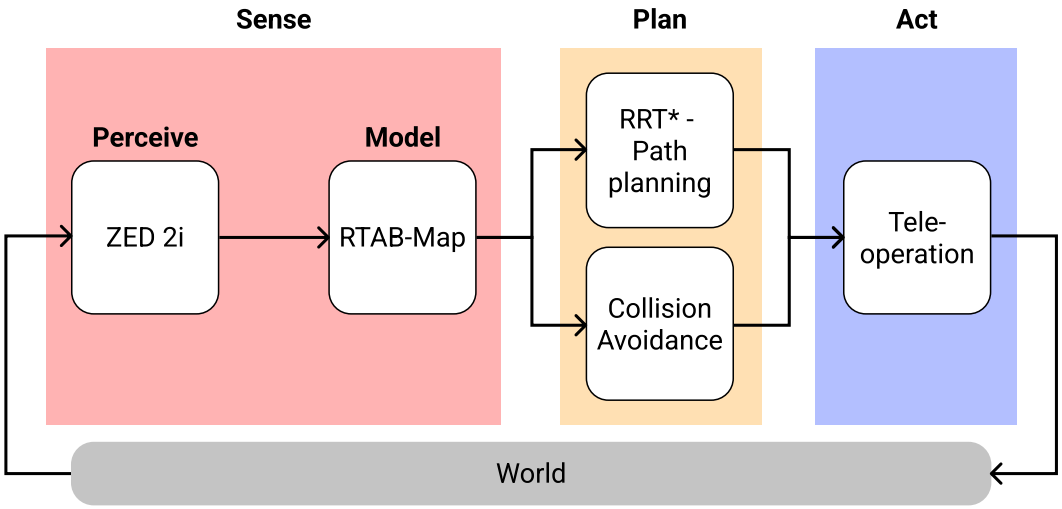


Figure 8.4: This diagram depicts how a hierarchical control system is implemented within Autumn. Using the data provided by a Stereolabs ZED 2i stereo camera, an abstracted model of the drone's environment is created. This model is utilized by path planning and collision avoidance algorithms to propose the best possible path to a given waypoint. Given this path, a drone pilot can navigate the quadrocopter through a hazardous environment.

case do not differ from any such system, which rendered the question of how autonomy is achieved a central concern during the development phase. In order to achieve the best possible product with the available hardware, a semi-autonomous approach was chosen, whose control structure is described in Figure 8.4. The individual interchangeable components the system consists of are described in further chapters.

Chapter 9

Sensors and Measurements

Author: Lukas Leskovar
The means of perceiving ones surrounding environment are a crucial part of any robotic system. This chapter aims to describe the need for interoceptive measurements aiding the different algorithms used within Autumn as well as any sensory equipment supporting such data acquisition. To this end, the underlying principles and concepts behind the measurement techniques are illustrated.

9.1 Motion Models

Kinematic models in robotics are used for describing and planning state-transitions between consecutive robot poses. While Chapter 10 focuses on the theoretic basics of such models and how to build upon them, this section is going to focus on the practical concepts used in robot localization. To this end, classical odometry and velocity-based motion models are described.
To facilitate any equations later in this section, the robot's pose is described by its state vector

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

which defines its position as Cartesian coordinates and bearing as angular orientation θ in 2-dimensional space.

9.1.1 Odometry

Ben-Ari and Mondada define this topic as follows: "Odometry—the measurement of distance—is a fundamental method used by robots for navigation.".[1] When performing linear odometry, e.g. without considering orientation changes, this distance is calculated rather trivially by inferring it through measured time elapsed and the robot's velocity proportionally to the motor power. Other systems utilize wheel encoders to count the number of wheel rotations, thus allowing for a rather precise estimation of distance.

[1] Ben-Ari et al., *Elements of robotics*, Page 69.

With non-linear motion, the distances d_l and d_r moved by the left and right wheel are unequal, which requires the calculation of both updated position and bearing of the robot. Lets suppose a robot moved from position $\begin{pmatrix} x & y & \theta \end{pmatrix}^T$ to $\begin{pmatrix} x' & y' & \theta' \end{pmatrix}^T$ with a slight left turn caused by the right wheel turning faster than the left one. This scenario is depicted in Figure 9.1 with φ corresponding to the turn angle in radians, the radii r_l, r_c, r_r describing the distance from the new position of the wheels and centre to point P, the origin of the turn. For small angles, these radii are approximately the same length as the distances d_l, d_c and d_r, which allows for the turn angle to be calculated as follows:

$$\varphi = \frac{d_i}{r_i}, \qquad i = l, r, c$$

However, in practical scenarios where these radii and the point P are unknown, φ has to be calculated only using d_l, d_r and b, which corresponds to the distance between both robot wheels.

$$\begin{aligned} \varphi r_r &= d_r \\ \varphi r_l &= d_l \\ \varphi r_r - \varphi r_l &= d_r - d_l \\ \varphi &= \frac{d_r - d_l}{r_r - r_l} \\ \varphi &= \frac{d_r - d_l}{b} \end{aligned}$$

To calculate the new coordinate positions the distance d_c is computed:

$$d_c = \frac{d_l + d_r}{2}$$

The final pose is calculated as follows:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} -d_c \sin\varphi \\ d_c \cos\varphi \\ \theta + \varphi \end{pmatrix}$$

Because the assumptions above only apply for small distances, such computations should be performed continuously in any real-world system. However, due to uncertain measurements, these calculations inherit some margin of error integrated up over multiple iterations, thus causing a slight drift in the output.[2] This drift can be corrected using techniques discussed in Chapter 10.

9.1.2 Inertial Measurement

Another way of modelling motion usually utilized when a robot is not equipped with wheel encoders or wheels is based on the vehicle's velocities. Typically the inputs of velocity-based motion models are measured using Inertial Measurement Units (IMU), Gyroscopes and Accelerometers.

[2] Ibid., Pages 69 - 77.

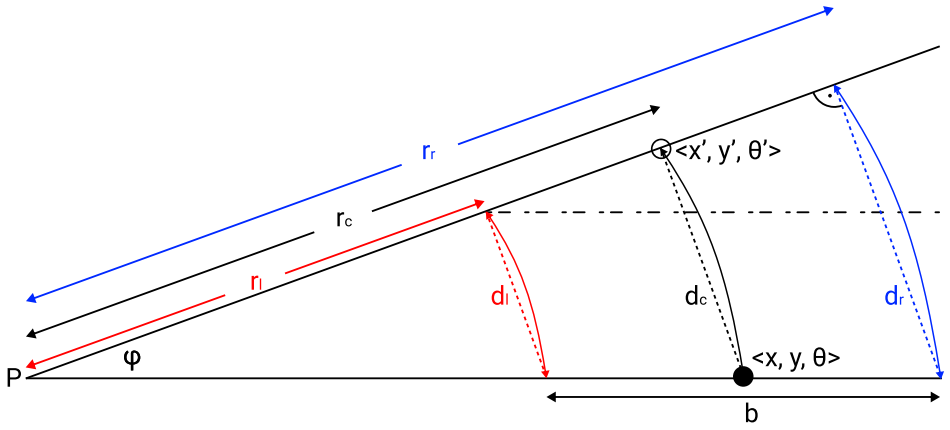


Figure 9.1: This diagram shows the geometry of how a slight turning motion can be calculated only using the distances d_l, d_r and wheel offset b. The robot pose $\begin{pmatrix} x & y & \theta \end{pmatrix}^T$ is associated with the center of the robot[4].

The variables v and ω denote the linear and angular velocities and are referred to as controls for the system. In contrast to odometry, which can only be calculated after a movement has been performed, these controls allow for prior motion planning.[5]
If these velocities stay fixed for the entire duration Δt of the motion from $\begin{pmatrix} x & y & \theta \end{pmatrix}^T$ to $\begin{pmatrix} x' & y' & \theta' \end{pmatrix}^T$ the robot moves in a circle with radius r as seen in Figure 9.2. The center P of this circle with coordinates $\begin{pmatrix} x_P & y_P \end{pmatrix}^T$ can be evaluated as follows:

$$\begin{aligned} x_P &= x - \frac{v}{\omega}\sin\theta \\ y_P &= y + \frac{v}{\omega}\cos\theta \end{aligned} \tag{9.1}$$

This allows for the new position to be calculated:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_P + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ y_P - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{pmatrix} \tag{9.2}$$

Despite being incapable of calculating movements in advance, odometry should be preferred over velocity motion models as they pose higher localization accuracy.[6]

9.2 Depth Sensing

Similar to motion models contributing to robot localization in Autumn, Depth-Sensing is utilized to perceive and map the drone's environment. The technologies in this section focus

[5] Thrun, *Probabilistic robotics*, Pages 92 - 99.
[6] Ibid., Page 107.

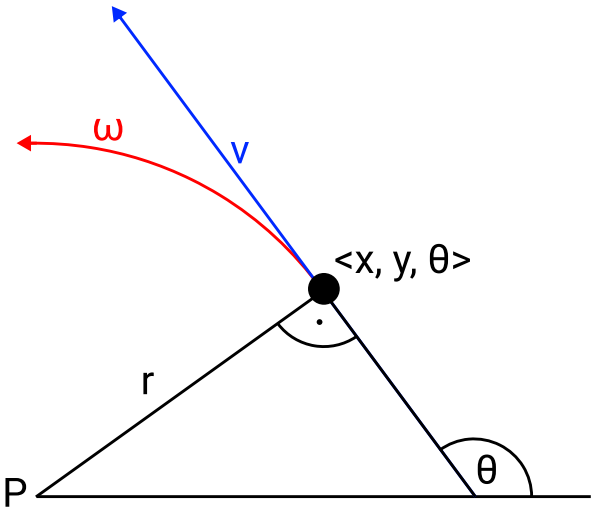


Figure 9.2: In this diagram, a slight turning motion of the robot is described by the translational velocity v and angular velocity ω. These velocities, if remaining unchanged, render the robot to move in a perfect circle with the centre point P and radius $r = |\frac{v}{\omega}|$. Even if the angular velocity remains 0, the motion is still described as a circle with an infinite radius.[8]

on mapping distance information to each point in the camera's field of view, thus providing 3D-Imaging.

9.2.1 Structured Light

One approach uses active illumination to project a varying intensity pattern onto the perceived scene, thus allowing a camera to extract 3D information from the distorted pattern as the scene's surface is non-planar.[9]

9.2.2 Time-of-Flight

Time-of-Flight (ToF) sensors perform active triangulation, which is performed by emitting modulated light to be reflected by a scene, as described in Figure 9.3. The distance to the scene is determined using the time difference between emission and detection of singular rays (more prominent with ToF Laser-Scanners like LiDAR) or the phase shift of light reflected by the whole scene. The latter allows for mapping depth information to the entire Field-of-View (FoV). Although LiDAR sensors fall under the category of ToF sensors, they perceive the distance to an object using pulsed lasers, thus outputting the distance to singular points in the sensor's surroundings rather than depth information mapped to an image.[10][11] As 3D-LiDAR sensors applied in many commercial mapping solutions are very expensive, they disqualify

[9] Geng, "Structured-light 3D surface imaging: a tutorial".
[10] Gokturk et al., "A time-of-flight depth sensor-system description, issues and solutions".
[11] Velodyne Lidar, Inc., *What is Lidar?*

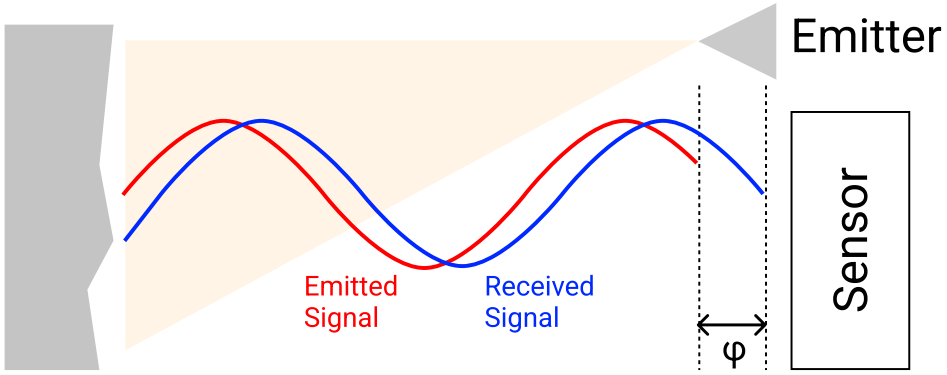


Figure 9.3: This diagram depicts a ToF depth camera emitting IR-light (red) onto a surface. The camera sensor then utilizes the phase-shift φ between the emitted (red) and received signal (blue) to calculate the distance between the scene and sensor.[13]

for use in Autumn. Instead, the project focuses on performing with approximate precision using relatively inexpensive camera-based equipment.

9.2.3 Stereo Vision

The last and most significant approach to Autumn performs passive triangulation using two cameras setup as seen in Figure 9.4. Although this eliminates the need for any light to be emitted, the problem of correspondence is introduced. Which deals with associating the different projections to the same point in the real world[14]. There are multiple approaches to this problem which are generally divided into global and local matching methods, with the latter being more computationally efficient while sacrificing quality.[15]

In Autumn, this approach was pursued using a Stereolabs ZED 2i depth camera as it was superior to any competitors such as the Microsoft Kinect or Intel Realsense concerning availability and cost-effectiveness.

9.2.4 ZED 1 vs ZED 2i

Chapter 6 mentions that the Stereolabs ZED 1 used in early prototypes was quickly replaced with the ZED 2i. This change was due to processing limitations and additional features provided by the newer product version. This section aims to benchmark the two cameras from a depth-performance standpoint to review if the upgrade enhanced depth perception, thus improving the system output or if no changes were necessary when only considering depth-perception.

[14]Ng et al., "Solving the stereo correspondence problem with false matches".
[15]Do et al., "A review of stereo-photogrammetry method for 3-D reconstruction in computer vision".

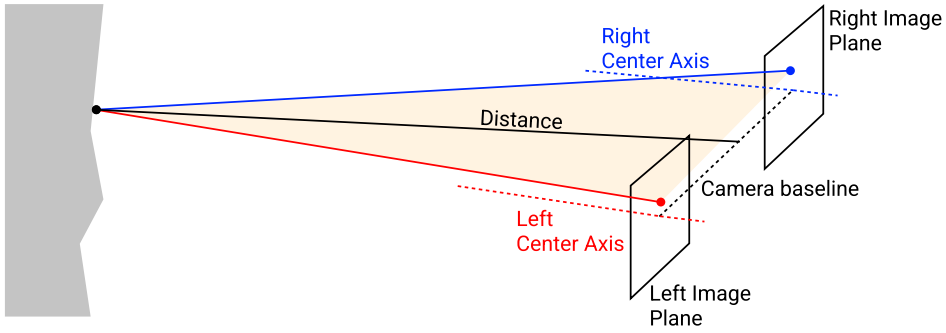


Figure 9.4: This diagram describes the camera setup found in passive triangulation systems. Using stereo-photogrammetry the distance to the perceived scene is calculated.[17]

Experiment

To answer this question the cameras measured the vertical distance enclosed by two points mounted 1 m above each other on a stationary structure, with the bottom one at a height 1 m above the ground. The cameras were placed on a movable rig perpendicular to the stationary structure and 1m above the ground. The cameras measured the aforementioned gap at 1 m increments between 3 m and 10 m horizontal distance measured amongst each structure's base.

The usage of ARUCO tags allowed for precisely tracking the position of each point within the camera coordinate system, thus ensuring the points remain fixed even though the camera rig is moved. ARUCO tags are square-based markers popular in many robotic and augmented reality applications. They provide a fast and robust solution for estimating a camera's pose relative to one or multiple markers. Each marker has a unique, identifiable pattern that allows for precisely determines and tracks its position.[18]

Results - ZED1

Figure 9.5 depicts the mean, minimal and maximal depth-perception error measured, thus indicating how well the camera performed. To illustrate the underlying model of this data, a cubic regression is superimposed. This well-fitted model indicates a relatively steady error just under 0.1 m which increases drastically after a local minimum at a distance of 5.6 m. While the mean error increases, the minimal deviation declines to 0 m between distances of 6 m to 8 m, indicating an optimal interval for measurements. Although the maximal error thrives within this range, it generally remains close to the mean at all other distances. Table 9.1 shows the data these findings are based on as well as a total mean, minimal and maximal deviation over all distances. In total, the ZED 1 has a mean error of 13%, which is drastically above what the cameras data sheet states[19].

[18] Garrido-Jurado et al., "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming".

[19] Stereolabs Inc., *ZED Camera and SDK Overview*.

Distances [m]	3	4	5	6	7	8	9	10	Total
Mean Deviation [m]	0.07	0.08	0.09	0.08	0.09	0.14	0.17	0.31	0.13
Min Deviation [m]	0.07	0.04	0.04	0.00	0.00	0.00	0.01	0.01	0.00
Max Deviation [m]	0.08	0.11	0.13	0.45	0.67	0.18	0.24	0.84	0.84

Table 9.1: Summarized data of mean, minimal and maximal deviation for the ZED 1 camera at increasing distances.

Results - ZED 2i

Data for the mean, minimal and maximal depth-perception error of the ZED 2i is plotted in Figure 9.6. The cubic regression model for this camera indicates a slight negative trend with a coherently low mean error, peaking at a local maximum at 8.6 m. However, the initial belief suggests that with increasing distances, the error thrives correspondingly. According to the coefficient of determination R^2, indicating that the cubic regression function does not fully match the data, a different regression model may result in more accurate assertions supporting the initial belief. Considering the overall error range, the minimal error remains below 0.02 m for all distances while the maximal error grows to approximately 0.4 m. An optimal measurement zone due to moderate mean and non-existent minimal deviation can be observed between 7 m and 8 m. Table 9.2 indicates an overall mean error of 5%, which does correspond to the statement of the ZED 2i data sheet[20].

Distances [m]	3	4	5	6	7	8	9	10	Total
Mean Deviation [m]	0.01	0.01	0.06	0.08	0.05	0.04	0.10	0.08	0.05
Min Deviation [m]	0.00	0.00	0.01	0.02	0.00	0.00	0.02	0.02	0.00
Max Deviation [m]	0.03	0.11	0.26	0.24	0.42	0.35	0.43	0.41	0.43

Table 9.2: The mean, minimal and maximal depth-perception deviation measured by the ZED 2i.

Comparison

Comparing both models, the ZED 1 clearly shows a much more aggressive upwards trend than the ZED 2i, which has a relatively monotonic mean error.
However, both cameras share a commonly low minimal error indicating that both can correctly and precisely take depth measurements within the tested distances. Although the ZED 2i generally outperforms its predecessor concerning mean deviation, its maximal depth-perception error strays drastically further from the mean compared to the ZED 1.
Having said this, in combination with the data sheets, the overall trend on depth-performance meets the initial assumption and justifies the upgrade to the ZED 2i.

[20]Stereolabs Inc., *ZED2 Camera and SDK Overview*.

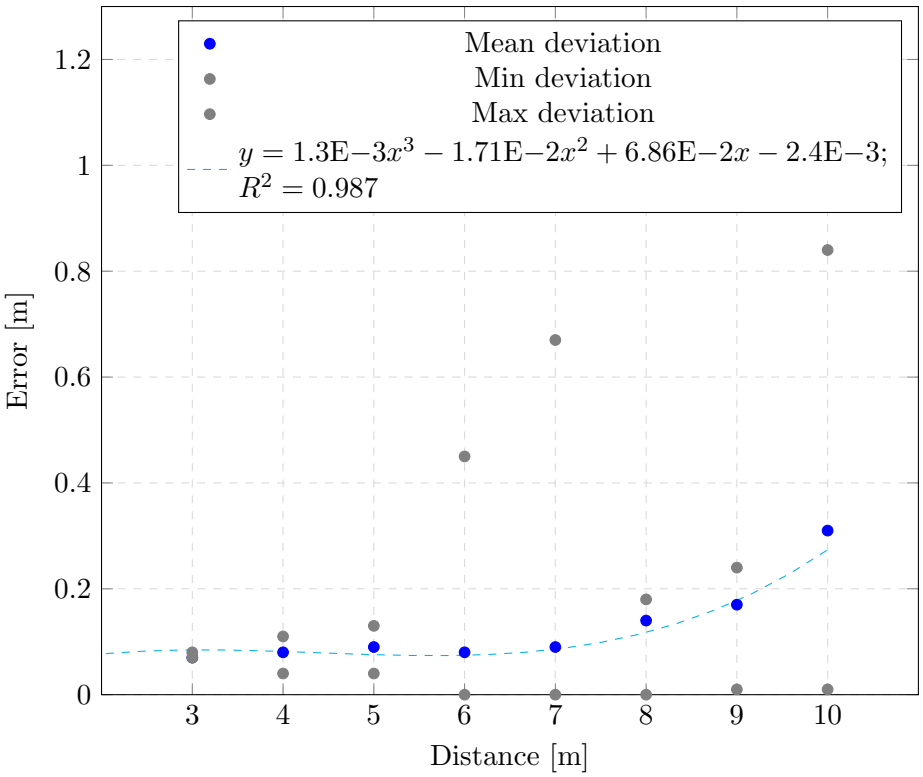


Figure 9.5: This plot describes how the mean (blue), as well as minimal and maximal (grey) depth-performance error of the ZED 1 stereo camera changes with increasing distance to an object. Furthermore the cubic regression function (cyan) for this data is plotted.

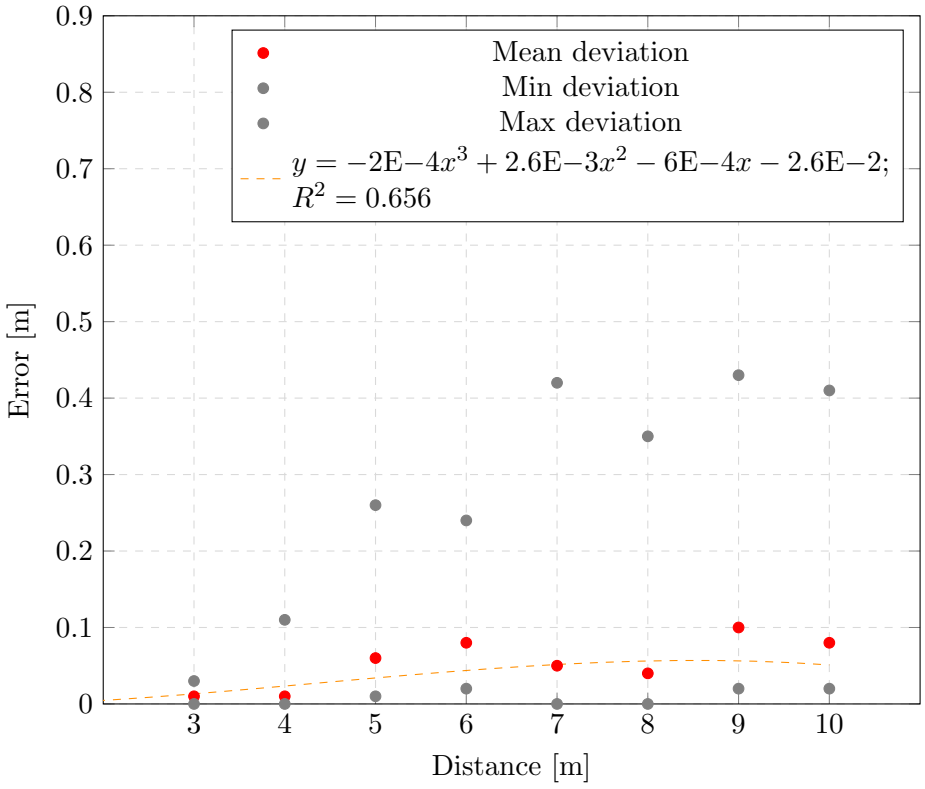


Figure 9.6: This plot describes how increasing distances to a stationary object affect the mean (red), as well as minimal and maximal (grey) depth-performance error of the ZED 2i. The model describing this data is approximated using a cubic regression (orange).

Chapter 10

Simultaneous Localization and Mapping

Author: Lukas Leskovar

The problem of localizing and navigating a system through an entirely or partly unknown environment without any external coordinate system (i.e. GPS, Optical Beacon Tracking) has proven to be one of the most complex and yet fundamental topics in many scientific research fields, with robotics being most prominent. The primary approach to this problem is Simultaneous Localization and Mapping (SLAM), which dates back to the mid-1980s. Back then, the first solutions based on Extended Kalman Filters or Rao-Blackwellised Filters were formulated.[1][2]

To date, the topic of SLAM has matured, algorithms have gotten more reliable and robust and are utilized in many industries. Applications for SLAM range from navigating a Mars-Rover over autonomous cars or warehouse robots to simple household appliances like vacuum cleaners.

As one major topic of this thesis is robot navigation in a GPS-denied area, and mapping of such environment, different modern SLAM approaches and solutions considered by the project team are discussed in this chapter.

10.1 Localization

Localization or state-estimation aims to reconstruct the state of a system using interoceptive measurements (e.g. acceleration, velocity) and an exteroceptive model (e.g. position and orientation) of the system.[3] Put into the context of mobile robotics, this means that to perform comprehensive localization of a mobile robot, a sensor fusion between on-board sensors and a global coordinate system ought to be performed as solely relying on sensor data such as odometry would quickly result in large accumulated errors. An exemplary

[1] Durrant-Whyte et al., "Simultaneous localization and mapping: part I".

[2] Cadena et al., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age".

[3] Barfoot, *State estimation for robotics*, Pages 3 - 5.

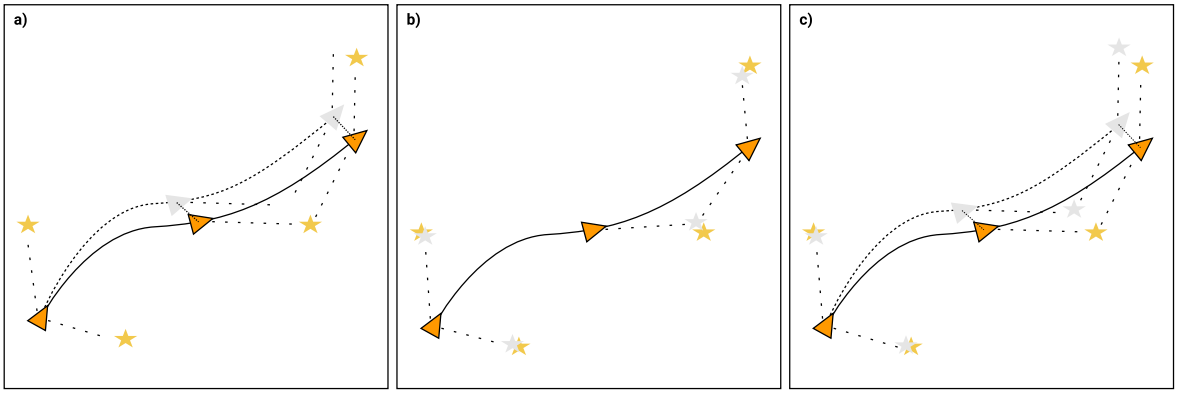


Figure 10.1: The figure above depicts the different tasks contributing to the SLAM problem. Panel a) shows a robot (triangles) moving through a known environment while the pose and path remain uncertain. In panel b) the robot's pose is certain, while the landmark (stars) locations need to be determined. Panel c) composes both previous problems where neither the robot state nor the landmark locations are certain.[5]

system for this would be an industrial robot utilizing a construction plan to correct errors by wheel-encoded odometry to navigate through a factory.

10.2 Mapping

Contrary to localization, in mapping problems, the current pose of the system is known while its environment remains uncharted. Therefore stand-alone mapping aims to generate a model of its environment by evaluating sensor readings and the systems pose to reconstruct environmental features in a global reference frame. Mapping applications often combine technologies typically found in other scientific fields such as photogrammetry or computer vision. An example of such a system would be a drone computing camera images and GPS positional data with a photogrammetric algorithm to reconstruct a 3D-Model of a building.

10.3 Localization and Mapping

On their own, the technologies mentioned above are considered rather simplistic problems as either the environments map is given or a reliable pose-estimation can be provided. However, while most applications meet either of these criteria with pre-built maps or GPS being available, some environments such as buildings, mineshafts or outer space, both the systems pose and environment remain uncertain and need to be determined simultaneously hence the name Simultaneous Localization and Mapping. To summarize, the crux of the SLAM problem, as described in Figure 10.1, is that localizations requires a map, and mapping depends on pose estimates however, neither are certain.

10.4 Map Representation

The way a robot perceives its surroundings and maintains an accurate map is directly dependent on many criteria such as complexity of tasks, size of its environment, and measurement quality mainly influenced by sensor noise. Tailoring to benefit some of the criteria mentioned above, most robotic mapping systems utilize either of two paradigms, metric or topological, each proposing their respective strengths and weaknesses.
Metric or grid-based maps build a map of the robot's environment as occupancy grids, with each cell indicating the presence of an obstacle. The main benefit of using metric maps is the facilitated construction of large-scale mappings and non-ambiguous determination of places. However, such maps have significant drawbacks concerning space and time complexity and require accurate pose estimation of the robot.
Topological or feature-based maps reconstruct their environment as graphs, with each node representing a feature or landmark perceived by the robot's sensors. In contrast to metric maps, they allow for comprehensive path planning and are significantly more compact as their resolution is directly proportional to the environment's complexity.[6]

10.5 Probabilistic Definition

To describe probabilistic SLAM, let's assume a robot is moving through an environment observing multiple landmarks at different times. The goal is that, for any time instant t, the robots state vector

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

as well as a time invariant set of all absolute landmark positions $m = \{m_1, m_2, ..., m_i\}$ is computed, given uncertain relative landmark observations $z_{0:t} = \{z_0, z_1, ..., z_t\}$, controls $u_{0:t} = \{u_0, u_1, ..., u_t\}$ and known initial location x_0. Figure 10.2 depicts this process and illustrates the relationships between the variables important in any SLAM system.

$$P(x_{0:t}, m | z_{0:t}, u_{0:t}, x_0) \tag{10.1}$$

In 10.1, which is often referred to as "Full-SLAM" or "Offline-SLAM", the joint posterior density of the robot's trajectory and landmark locations is estimated. In other words, this formulation of the problem aims to recover the whole robot trajectory.

$$P(x_t, m | z_{0:t}, u_{0:t}, x_0) \tag{10.2}$$

The pendant to this is "Online-SLAM", which aims to estimate only the robot's latest location at time instant t, as seen in 10.2. In contrast to "Full-SLAM", performing batch computation on the whole data, algorithms pursuing this online approach typically compute the probability distribution incrementally.

[6] Thrun, "Learning metric-topological maps for indoor mobile robot navigation".

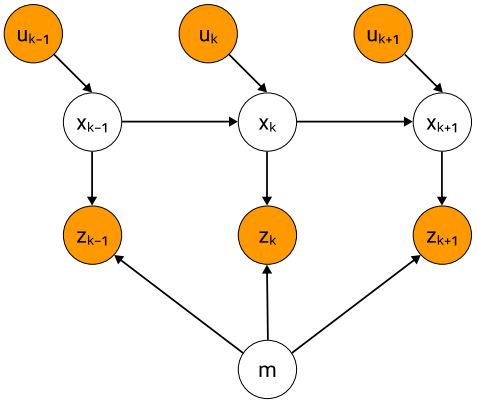


Figure 10.2: A Bayes network graph depicting the causal relationships between sensor measurements (orange circles) and uncertain variables (white circles). It is made apparent how the relative measurements by odometry or landmark observation influence the robot state as well as landmark locations.

To compute either problem a mathematical model representing the robots state transition and a model describing observations ought to be defined:

$$P(x_t|x_{t-1}, u_t) \tag{10.3}$$

$$P(z_t|x_t, m) \tag{10.4}$$

The motion model 10.3 describes the robots location at time t given a known previous location x_{t-1} and control u_t assuming the state-transition is a Markov process[7]. 10.4 calculates the probability of observing a landmark z_t when the robots location x_t and environment m are known.[8]
For filtering approaches which are discussed later in this chapter, the joint posterior is then implemented in a recursive prediction and correction form, which is often described as time-update and measurement-update:

$$P(x_t, m|z_{0:t-1}, u_{0:t}, x_0) = \int P(x_t|x_{t-1}, u_t)P(x_{t-1}, m|z_{0:t-1}, u_{0:t-1}, x_0)dx_{t-1} \tag{10.5}$$

$$P(x_t, m|z_{0:t}, u_{0:t}, x_0) = \frac{P(z_t|x_t, m)P(x_t, m|z_{0:t-1}, u_{0:t}, x_0))}{P(z_t|z_{0:t-1}, u_{0:t}} \tag{10.6}$$

The time update in 10.5 calculates an estimate of the robots state using a previously known state at $t-1$ and state-transition given the newest control u_t. In the next step, the measurement update 10.6, a observation z_t is taken and the previously calculated estimate is corrected using Bayes-Theorem[9].

[7] Haenelt, Karin, *Hidden Markov Models (HMM)*.
[8] Durrant-Whyte et al., "Simultaneous localization and mapping: part I".
[9] Ibid.

To make the the recursive structure of such filtering approaches more apparent the equations 10.5 and 10.6 can be compared to the Bayes filter algorithm, commonly used as framework for further state-estimation problems[10].

$$\overline{\text{bel}}(x_t) = \int P(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1} \tag{10.7}$$

$$bel(x_t) = \eta P(z_t|x_t) \overline{\text{bel}}(x_{t-1}) \tag{10.8}$$

10.6 Solution Paradigms

Over time many different approaches to the SLAM problem have been developed, which can be categorized into either of three paradigms. The two mature ones are based on recursive filtering techniques, while the modern graph-based solutions perform non-linear sparse optimization.

10.6.1 Extended Kalman Filter

The Extended Kalman Filter (EKF) is one of the first approaches to the Online-SLAM problem and an extension of the Kalman Filter (KF). Contrary to the standard KF, it is applicable to non-linear systems by performing linear approximation.
The Kalman Filter assumes a environment in which landmarks can be fully represented as points within the coordinate frame. The earlier introduced state-vector x is extended to a state-space vector with dimensions $3 + 2n$

$$\mu = \begin{pmatrix} x \\ m \end{pmatrix} = \begin{pmatrix} x & y & \theta & m_{1,x} & m_{1,y} & \dots & m_{n,x} & m_{n,y} \end{pmatrix}^T$$

to incorporate n landmark locations.
Furthermore a covariance matrix $\Sigma = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xm} \\ \Sigma_{mx} & \Sigma_{mm} \end{pmatrix}$ is introduced which specifies the state x and landmark m uncertainties as well as correlations between landmarks and robot pose.
The motion and observation models are implemented as non-linear functions $x_t = g(u_t, x_{t-1}) + \epsilon_t$ and $z_t = h(x_t) + \delta_t$ with ϵ_t and δ_t defined as random noise variables.
To still calculate the state transition as well as measurement update these functions are linearized by performing first order Taylor Expansion[11].
This allows for the functions to be incorporated in the vanilla Kalman Filter Algorithm which consists of the following five step procedure:

[10] Thrun, *Probabilistic robotics*, Page 23.
[11] Ibid., Pages 33-51.

Algorithm 1: Pseudo-Code describing the filter cycle of a Extenden Kalman Filter[a]

Function `ExtendedKalmanFilter(` μ_{t-1}, Σ_{t-1}, u_t, z_t`):`
  $\overline{\mu}_t \leftarrow g(u_t, \mu_{t-1})$;
  $\overline{\Sigma}_t \leftarrow G_t \Sigma_{t-1} G_{t-1}^T + Rt$;
  $K_t \leftarrow \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$;
  $\mu_t \leftarrow \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t))$;
  $\overline{\Sigma}_t \leftarrow (I - K_t H_t)\overline{\Sigma}_t$;
  return μ_t, Σ_t;
End Function

[a] Thrun, *Probabilistic robotics*, Page 51.

Notice that the Jacobian matrices G_t and H_t containing all partial derivatives of g and h are utilized describing the transformation of robot state as well as observations based on the pose.

10.6.2 Particle Filter

One major problem with the EKF paradigm is its inability to perform estimations under non-Gaussian distributions. To this end, particle filters estimate a specific probability distribution by proposing multiple hypotheses, which get continuously more accurate as the algorithm proceeds. Particle filtering for SLAM with algorithms like Fast-SLAM is based on Monte-Carlo Localization (MCL) which is roughly described in the following steps:

- For each state x_t a set $\mathcal{X} = \{\langle x_t^{[j]}, \omega^{[j]} \rangle, ...\}_{j=1,...,J}$ of particles or samples with random state hypothesis is generated from a proposal distribution.
- The probability for each particle to be representing the true state is calculated by comparing its estimation with a measurement, thus yielding in each particles importance weight $\omega^{[j]}$.
- The last step is to remove samples with a lower likelihood and add more samples in regions with higher importance weights. This process is called resampling and shares many similarities with survival of the fittest algorithms.

Because the number of particles greatly affects the performance of a particle filter, such approaches perform best in low dimension spaces. To this end, the Fast-SLAM algorithm utilizes Rao-Blackwellization, see Equation 10.9, to separate the joint probability of robot state and map into individual distributions, which can be computed with much more ease. This means that given the robot's poses, all landmarks are independent, which is described in Figure 10.3.

$$P(x_{0:t}, m | z_{0:t}, u_{0:t}) = P(x_{0:t} | z_{0:t}, u_{0:t}) P(m | x_{0:t}, z_{0:t}) \tag{10.9}$$

Following this approach the robots path is estimated by performing a Monte-Carlo localization drawing samples from the robots motion model $x_t^{[j]} \sim P(x_t | x_{t-1}^{[j]}, u_t)$. Besides the state information each particle maintains a set of landmarks, each represented as a low dimensional

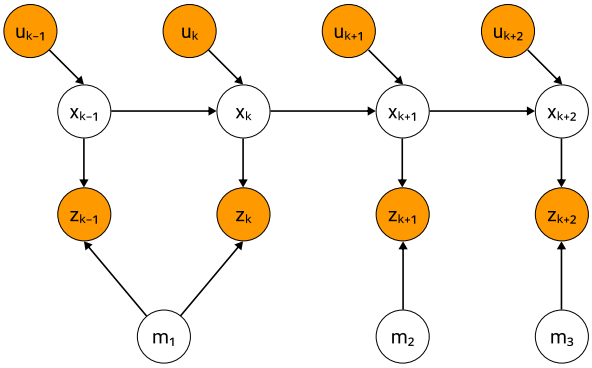


Figure 10.3: This graphical description of the SLAM problem shows the assumption that the landmark estimates are all mediated through the robot path, thus allowing for easier implementation by using smaller Gaussians for each landmark m_i rather than a larger state-space vector as in EKF SLAM.

EKF. Therefore the importance weight for each particle corresponds to the accuracy between expected measurement and landmark observation as $\omega^{[j]} \sim P(z_t|x_t^{[j]}, \overline{z}_t)$. The final algorithm functions exactly as described with the exception that it incorporates the aforementioned EKF to estimate the landmark locations and compute the particle weight before updating its belief on landmark locations and initiating the resampling process[12].

10.6.3 Graph-based

The newest paradigm to solve the offline SLAM problem are graph-based methods that essentially use a non-linear least-squares approach to compute the graph of poses best fitting the measurements. To this end, the SLAM posterior is described as a graph of robot poses x_i at time t_i as nodes connected by soft constraints that correspond to spatial measurements z_{ij} between two states. These edges are generated in the following two cases:

$$(X_i^{-1} X_{i+1}) \tag{10.10}$$

$$(X_i^{-1} X_j) \tag{10.11}$$

Here 10.10 corresponds to an edge based on odometry transformations by moving the robot from state X_i to X_{i+1}, both of which are represented as vectors relative to the coordinate origin. These transformations are illustrated in more detail in Figure 10.4. The more interesting case is when an observation of a previously captured area occurs, and a virtual measurement 10.11 is computed, describing how the states X_j and X_i relate to each other according to the displacement between their observations, see Figure 10.5.
This approach marginalizes any data about the map other than the correspondence of poses to focus on solving the robot path primarily. Given the pose-graph, the goal is to compute the

[12]Stachniss et al., "Simultaneous localization and mapping", Pages 1159-1162.

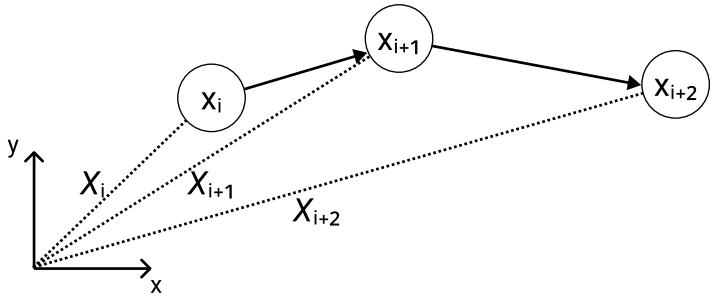


Figure 10.4: The graph of robot poses connected by odometry measurements. To calculate the relative movement between two nodes, each node can be represented as a matrix X_i describing how it is transformed relative to the global coordinate frame.

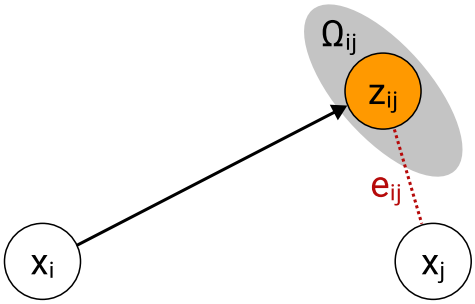


Figure 10.5: This pose-graph with two nodes describes how any pair of nodes are related to each other given a measurement. The measurement z_{ij} corresponds to how the node x_j should be placed relative to node x_i given that each of them share a similar observation. As seen in the figure, this is not the case, thus demanding for the error e_{ij} to be minimized. In other words, by moving the node x_j closer to the optimum minimizes the error and corrects the pose-graph.[15]

node configuration that minimizes the error between predicted edges, see 10.11, and actual measurements $\langle z_{ij}, \Omega_{ij} \rangle$, with the information matrix Ω_{ij} describing the measurement noise. Equation 10.12 describes this optimization process, with the error function e_{ij} defined as any non-linear function essentially calculating the difference between the estimate and true measurement for any pair of nodes. Once the optimal pose-graph is constructed, the map data is rendered given known poses, thus reducing the problem to simple mapping[13].

$$x* = argmin_x \sum_{ij} e_{ij}^T \Omega_{ij} e_{ij} \tag{10.12}$$

10.7 Graph-SLAM topology

Most modern graph-based SLAM systems are partitioned into the following modules:

[13] Grisetti, Kümmerle, et al., "A tutorial on graph-based SLAM".

- A front-end in charge of abstracting sensory input by performing feature extraction as well as data association, further described in Section 10.8. Essentially it constructs the pose-graph and feeds it into the back-end.

- A back-end performing pose-graph optimization based on the abstracted data fed in by the front-end. The back-end also provides positional information about nodes to the front-end thus supporting loop closure detection.

While the front-end remains application-specific, as the data abstraction depends on the sensors utilized by the robot, the same back-end can be applied for many different scenarios. As the pose-graph optimization essentially represents a squared-error minimization, many elaborated algorithms such as Gauss-Newton can be utilized.

10.8 Data Association

One key trait that all SLAM paradigms possess is to recognise similarities in measurements and correct the incremental odometry error after revisiting previously mapped areas. This process is referred to as loop-closing or long-term data-association and reduces the uncertainty of pose and map estimates, thus qualifying it as an integral part of any SLAM problem.
While loop-closures aim to detect large correspondences in the global map, short-term data-association is responsible for associating corresponding features over multiple measurements, which helps establish the pose-graph.

10.9 Hierarchical Pose-Graphs

While building the pose-graph can be computed with relatively low effort, performing loop-closures is a computationally heavy process. This makes graph-based online SLAM solutions challenging to implement as robot state and map need to be estimated in real-time scenarios. To facilitate the data association process, the global map is partitioned into sub-maps, thus allowing data-associations to be calculated only in parts of the graph the robot is currently present in.

10.10 Comparison of SLAM Paradigms

The Extended Kalman Filter is a mature approach to the SLAM problem and has been implemented many times in different applications. However, it does not only lack computational efficiency in its original implementation, see. Table 10.1, but also has shown to be very problematic concerning consistency and robust data-association. These problems stem from the algorithm definition itself. Firstly, local-linearization of non-linear problems impairs the system to reach complete convergence, e.g. all landmark uncertainties converging to zero. Secondly, the data-association model of EKF assumes all landmarks to be fully identifiable from any point of view, which cannot be guaranteed in most real-life scenarios and renders

Approach	Problem	Time Complexity
EKF	online SLAM	$O(M^2)$
Fast-SLAM	full & online SLAM	$O(JM)$
Graph SLAM	full SLAM	$O(\|V\| + \|N\|)$

Table 10.1: The three main SLAM solution paradigms in basic implementation compared by their computational complexity. Note that the quadratic time complexity of the EKF approach is caused by quadratically extending the covariance matrix as landmarks M are added and updated throughout mapping[22]. For Fast-SLAM, the complexity concerning just the particles J stays linear. However during every resampling step, each particle, including all landmark estimates M contained within, need to be copied[23]. Lastly, the back-end for most graph SLAM solutions is based on least-squares error minimization of the pose graph with $|N|$ nodes as robot poses and $|V|$ vertices as measurements between nodes, thus rendering the computational complexity to be $O(|V| + |N|)$ without utilized hierarchical pose-graphs as studied by Korovko and Robustov[24].

data-association very fragile. Fortunately, the EKF has been improved and optimized to address some of the aforementioned issues[16].
Another paradigm proposed was Particle Filtering, which essentially applies a survival of the fittest model to robot poses. Solutions like Fast-SLAM optimize the MCL algorithm by decoupling landmarks and estimating them through individual EKFs. The main benefit of this solution is the utilization of a non-linear model and the ability to perform even under non-Gaussian distributions. Although it is computationally more efficient compared to the previous approach, one problem is the need for large amounts of particles to achieve high accuracy. Montemerlo suggests implementing a balanced binary tree to store and organize particles which improves the complexity of the algorithm to $O(J \log M)$ with J representing the number of particles and M being the number of landmarks[17].
Lastly mentioned was the graph-based approach to SLAM which has grown significant popularity due to its versatility and reusability in different applications. The robot posterior is modelled as a sparse pose-graph, optimized using a least-squares algorithm and is solved offline after the whole robot path has been captured. Some newer algorithms try to manipulate the pose-graph in real-time to establish a graph-based online SLAM solution[18].

10.11 SLAM in Autumn

As localization and mapping data is utilized to perform real-time path-planning and drone control, these modules are a crucial part of the Autumn Drone, thus requiring an algorithm that efficiently solves the online SLAM problem while maintaining compatibility to the hardware in use. This restraints the solution in question to perform visual SLAM supporting

[16]Bailey et al., "Simultaneous localization and mapping (SLAM): Part II".
[17]Montemerlo et al., "FastSLAM: A factored solution to the simultaneous localization and mapping problem".
[18]Stachniss et al., "Simultaneous localization and mapping".

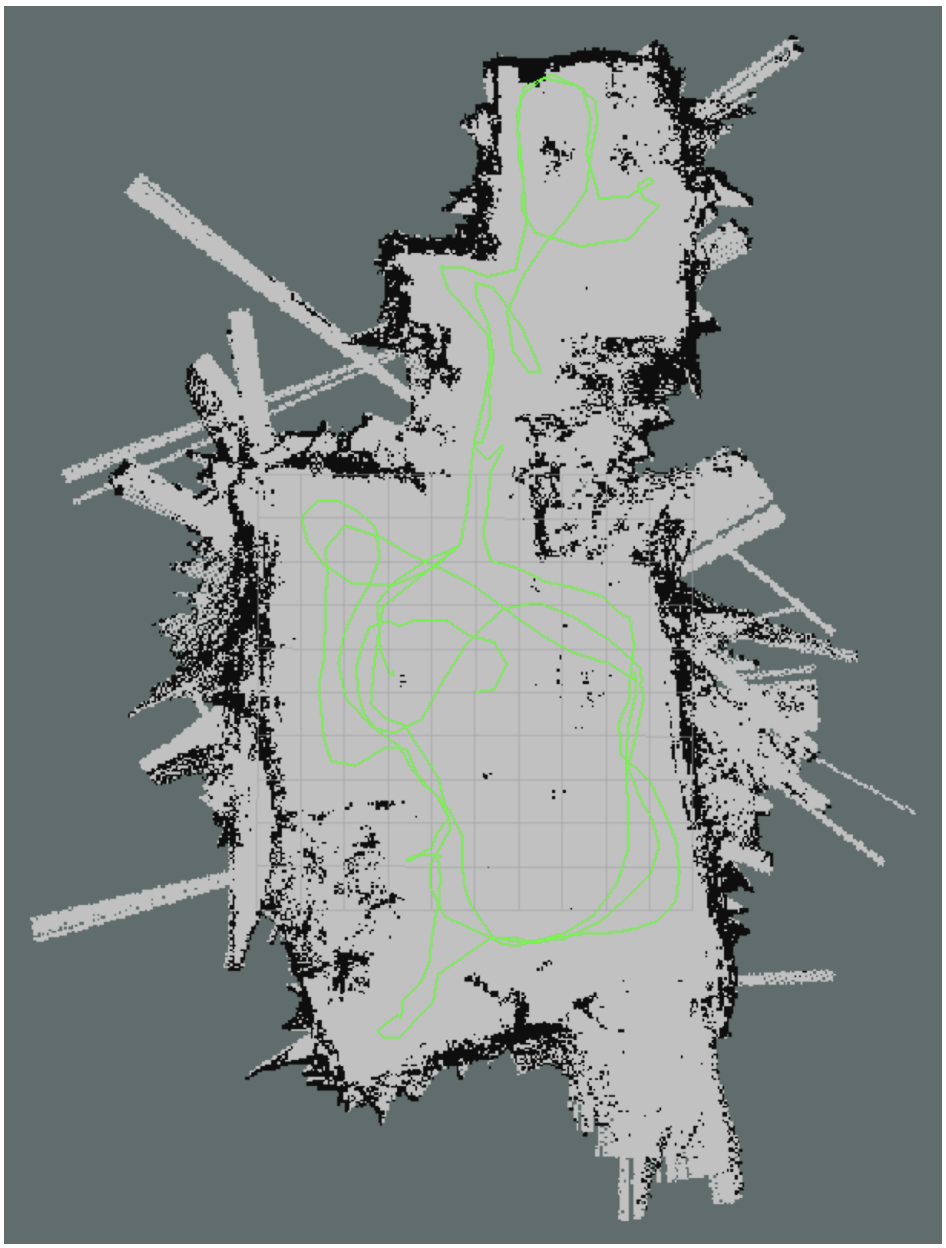


Figure 10.6: Occupancy grid of the AIRLab at the HTBLuVA Wiener Neustadt, including the path pursued by the drone as it got carried through the laboratory. The map, as seen above, is of low quality, including many artefacts, sensor noise, and misaligned proportions. Although the main room is poorly mapped with many errors, the smaller workshop at the top is depicted relatively accurate.

stereo images and 6DoF-Odometry. While remaining within the package-space of ROS, this reduces the number of potential algorithms to the following graph-based SLAM solutions:

- ORB-SLAM2
- S-PTAM
- RTAB-Map
- RGBDSLAMv2

ORB-SLAM2 and S-PTAM were not selected as both approaches do not provide online occupancy grids as ROS-Topics, making them infeasible to use and incorporate with the path-planning module of Autumn. Furthermore, they do not propose a memory-management strategy, thus potentially making loop-closure processing within large maps cumbersome.
The algorithm chosen for Autumn was RTAB-Map as incorporating stereo-images, and external odometry is supported out-of-the-box. Furthermore a variety of outputs such as 2D and 3D occupancy grids and point clouds are offered. As loop-closure detection algorithm with memory management was the primary goal of RTAB-Map, it qualifies for usage in large-scale environments during long mapping sessions. Since its publication in 2013, this algorithm has been extended and improved to support stereo- or RGBD-footage, 2D and 3D Lidar data and provides nodes for processing visual or lidar-based odometry if no other external topic is available. For Autumn, this approach was implemented by providing the stereo-image stream as well as visual odometry data from the Stereolabs ZED 2i, thus yielding in multiple occupancy grids and robot paths as visualized in Figure 10.6.
Although RGBDSLAMv2 does not fully qualify to be considered in this project as it only works with RGBD-Cameras like the XBOX Kinect or Intel RealSense, it is worth mentioning as it provides a 3D occupancy grid and a dense point-cloud similar to RTAB-Map.[25]

10.12 Topics not mentioned in this Chapter

The SLAM problem is a highly diversified field of study with many different concepts not only concerning one paradigms adaption to other use cases than originally mentioned but also diving into the exact implementations of any of its components. Due to this abundance of concepts and algorithms, it would be infeasible to characterize every one of them. However, for the intrigued reader, the author suggests the following topics and resources for further reading:

- Particle Filtering for grid-based SLAM[26].
- Visual SLAM approaches. A wide range of literature on this topic is provided by the Technical University in Munich (TUM).

[25] Labbé et al., "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation".
[26] Grisetti, Stachniss, et al., "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters By Adaptive Proposals and Selective Resampling".

- Graph-SLAM frontends using different data association approaches for visual SLAM such as feature-based matching like RANSAC[27] or point-to-point matching like ICP[28].
- Mapping in dynamic environments capable of comprehending moving obstacles within the environment[29].
- SLAM with multiple robots[30].

[27] Jia et al., "An Improved RANSAC Algorithm for Simultaneous Localization and Mapping".
[28] Horn et al., "Closed-Form Solution of Absolute Orientation using Orthonormal Matrices".
[29] Wang et al., "Online simultaneous localization and mapping with detection and tracking of moving objects: theory and results from a ground vehicle in crowded urban areas".
[30] Gutmann et al., "Incremental Mapping of Large Cyclic Environments".

Chapter 11

Representation of Environments

Author: Fabian Kleinrad
This chapter focuses on the critical intersection of physical and digital space. Fundamental for autonomous navigation are means for the algorithm to observe its surroundings. In order to accomplish a translation from physical space to a medium tailored towards information propagation, a SLAM algorithm, which is described in detail in chapter 10, is being used. The following sections are going to explore the data, which is the output of the mapping phase and input into the path planning phase.

11.1 Abstract Environments

Abstraction being the process of reducing an Object to the information needed for further processing. In conjunction with autonomous navigation, this principle is ubiquitous. Due to computational space and time restrictions, a simplification of complex compounds is needed. Such a loss of information naturally happens when using equivalent hardware to process physical environments.
For autonomous navigation to work, it is necessary for the computer to know the existence and position of objects surrounding the navigated vessel. This information can be provided or self-taught in case of using a SLAM algorithm. SLAM reduces the environment to positions in a two or 3-dimensional space. The algorithm is then left with an assortment of positions relative to the starting point, represented by the coordinate origin. To make use of this information, it is imperative to know the robot's position in this abstracted space.
This approach of letting the algorithm perceive the environment guarantees an efficient and cost-effective computation. Moreover, by reducing obstacles to only the properties needed, unnecessary complexity is circumvented. The Kalman Filter described in section 10.6.1, reflects this principle.

11.2 Representation in Autumn

To utilize the information provided, there needs to be the capability of bundling data into a tried and tested format that enables easy access and reliable accuracy.

Autumn uses the occupancy grid and point cloud to represent the data acquired in the mapping stage. The occupancy grid serves its purpose in a 2-dimensional environment, whereas on the contrary, the point cloud is utilized in 3-dimensional use cases.

11.2.1 Occupancy Grid

An occupancy grid is used for a 2-dimensional representation of the environment. In the case of an occupancy grid, a grid structure is being overlaid over the environment. Therefore it is possible to discretize the world into cells. Each cell contains information about the probability of being occupied. In this context, the assumption is made that cells can have either one of the two states. Thus, it simplifies the algorithms needed to update such data structures because this assumption allows for the use of binary random variables. Grid cells that are for certain occupied are represented by 1, which relates to the 100 per cent probability of being occupied. Contrary free cells are described by a 0 per cent probability of being occupied. Visually these features are displayed by colouring the cells black and white and every probability in between.[1]

Figure 11.1: Visual depiction of probability values in an occupancy grid map.

Two additional assumptions are made to add to the simplicity of the data structure and lower the time needed to update. First, the occupancy grid and many other data structures containing real-world information assume a static world, which entails that occupied cells

[1] Cyrill Stachniss, *Occupancy grid maps*.

and unoccupied cells will not change their value of occupation. Secondly, cells are viewed as individuals with no influence on their neighbouring cells. As a result, the calculation of probabilities is simplified. The probability is, in this case, the product of the probability of single sensor readings.[2]

11.2.2 Voxel Grids

When familiar with the concept of an occupancy grid, it would appear that adding a dimension would solve the problem of 3-dimensional representation. Using an occupancy grid in such a use case would be called a voxel grid. The environment is not subdivided into two-dimensional planes but cubes with voxel grids. The basic principle of the occupancy grid stays the same. However, adding this additional dimension accentuates a drawback that accompanies the concept of occupancy grids. When working with occupancy grids, the space needed to store this kind of structure grows linearly with the number of cells. Using it in a 3-dimensional context makes larger scans hard to maintain and inefficient to work with, particularly in a real-time navigation scenario.[3]

11.2.3 Point Cloud

The most widespread data structure for storing 3-dimensional environments is the point cloud. In contrast to its adversaries, it stores only information that is known. The drawback of the voxel grid is thereby minimized. Point Clouds store points in three-dimensional space. These points represent an estimation of a point on a surface of an object, derived from sensor readings and the position these readings took place. Therefore, a Point Cloud is an accumulation of various points that represent the boundaries in which collision-free navigation can occur. Additionally, the positional information stored in these points can also contain colour and luminescence values.[4]
Discarding a strict grid structure, which decreases the spacial requirements, also increases the complexity of working with this kind of data structure. In robotics, almost always are point clouds used in conjunction with the point cloud library.

PCL

The Point Cloud Library simplifies working with a point cloud data structure. PCL is a free library supporting point clouds of any dimension. It is written for optimal performance and offers a variety of 3D processing functionality. These include filtering, segmentation, feature estimation. Furthermore, PCL is fully integrated into ROS, which means specific functionalities are packaged into nodelets that can be added to fit the underlying use case.[5]

[2] Cyrill Stachniss, *Occupancy grid maps*.
[3] Ibid.
[4] TECH27 SYSTEMS, *WHAT ARE POINT CLOUDS?*.
[5] Conselho Nacional de Desenvolvimento Cientifico e Tecnologico., "O CNPq e a formacao de recursos humanos de C&T para o Brasil : estatisticas de bolsas no pais e no exterior, 1980-95".

11.3 Abstracted Environments in Autumn

Autumn contains two different path planning packages catering to different dimensional environments. When using a drone, three-dimensional path planning is necessary to benefit from the usage of an aerial vehicle. Therefore the "autumn_pathfinding" is used, based on a point cloud. Additionally, the package "autumn_pathfinding_2D" uses an occupancy grid and thus is appropriate for situations where the third dimension cannot be utilized.
Through the use of ROS, explained in detail in Chapter 7, this modular approach is made possible. Furthermore, the Point Cloud and Occupancy Grid data types are integrated into ROS and universally applicable.

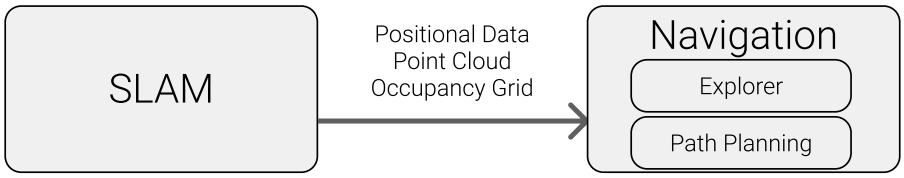


Figure 11.2: Excerpt of the Autumn-Life-Cycle. Lifespan of point cloud and occupancy grid. Created in the SLAM stage, published as a topic and used in the navigation stage.

11.3.1 autumn_pathfinding_2D

The package "autumn_pathfinding_2D" subscribes to a topic published from the SLAM Node, which carries a nav_msgs/OccupancyGrid. This is a standardized data type for occupancy maps. It consists of a header of type std_msgs/Header, containing a sequence number, timestamp, and frame id used to identify individual messages. Additionally, the map's metadata and the current data contained in an int8 array are transferred over this topic.[6][7]
Upon receiving a new map and goal point, given that current calculations are ongoing, the algorithm aligns a Cartesian coordinate system relative to the drone's current position. From this point onward, access to the occupancy grid represented by the int8 array happens through a function that takes x and y coordinates and maps it to the one-dimensional array. This is essential to guarantee usability alongside performance.
The coordinates get also adjusted based on the occupancy grid resolution. Benefits gained from such a procedure are that it is easy to check duplicate cells because only whole cells can be represented. Another reason for it is the use of the cantor pairing function[8] which only works with natural numbers. The reason for using such a pairing function is to store and search for points efficiently. The underlying path planning algorithm explained in detail in chapter 12 is not affected by the input data structure. The data array, holding occupancy information, contains per index a value between 0 and 100 or, if the cell is unknown, the value -1. The path planning algorithm works in a way that unknown cells with value -1 are

[6] Open Source Robotics Foundation, *nav_msgs/OccupancyGrid Message*.
[7] Open Source Robotics Foundation, *std_msgs/Header Message*.
[8] Szudzik, "The Rosenberg-Strong Pairing Function".

taken as free cells. This "free until proven occupied" mentality allows for more efficient space exploration.

11.3.2 autumn_pathfinding

The three dimensional equivalent to the two-dimensional path planning packages uses a point cloud in order to navigate the environment. This point cloud is published as a sensor_msgs/PointCloud message type. This universal point cloud type contains a header with message-specific information as well as an array of geometry_msgs/Point32, which represent the individual points of the point cloud. In addition, Point32 contains x, y and z attributes representing the Cartesian coordinates.[9][10]
The point cloud library will be used because of the difficulty of working with a three-dimensional space represented by a one-dimensional array. Same as the two-dimensional model, the algorithm of the three-dimensional path planning accesses the point cloud data through a function that takes three coordinates and returns if an object exists at this position.
Like the 2d solution, point coordinates get adjusted to a specific resolution. However, in this case, it is not bound to the data type itself because of the lack of structure in a point cloud. Instead, the point resolution is chosen to guarantee the best performance while retaining a high level of flexibility. This process of adjusting the resolution can be understood as altering the size of cubes covering the space, like cells in an occupancy grid. This way, a structure is added to the otherwise unstructured set of points, which allows for the exact implementation of the algorithm as in the two-dimensional solution.

[9] Open Source Robotics Foundation, *geometry_msgs/Point32*.
[10] Open Source Robotics Foundation, *sensor_msgs/PointCloud*.

Chapter 12

Path Planning

Author: Fabian Kleinrad
A crucial part of autonomy in robotics are the means for planning movements in a cooperative manner with the environment. Methods to accomplish this are generally referred to as path planning algorithms. This chapter will cover different kinds of approaches to path planning and evaluate which approach is most fitting to be used in a real-time, high-dimensional use case present in the Autumn Project.

12.1 Types of Path Planning

The problem of finding an optimal path between two points is an old one. The first proposed solution was Dijkstra's algorithm. However, with steadily evolving computer science, the challenges to be mastered by such algorithms got harder and harder. That is the reason why over the last years, the simple principle of the Dijkstra algorithm has branched out, specializing and excelling in specific real-world applications.[1]

12.1.1 Sampling-based Algoritms

In motion planning, sampling-based algorithms can be differentiated from other kinds of approaches by the way they explore their environment. For example, sampling-based algorithms such as the probabilistic road map algorithm or the rapidly exploring random tree use a random point in their reference space and expand in that direction. This random point is considered a sample.

12.1.2 Multiple-Query and Single-Query

The term multiple-query refers, in connection with path planning Algorithms, to the feasibility of deriving a variety of different paths, without the need of rerunning the algorithm. In contrast, single-query algorithms are only able to compute one path at a time.
Use cases for Multiple-Query Algorithms would be unchanging environments. The reason for that, by generating an extensive grid of connections to be able to calculate a multitude of

[1] Pan et al., "A Discussion on the Evolution of the Pathfinding Algorithms".

different start/goal combinations, more computational time is needed.
Single-Query approaches focus on performance instead of reuse-ability, making them ideal for dynamic domains.[2][3]

12.2 PRM

Probabilistic RoadMap is a path planning algorithm tailored to multi-query applications. It is considered one of the most influential sampling-based path planning algorithms.
The algorithm can be broken down into two phases. The first phase, referred to as the pre-processing phase, starts with an empty graph. At first, it samples n random points and adds them to a set of vertices if they are located in space free from obstacles. After constructing a set of n vertices, it attempts connections between a random vertex and its neighbouring nodes in a predefined radius. This connecting of to vertices is realized with a simple straight-line connection. Finally, all collision-free connections between vertices and their respective neighbours are added to a set of edges. This pre-processing phase is a roadmap, with the number of sampled points determining the quality of to be calculated paths.[4]
Upon finishing the initial construction phase of a roadmap, like the one depicted in figure 12.1, start/goal combinations can be processed. The actual pathfinding in the generated graph is handled by other non-sampling-based pathfinding methods such as A*.
With the PRM focusing on a multi-query approach, it is possible to calculate an arbitrary number of different paths without the need to construct a new graph.

12.3 RRT Algorithm

Rapidly exploring random tree is a path planning algorithm that can be categorized as sampling-based and single-query. In the field of sampling-based path-finding algorithms, it is considered the most influential with the PRM. RRT works by constructing a tree of possible trajectories. Therefore it is first required to define an initial vertex, much like a root. After each following iteration, a random sample is taken from the space considered free from obstacles. After generating a random vertex, the nearest neighbouring point in the tree is searched for. The closest node is then used as a pivot point, and a new vertex is constructed a predefined distance away from the nearest node in the direction of the random sample. Thereafter it is attempted to connect the new vertex with the nearest. If this straight-line connection can exit without colliding with obstacles, it is added to the set of edges. This procedure is depicted in Figure 12.2. The algorithm ends when a new node is within the predefined distance from the goal point.[5]
The name rapidly exploring random tree stems from the tree-like structure constructed, like the one depicted in Figure 12.2 when exploring spaces. This approach to path planning makes it possible to explore rapidly changing environments efficiently.

[2] Bekris et al., "Multiple Query Probabilistic Roadmap Planning using Single Query Planning Primitives".
[3] Adwait Naik, *Difference between Single-Query and Multiple Query Algorithms?*
[4] Karaman et al., "Incremental sampling-based algorithms for optimal motion planning".
[5] Ibid.

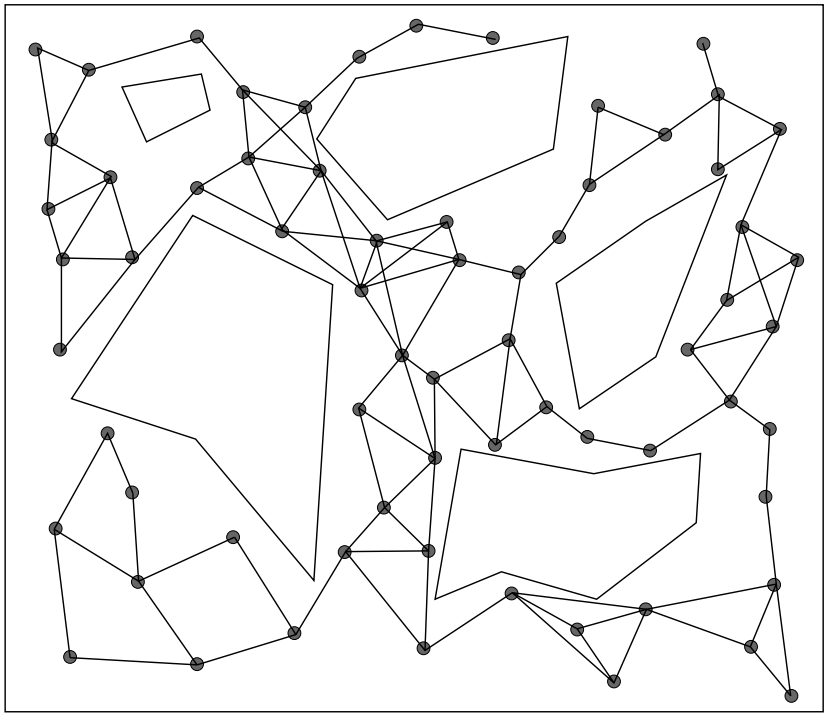


Figure 12.1: Example of a roadmap in two dimensional space constructed by the PRM algorithm.

12.4 A* Algorithm

The A* algorithm is a simplistic and reliable method compared to other path-planning techniques. Therefore it is nowadays prevalently used in a variety of different applications.[8]
It combines heuristic properties with the traditional virtues of Dijkstra's algorithm. Whereas a purely heuristic approach would not guarantee that a path can be found even if a possible solution existed, the A* guarantees that the shortest path is found.[9] A* works by trying all possible combinations like the Dijkstra algorithm in a grid or graph-based environment. With A* being based on the Dijkstra algorithm, it favours nodes closer to the current position. Additionally, it follows heuristic principles by putting a higher priority on nodes closer to the goal. An example of what an A* search algorithm looks like in a grid-based environment is depicted in Figure 12.3.[10]
With these properties, it can be considered an informed Dijkstra search.

[8] Zammit et al., "Comparison between A* and RRT algorithms for UAV path planning".
[9] Sathyaraj et al., "Multiple UAVs path planning algorithms: A comparative study".
[10] Amit Patel, *Introduction to A**.

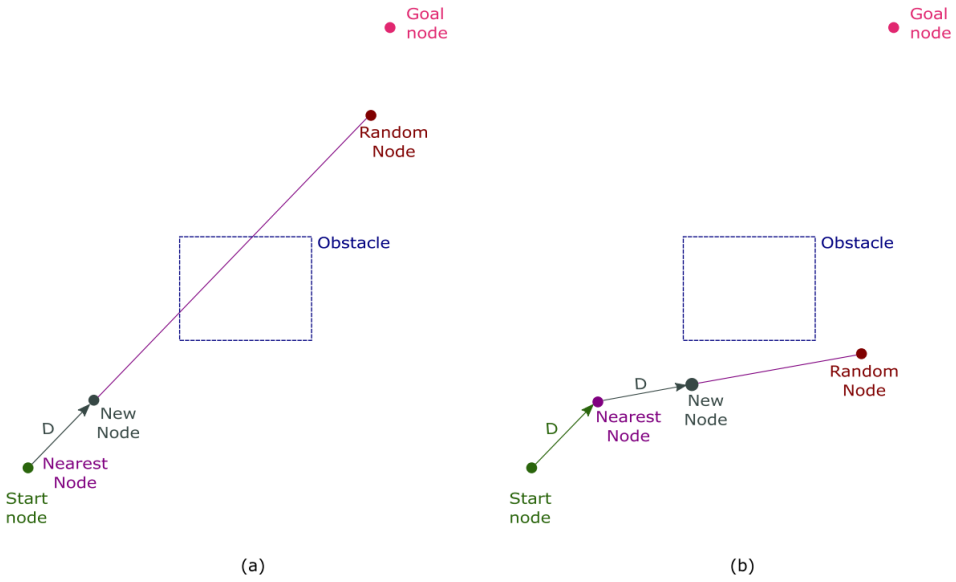


Figure 12.2: Visualization of the first two iterations (a), (b) when an rrt algorithm is exploring a simple 2d space with an obstacle present. D referring to the predefined distance between vertices.[7]

12.5 Evaluation

This section will cover the reasons behind the selection of the path-planning algorithm used in autumn. Therefore the performance of each algorithm in relation to the environment provided by the project is going to be evaluated. Additionally, the focus is on properties essential for accomplishing the goal set in the Autumn Project.

12.5.1 Autumn Use-case

There are several criteria to be fulfilled to be considered a working solution for the challenge of path planning faced by Autumn. To use one of the aforementioned algorithms, they have to be suitable for a real-time, 3-dimensional application, like the one present in the project. When working with UAVs, an optimal path must be calculated, reliable and possible in the shortest amount of time.

12.5.2 Comparison

The following comparison will look at the performance of the three different algorithms already mentioned in this chapter. Knowing that the PRM algorithm does not find a path on its own, it is implied from here on out that the PRM works in combination with the A* algorithm and the A* on its own is only applied with grid-based environments.

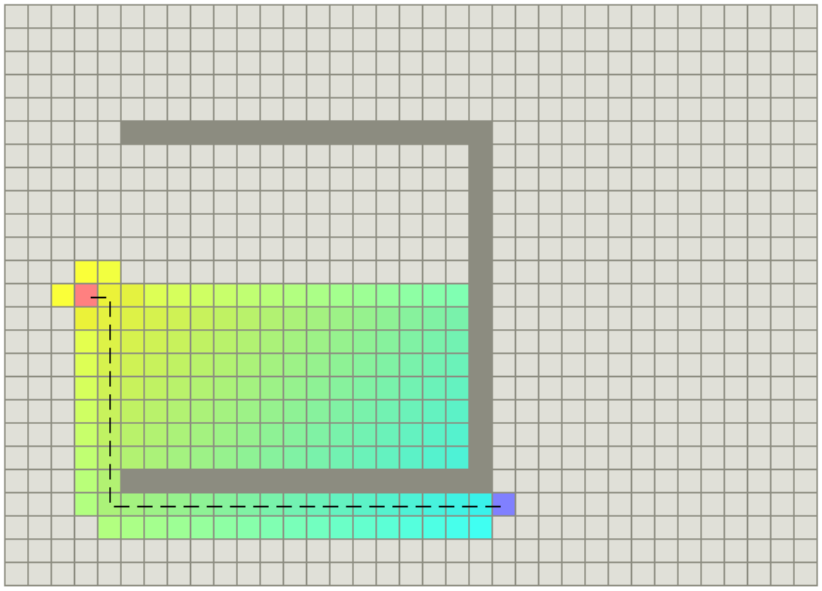


Figure 12.3: A* path finding around a concave obstacle in a grid-based environment.[11]

Computational time requirement

An essential requirement to be fulfilled by the path planning algorithm of the Autumn Project is to require as little computational time as possible. Taking a look at the PRM algorithm, with its principle of reusing the generated graph, as a roadmap, resulting in a short runtime of the algorithm, when looking at the computation of multiple paths. The best-case scenario for the PRM in this configuration would be an A* algorithm running in a graph-based environment, with the PRM only having to generate the graph and improve it progressively. A problem that arises is that this best-case scenario is impossible to happen in the Autumn Project. Like in Autumn, starting uneducated about the environment is the worst case for the PRM. At first, the PRM algorithm would generate a roadmap in an environment where most of the obstacles are still unknown. After the drone gets its first path to its destination, it detects obstacles and expands the known space for the algorithm. With the environment changing as much as in Autumn, it would be far too expensive to rewire the existing roadmap around new obstacles. A* is the best-case scenario in a graph-based environment for the PRM, but what about now running the A* standalone in a grid-based environment. Grid-based environments tend to be very resource-intensive, with full coverage of the entire space. To preclude the possibility of large portions of the search being dissipated, the A* is biased towards the goal. When comparing the A* to the PRM in terms of computational speed in Autumn, A* would be superior because of the substantial amount of time the PRM takes to generate a roadmap, only to discard it a few iterations later. That being said, the A* still needs to cover a vast grid, depending on the granularity of the grid. With this, both the A* and PRM algorithms are not the most efficient in the matter of short runtime. RRT follows a graph-based and sampling-based approach like the PRM. It covers the space to be explored within a short time frame. The RRT, in contrast to the PRM, does not take

time to construct the graph and then start its pathfinding. RRT searches for a path while constructing the tree-like structure. This results in the fastest computational time of all other mentioned algorithms when looking at one-time path planning. With the reference space of the algorithm in Autumn being drastically and continuously changing, it resembles a one-time use path perfectly fitted to be determined by a single-query path planning algorithm like the RRT.

Scalability for high-dimensional spaces

Thus far, the application of proposed algorithms happened in a two-dimensional environment. However, when working with UAVs, there is the added challenge of a third dimension to be covered, in contrast to using ground-based vehicles. This added dimension imposes new challenges that the whole system must handle, with the means to plan a path being one of the most affected. Additional time complexity being the most crucial change when working in a higher dimension, it is essential to be as efficient as possible. The last point covered the time required for each algorithm. With most path-planning algorithms having a time complexity of $O\left(b^d\right)$[12]. With A* being one of the most time-intensive. In this equation, the variable b is the branching factor and describes the number of descendants of a parent node. d refers to the depth of the goal node. The worst-case scenario would be $O\left(|V|\right)$, meaning that every vertex of the reference space has to be checked. The problem of three dimensions is that the added dimension exponentially increases the number of vertices. A* being a simple approach with brute force characteristics is not suitable for application in high dimensional spaces.

Comparing the RRT to the A* highlights the benefits of a sampling-based approach. While the A* looks at direct neighbours to continue the search, RRT has a step range as a predefined constant. Therefore the RRT explores faster due to a rougher coverage of the space. This means in relation to higher-dimensional spaces, it converges more efficiently than the A*.

12.5.3 Conclusion

The RRT came out on top by evaluating the potential algorithms to be used in the Autumn Project. With each algorithm providing unique features best fitted for the use-case they are designed for, the problems the RRT algorithm tries to solve are the same encountered in autumn. Properties like space complexity and path quality were neglected in this comparison due to the low importance of the project. By shifting the computation away from the drone, the importance of minimum time far outweighs the required space. Furthermore, the need for an optimal path is not required on account of the goal being to cover the whole space eventually. That being said, a significant drawback of the base RRT algorithm is the incapability to provide the possibility to improve on an existing path by rewiring the graph. A variant of RRT, the RRT* solves this exact problem.

[12] D.W., *A* graph search time-complexity*.

12.6 RRT* Algorithm

This section will cover the improved version of the RRT algorithm mentioned above. The RRT* algorithm is used in the Autumn Project as the path planning algorithm. The rationale for this decision is covered in the evaluation section.

12.6.1 Concept behind RRT*

A problem that arises when using the RRT algorithm is that finding an optimal path is impossible due to the lack of rewiring options. Therefore the RRT* algorithm was invented. The improved algorithm works differently when connecting a newly sampled node into the existing tree structure, which in the RRT is done by connecting the new node to its nearest neighbour. In the case of the RRT*, it calculates the cost of other possible connections stemming from the root node. Thereby a set of nodes closest to the newly sampled vertex are used as possible connection candidates. Rewiring the tree after each new node is sampled makes it possible for a near-optimal path to be found.

12.6.2 Algorithm

The RRT* works at first the same as the RRT algorithm. It first defines a set of vertices, V with the start node and an empty set of Edges, E. In contrast to the RRT algorithm is, the runtime of the RRT* algorithm is not dependent on how fast the goal node is reached but instead on the degree of optimization for the calculated path. Line four to six in Algorithm 1 are the same as the base algorithm. Thereby is a random node sampled, the nearest node in relation to the random node is calculated, and a new node is generated. After checking if a straight-line connection is possible between $x_{nearest}$ and x_{new}, a set of neighbouring vertices, X_{near} is initialized. X_{near} consist of nodes in the tree with a distance D, in relation to x_{new}. Thereafter x_{new} gets added to the V. In lines 9 to 13, the cost of paths connecting the nodes of X_{near} with x_{new} get calculated, and an edge between the node with the cheapest path, x_{min} and x_{new} gets added to E. The final stage of the RRT* is to rewire all neighbouring nodes in X_{near} if the path connecting to x_{new} is cheaper than the original path to the root vertex.

Algorithm 2: RRT* 2011[a]

$V \leftarrow \{x_{init}\}$;
$E \leftarrow 0$;
for $i \leftarrow 1$ ***to*** n **do**
  $x_{rand} \leftarrow \texttt{SampleFree}()$;
  $x_{nearest} \leftarrow \texttt{Nearest}(G = (V, E), x_{rand})$;
  $x_{new} \leftarrow \texttt{Steer}(x_{nearest}, x_{rand}, D)$;
  if `ObstacleFree` $(x_{nearest}, x_{new})$ **then**
    $X_{near} \leftarrow \texttt{Near}(G = (V, E), x_{new}, D)$;
    $V \leftarrow V \cup \{x_{new}\}$;
    $x_{min} \leftarrow x_{nearest}$;
    $c_{min} \leftarrow \texttt{Cost}(x_{nearest}) + c(\texttt{Line}(x_{nearest}, x_{new}))$;
    foreach $x_{near} \in X_{near}$ **do**
      if `CollisionFree` $(x_{near}, x_{new}) \wedge$ `Cost` $(x_{near}) + c(\texttt{Line}(x_{near}, x_{new})) < c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$;
        $c_{min} \leftarrow \texttt{Cost}(x_{near}) + c(\texttt{Line}(x_{near}, x_{new}))$;
      end
    end
    $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
    foreach $x_{near} \in X_{near}$ **do**
      if `CollisionFree` $(x_{near}, x_{new}) \wedge$ `Cost` $(x_{new}) + c(\texttt{Line}(x_{near}, x_{new})) < c_{near}$ **then**
        $x_{parent} \leftarrow Parent(x_{near})$;
      end
      $E \leftarrow (E \backslash \{(x_{parent}, x_{near})\} \cup \{(x_{new}, x_{near})\})$;
    end
  end
end
return G = (V, E);

[a] Karaman et al., "Incremental sampling-based algorithms for optimal motion planning".

12.7 RRT* Variants

12.7.1 RT-RRT*

The RT-RRT* takes focus on real-time path planning. RT-RRT* makes it possible to reposition the root node. Using this online tree rewiring strategy makes it possible to keep the previously sampled tree. The two core principles introduced with the RT-RRT*are tree expansion and tree rewiring. At first, the Algorithm starts by initializing a root node. After each following iteration, the tree expands and rewires. This sampling lasts for a user-defined time and is followed by planning a path from the root. The user controls the length of this

path by defining the number of steps the path consists of. In this phase, the agent is moved gradually towards the tree root. Thereafter, when path planning is finished, and the agent is located at the position of the tree root, the position of the root is changed to the next node in the generated path. This approach enables the agent to move without the time needed to compute the whole path. Expansion in the RT-RRT* is done similar to the base algorithm by sampling a random node, finding its neighbour with the minimum cost-to-reach and connecting them. The rewiring is done in the default scenario because newly added nodes have a smaller cost-to-reach than the parent of a specific node. However, rewiring is needed if the environment changes in the second case. Therefore a more significant portion of the tree has to change. Thereby two different modes are utilized. The first one consists of rewiring the tree in a circle centred around the tree node. For the second option, both focused and uniform sampling is used, with the difference that it is not done for one node but instead concentrates on patches.[13]

RT-RRT* in Autumn

When applying the concept of the RT-RRT* to Autumn, it would seem a good idea. However, a closer look uncovers that the proposed RT-RRT* only works efficiently in environments with limited changes. Rewiring the tree, done in the case of a dynamic obstacle changing, does not make sense when the obstacle referred to covers a large portion of the space. This is the case in Autumn because contrary to the examples in the paper covering the RT-RRT*, the algorithm is not educated about the environment. Only while moving are obstacles and boundaries discovered. In general, the RT-RRT* uses concepts of road-maps like the one generated in the PRM algorithm and applies it to the RRT* algorithm.

12.7.2 Smart-RRT*

Smart-RRT* focuses on solving the problem imposed using the RRT* algorithm. However, with the slow convergence rate of the RRT* towards an ideal path, it takes up to an infinite amount of time to reach the most optimal path. Therefore Smart-RRT* utilizes two techniques, path optimization and intelligent sampling, in an effort of accelerating the rate of convergence. It works the same as the base algorithm till the first path is found. From there on out, it applies path optimization and intelligent sampling. Path optimization works by interconnecting nodes in the found path that are directly visible. This leads based on triangular inequality to a shorter path. The reconnected nodes are then used as biased points for intelligent sampling. Intelligent sampling samples new points based on a bias generated from finding a shorter path. The result is a greater node density in the critical points on the path, primarily located around obstacles.[14]

Smart-RRT* in Autumn

Using Smart-RRT* improves the convergence rate and time to reach an optimal path drastically. This leaves the question of why not use this improved version of the implemented

[13] Naderi et al., "RT-RRT: A real-time path planning algorithm based on RRT".
[14] Islam et al., "RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution".

algorithm in the project. When looking at this, it is vital to keep the goal in mind that the Autumn Project aims to reach, which is to capture an environment in the form of a 3-dimensional model. Therefore it is unnecessary to compute an optimal path. The whole space will be covered eventually, which would only be a waste of computational power to invest in additional path optimization.

12.8 autumn_pathplanning_2d Benchmark

This section will concern itself with the performance of the autumn_pathplanning_2d Algorithm. The main aspects of these tests are reliability, computational time requirement and path length. That are the most significant values in the case of the Autumn Use Case.

12.8.1 Experimental Setup

The experiment is divided into three different parts. Each of these test scenarios contains data for a range of varying amounts iterations. By the nature of the RRT* algorithm, the number of iterations defines the resulting path's quality. Quality in this context refers to the length of the generated path. The higher the iteration count, the higher the probability of getting a shorter path. These iteration samples range from 1000 to 10000.

Test Environments

The three tests were conducted in two different environments. Test 1 was carried out on Map 1, to be seen in Figure 12.4. Defining characteristics of Map 1 are an open environment with minimal obstacles. These open style environments without any borders present a challenge for the algorithm closer described in the Section 12.8.3. Tests two and three were performed on Map 2, which can be seen in Figure 12.4. Map 2 is a closed environment with higher complexity than Map 1.

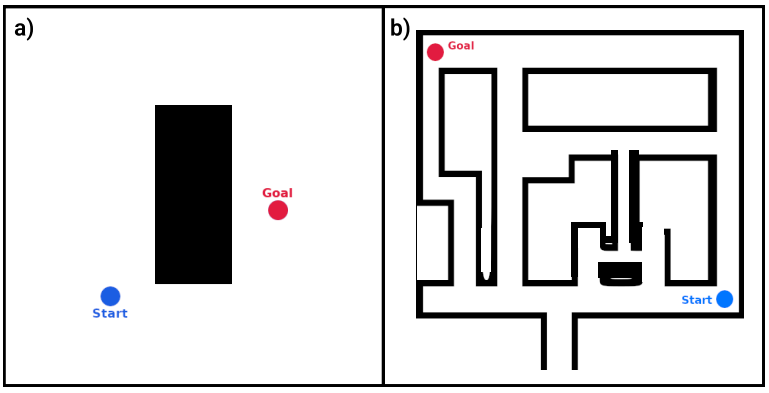


Figure 12.4: a) showing map 1 for test 1. Map 1 consist of a rectangular obstacle in the centre. b) is displaying map 2 used for test 2 and test 3. It compromises a three path design, which can be seen by three distinct paths to the goal, bottom, top and through the middle.

12.8.2 Results

The following will present the results of tests conducted to measure reliability, computational time requirement and path cost.

Test 1

Test 1 was conducted on map 1, with iterations between 1000 and 8000, with an increment of 1000. To minimize the impact of the random nature of the RRT algorithm, every configuration generated 100 paths, and the time and cost present in Table 12.8.2 is the average of all 100 runs. The same table shows how many of the runs failed, which means they could not get to the goal, and no path was found. Additionally, to demonstrate the data coherence of the individual runs the standard deviation of the measured time as well, as the standard deviation of the path cost is listed.

Iterations	1000	2000	3000	4000	5000	6000	7000	8000
Mean time[ms]	36,61	139,92	306,20	533,78	818,56	1165,98	1563,83	2018,33
Mean cost	118,52	91,71	82,43	78,72	76,37	75,49	74,58	74,27
Failed[%]	2	0	0	0	0	0	0	0
σ time[ms]	3,70	10,24	17,40	24,41	30,08	40,22	47,50	51,73
σ cost	38,72	14,58	7,24	4,63	3,59	2,85	1,96	1,69
Min time[ms]	33,37	130,07	286,27	498,46	771,62	1101,18	1481,98	1909,52

Table 12.1: Results of test 1 conducted on map1

Test 2

Test 2 uses the same principle as test 1, while being performed on map 2. Furthermore, test 2 uses a sample size of 500 paths per iteration configuration. Additionally, nodes in test 2 are closer together due to the smaller node spacing parameter.

Iterations	2000	3000	4000	5000	6000	7000	8000	9000
Mean time[ms]	8,36	13,66	19,54	25,36	32,31	38,73	46,33	53,84
Mean cost	181,82	181,99	181,21	181,00	180,65	180,55	179,77	179,59
Failed[%]	36	25	16	16	12	12	9	8
σ time[ms]	4,60	6,63	7,98	10,06	11,51	14,04	14,57	16,23
σ cost	88,21	79,14	67,38	66,64	58,74	60,41	52,78	49,56
Min time[ms]	4,92	4,21	7,00	7,05	7,71	11,07	17,20	18,30

Table 12.2: Results of test 2 conducted on Map2.

Test 3

Test 3 works with the same principle as the aforementioned tests. Test 3 was also performed on map 2 and serves as a control run to test the significance of the results of test 2. Test 3 is a replica of test 2.

Iterations	2000	3000	4000	5000	6000	7000	8000	9000
Mean time[ms]	10,93	17,47	24,96	32,66	41,13	49,07	58,34	69,14
Mean cost	181,89	182,17	181,47	181,50	181,45	180,28	180,02	180,12
Failed[%]	39	23	15	11	11	7	9	8
σ time[ms]	5,97	8,06	9,56	11,30	14,17	14,49	19,15	20,57
σ cost	89,29	77,21	64,72	57,30	58,05	47,50	52,91	50,36
Min time[ms]	4,17	5,85	10,42	14,81	9,69	12,41	13,51	15,87

Table 12.3: Results of test 3 conducted on Map2.

12.8.3 Evaluation

Comparing the results from Section 12.8.2, it is made obvious that test 1 has a much higher time requirement than test 2 and 3. This can be observed in figure 12.5. It has do mentioned that this plot uses a logarithmic y-axis. The behaviour can be ascribed to the fundamental difference between map 1 and map 2. Map 2 has borders containing the algorithm, while map 1 does not. The enclosed environments are better suited because if the algorithm encounters an obstacle or, in this case, a wall, it will start to propagate along with it, resulting in fewer nodes being placed in the wrong direction to the goal. Another essential fact to take into account is that tests 2, and 3 use a smaller node spacing parameter. By this means, newly generated nodes are closer to their nearest neighbour, which results in less elaborate collision detection. An observation made looking at Figure 12.5, that was already anticipated, is that with an increasing number of iterations, the time to compute a path also increases.
Now looking at Figure 12.6 the way the cost of generated paths is affected by varying the number of iterations can be observed. Thereby only data from test 2 and 3 are present, due to the fact that test 1 uses a map with a different minimal path length and thus can not be compared with paths from test 2 and 3. The different data points vary strongly, which is primarily due to the fact that the range of iterations is relatively small for this use case. The average time needed to compute a path using 8000 iterations only amounts to 53ms, which is slower than 8ms for 2000 iterations but still quite fast. In order to have more consistency in the decline of the path cost with increasing iterations, a higher iteration count would be needed.
The last important aspect of this path planning algorithm being its reliability, which can be observed in Figure 12.7. Test 1 being ahead of both test 2 and 3, with a maximum of about 20% failed. This is due to the fact that test 2, as well as test 3, are performed in a complex environment, were as test 1 was conducted in an environment presenting hardly any challenges. Furthermore, in test 1, the goal is closer than in test 2 and 3. With 1500

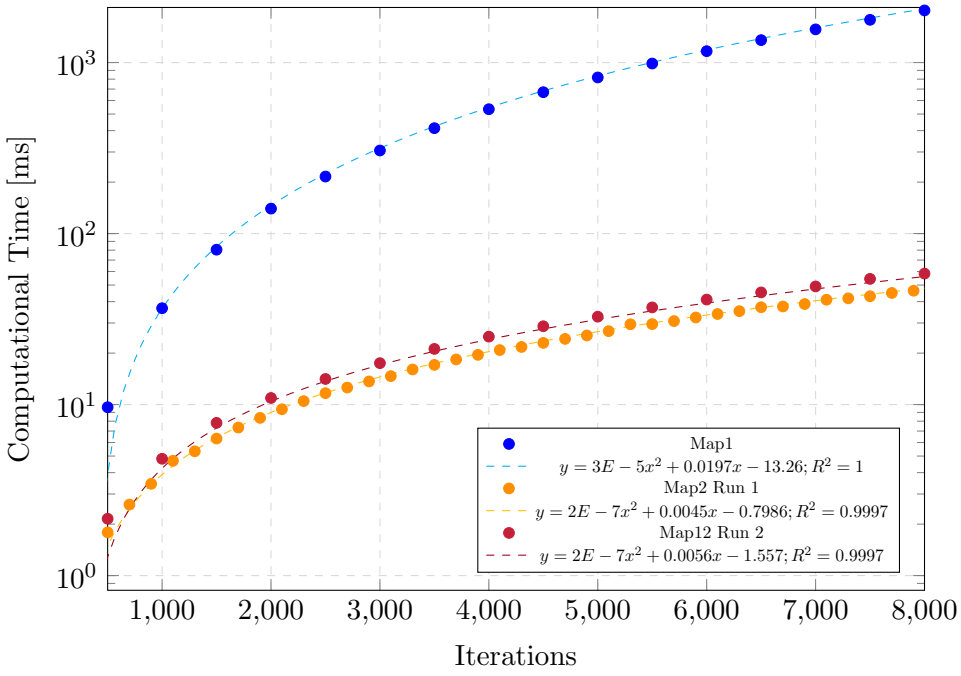


Figure 12.5: This plot displays time data from test 1(blue), test 2(orange) and test 3(red).

iterations, test 1 has a 100% success rate. What also needs to be taken into account is that tests 2 and 3 are by a multiple faster than test 1. Thereby, it would be no problem to lower the failure rate by increasing the number of iterations while still being faster than the comparable results from test 1.

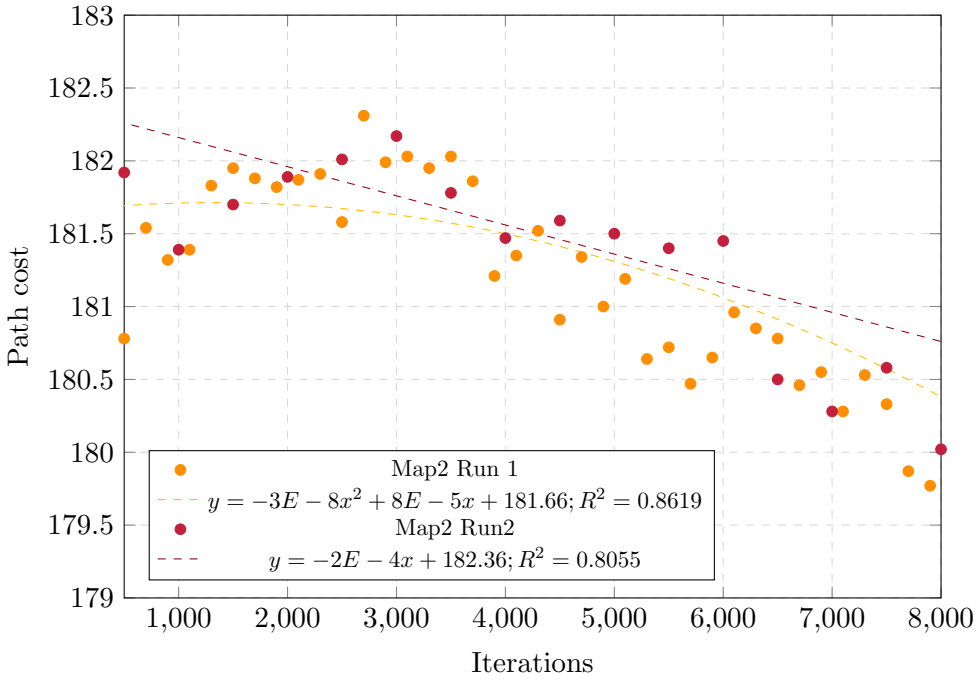


Figure 12.6: In this plot path cost data from test 2(orange) and test 3(red) are compared.

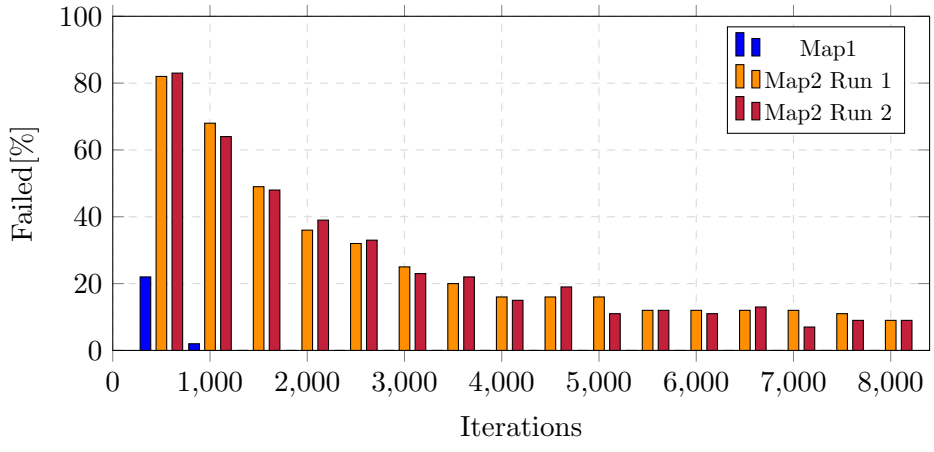


Figure 12.7: This plot visualized the failure rate of paths planned for the same iteration configuration, for test 1(blue), test 2(orange) and test 3(red). The values in this plot refer to the percentage of failed path planning attempts at the respective iteration count.

Chapter 13

Collision Avoidance

Author: Fabian Kleinrad
This chapter covers the method of detecting collisions within the path planning phase. The needed information about the environment stems from the data structures covered in chapter 11.

13.1 Fundamental Principle

An essential part of autonomy in robotics is the aspect of collision avoidance. This is only possible if the robot has the means to identify and detect possible collisions and act suitably to circumvent collisions from happening.
In Autumn, this step of collision detection happens alongside path planning. With collision detection and collision avoidance being a very cost-intensive process, it is essential to optimize the algorithm for better performance while simultaneously keeping a safety margin. This safety margin is especially important since the algorithm controls a UAV.

13.1.1 Base Algorithm

The principle of collision avoidance is elementary due to the organized and semi-organized data structures the algorithm uses. A function can be called that checks for collisions. It takes Cartesian coordinates as parameters and returns a value that indicates if an obstacle is present or not. All the collision detection algorithm has to do is check if there are any collisions between two points sampled by the path planning algorithm. To accomplish this task, Bresenham's Algorithm will be used.

Bresenham's Algorithm

Bresenham's Algorithm is a solution to the problem of rasterizing curves, rasterizing being the process of converting curves or lines into cells on a grid, representing the same shape. The Algorithm covers many different shapes, but only a straight line will be necessary because of the nature of the path planning algorithm. The algorithm works by calculating an error for

each candidate cell. The formula for this error being: $e = (y - y_0)dx - (x - x_0)dy$. Hereby x and y are the coordinates of the cell the error is calculated for, which are being subtracted by the coordinates of the starting point. dx and dy are the differences between the first and last point of the line. The error symbolizes the deviation from the original line. Calculating the errors for each cell makes it possible to choose cells best representing the original line.[1]

Algorithm 3: Bresenham's Line Algorithm[2]

```
1  dx ← abs(x1 − x0);
2  dy ← abs(y1 − y0);
3  sx ← x0 < x1?1 : −1;
4  sy ← y0 < y1?1 : −1;
5  error ← dx + dy;
6  while x0 ≠ x1 and y0 ≠ y1 do
7  |  UsePixel(x0, y0);
8  |  e2 ← error ∗ 2;
9  |  if e2 ≥ dy then
10 |  |  error ← error + dy;
11 |  |  x0 ← x0 + sx;
12 |  end
13 |  if e2 ≤ dx then
14 |  |  error ← error + dx;
15 |  |  y0 ← y0 + sy;
16 |  end
17 end
```

13.2 Implementation

The collision avoidance logic is in effect every time a new point is added to the graph. The path planning algorithm samples a random point and calculates its nearest neighbour. After that, a function is called, which takes two points and checks for collision on a straight-line path between these two points.
A problem that arises when using it this way is the not accounted for dimensions of the drone. For that, the function takes a third parameter defining the search radius. Using this method, paths can be generated suitable for the physical drone. However, this is accompanied by an increase in computational time, which stems from how this radius, r, is used. For every cell that is not the first and last cell, $2r$ neighbouring cells in a line opposite the line's direction get checked. For the first and last cell, the radius translates to the circle where cells get checked. In order to not overlap validation areas, on the first and last cell, only a semicircle is being covered.
Therefore the to be checked cells can be calculated in dependency of the line length l and radius r. l thereby being the number of cells present in the line calculated by Bresenham's

[1] Zingl, "A Rasterizing Algorithm for Drawing Curves".

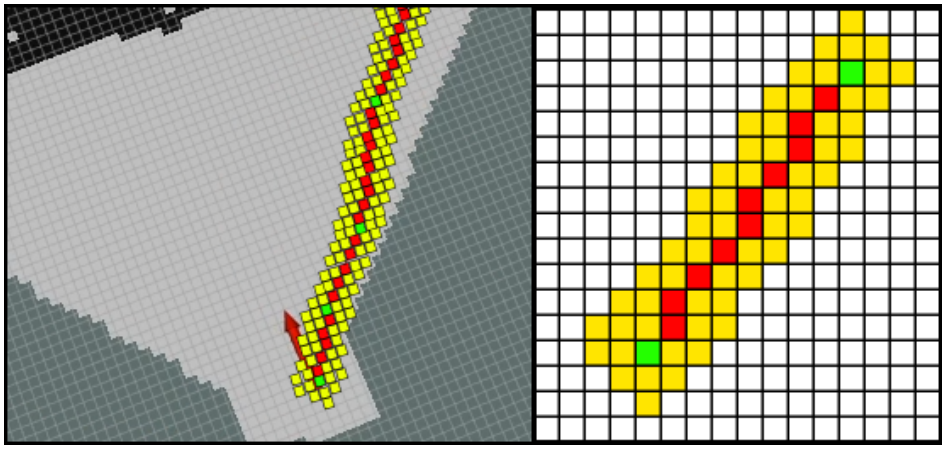


Figure 13.1: Visualization of checked pixel in paths. Green being start/end points, red the lines calculated by the Bresenham's Algorithm and yellow the additionally checked cells with $r = 2$.

Algorithm.

$$C = r^2 + (l - 2) * 2r$$

13.2.1 3 dimensional space

Using this concept in a 3-dimensional environment only requires adding the new third dimension. Bresenham's Algorithm is easy to translate into the third[1]2 dimension. The problem that arises is performance. When adding the radial search, it increases the total cells checked by $C^2 - l$.

13.3 Experiment

In order to be able to assess the practical impact of the collision avoidance performed by the Autumn Pathfinding algorithm, an experiment was conducted, which illustrates the computational time needed to guarantee no collisions from happening.

13.3.1 Experimental Setup

The experiment is split into two parts: the impact of collision avoidance on the computational time of two-dimensional and three-dimensional path-finding. To get significant data, which can represent various scenarios in which the algorithm has to perform, a setup was selected to cover all aspects of the algorithm. The experiment works by planning a path through a static environment. This environment stays the same for all iterations of one algorithm and the two different variations of the algorithm. To get conclusive evidence concerning the collision avoidance, which is not polluted by the random nature of the algorithm, the sample size has been set to 100.

One sample refers thereby to a path planning with consistent parameters. The parameters are

- the number of iterations of the RRT* algorithm(i),
- the distance between the start and end node(d), and
- the spacing of the nodes of the expanding tree(D).

These three parameters stay constant throughout the entire duration of the experiment, the reason being that these three Parameters affect the parts of the path planning algorithm that aren't of concern in this experiment. The Parameter controlling the collision detection behaviour is the radius(r). For each radius, 100 paths are generated, and each execution of the algorithm is timed from the point of the function call till the return of the calculated path. The set of radii to be tested depends on the space in which paths are generated. For each radius, the meantime of all 100 data points is calculated. This results in data representing how collision avoidance impacts the computational time of the Autumn Path-Planning algorithm.

13.3.2 Two-dimensional Algorithm

For autumn_pathplanning_2d, radii starting from zero, resulting in disabling collision avoidance functionality, up to two 2000, with a step size of 100, have been tested. In figure 13.2 the linear relationship between collision avoidance and the computational time can be observed. Furthermore, it can be derived from this plot that the two-dimensional path planning algorithm can handle high amounts of collision checks without making the path calculation unusable. On average, an additional 100 in radius results in a 90ms longer runtime. In this context, it should be pointed out that the radii chosen in this experiment are not in relation to radii used in practical application. For a 0.05 resolution of the occupancy grid used, radii in a range from 15 to 50 are practical. Nevertheless, this accentuated representation with a correlation coefficient of nearly one can also be applied to the practical values.

13.3.3 Three-dimensional Algorithm

The three-dimensional algorithm was tested with values ranging from 10 to 320, with an increment of $2^n * 10$. Figure 13.3 illustrates the result of the experiment performed using the autumn_pathplanning algorithm. The exponential relationship between the number of collision checks and computational time is apparent. With a block size of the point cloud of 0.01, the radii used in practice would range from 20 up to 150.

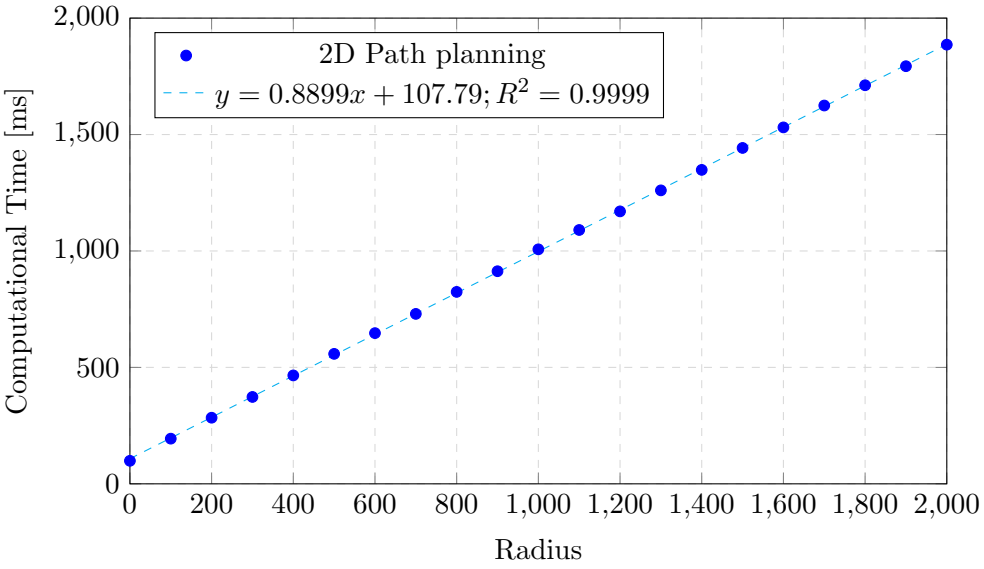


Figure 13.2: This plot illustrates the increase in computational time with increased collision detection, for the two-dimensional path-planning algorithm. Each point represents the average time needed to compute 100 paths, with equivalent parameters.

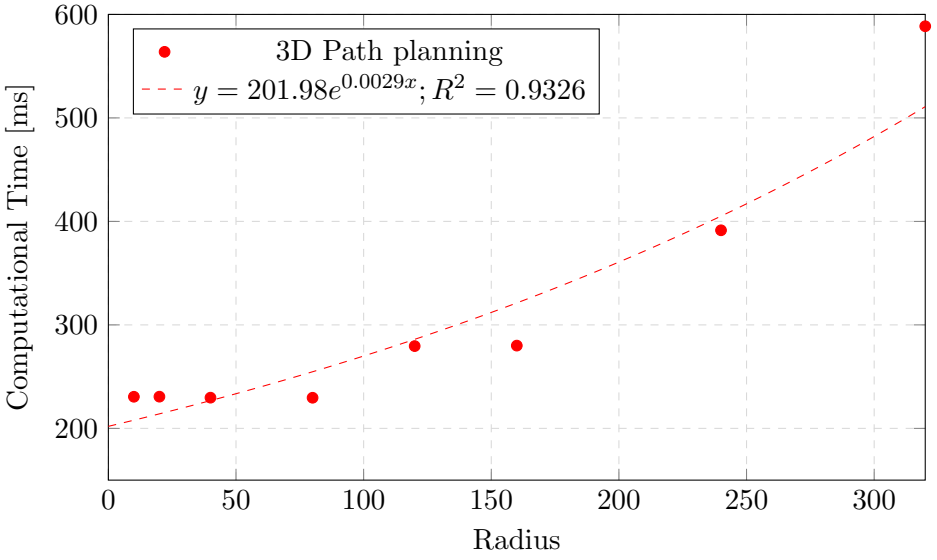


Figure 13.3: In this plot the correlation between increased collision detection and computational time, for the three-dimensional path-planning algorithm.

Chapter 14

Custom Parts

Author: Fabian Kleinrad
This chapter takes a look at the practical side of the Autumn Project. There is an inevitable need for parts fit for the underlying application when working with hardware. In the case of Autumn, with the Matrice 100 being in the centre of the project, the need arises to design and manufacture parts to make the hardware portion in Autumn come together without any complication and potential weak points in the final product.

14.1 Reasons

The goal of the Autumn Project being as innovative and unique in terms of methods used in the realization of the project, parts needed are either not readily available or not compliant with the budget. Another difficulty is that parts are too specific for there to be any commercially available. Therefore to guarantee the reusability and reliability of the final product, methods to design and produce parts specifically tailored to the needs in the Autumn Project have to be utilized.

14.1.1 Camera Mount

Autumn conquers the problem of mapping an environment with the use of a drone. Therefore it is necessary to attach the camera used to capture the surroundings to the drone that allows it to move through that space. In the case of autumn, the camera being used is a ZED2i, which needs to be securely attached to a Matrice 100. Additionally to the cost factor and accompanying risk of damaging the hardware in use, the positioning and angle are essential properties to consider. For that reason, a commercial solution is precluded.

First approach

When working under the principle of fast prototyping, the drawbacks of the first approach being incomplete is inherent. Following this principle led to attaching the ZED2i to the Matrice 100 with the help of zip ties. The rationale for this decision was the reliability and strength of zip ties which allowed the camera to be secured tightly. This was important

because of the need to test segments of the Autumn Project, worked on separately and isolated from one another. Taking simple and easy to realize steps enables the project to progress more smoothly without requiring other sub-areas to be set on hold because of long, tedious planning and manufacturing periods of parts like a custom camera mount for a drone.

Problems without a Mount

After a period of testing with a prototype, problems arise with the need for correction. In the case of, what most people consider a work-around rather than a solution to the problem of mounting a camera, they can be divided into two categories in the case of this application. The first aspect to consider when mounting a camera to a drone is its field of view. With UAVs hovering above the ground, a downward pointing angle is needed in order to be able to capture details close to the ground.

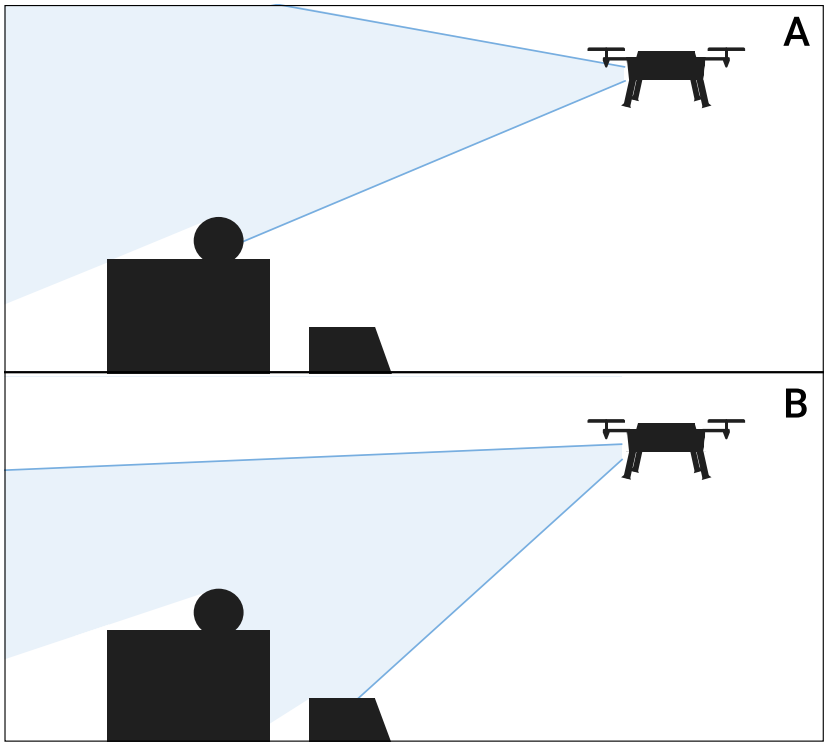


Figure 14.1: Depiction of the FOV for cameras mounted at different angles on a drone.
A: camera mounted vertically, B: camera mounted with a downward pointing angle.

Using zip ties results in the camera being mounted parallel to the ground. This leads to overlooking obstacles located below the field of view. This can be seen in figure 14.1 A. Adjust the angle to be able to see as much as needed, directly in front of the drone and the more significant portion of what is below the drone. Having the ability to capture obstacles directly in front of the drone is necessary when wanting to let the drone autonomously navigate an environment successfully. Therefore is, a zip-tie solution valid but far from optimal. A result of having a camera mounted parallel to the ground is the need to hover closer to the ground,

which is potentially dangerous considering that we cannot capture obstacles directly below the drone in autumn.
The second problem that arises when not using a mount fitted for use with Autumn the ZED 2i is that the camera is not parallel to the ground. This portrays a significant flaw when using an uneven camera combined with a SLAM Algorithm. SLAM uses odometry data for localization and alignment of the ground plane. If the camera is slightly tilted when starting the algorithm and without any manual adjustment, the ground plane will be offset on one or multiple rotational axes. This results in the 3D model being a false representation of reality. This phenomenon can be observed in Figure 14.2.

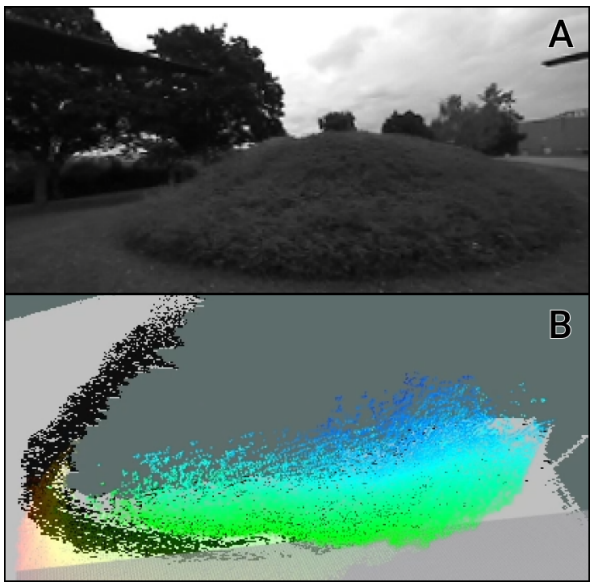


Figure 14.2: Resulting point cloud of unleveled camera with invalid odometry start data.
A: real life footage of the obstacle, B: tilted scan of the obstacle.

Design Process

The solution to the aforementioned problems is a mount securing the ZED2i to the Matrice 100. Additionally to the specific characteristics the mount must fulfil, the ZED2i is the newest model, with commercial products like mounts not being available. Therefore, to attain this custom mount design, manufacturing methods must be used. The design portion is done with the help of CAD Software. CAD stands for Computer-Aided Design, meaning that rather complex designs can be realized in a short amount of time and less effort than doing it the traditional way. The software used in the Autumn Project is Fusion360 due to its option for a free license and previously acquired knowledge working with this software. The mount had to perfectly fit the camera to ensure the safety of the camera and drone. Therefore it is necessary to take measurements, which then are used to shape the design of the 3D model. In Fusion360, this can be best realized using parameterized design.
Figure 14.3 depicts this concept of making the design dependent on parameters that can be

Name	Unit	Expression	Value
mountLength	mm	60 mm	60.00
plateAddMountD	mm	5 mm	5.00
mountDepth	mm	(plateHtotal * cos((360 deg - mountAngle) - 180 deg))	5.367
plateHtotal	mm	plateH + plateBottomH + platAddDHeight - gapFree	12.70
mountBottomAngle	deg	180 deg - ((360 deg - mountAngle) - 90 deg)	25.0
mountCableHoleHP	mm	2 mm	2.00
mountCableHoleSideT	mm	8.5 mm	8.50
mountCableHoleH	mm	18 mm	18.00
mountAddLenght	mm	(plateHtotal * sin((360 deg - mountAngle) - 180 deg) ...	11.51
conPlateW	mm	45 mm	45.00
conPlateD	mm	42 mm	42.00
conPlateCenterWallT	mm	13 mm	13.00

Figure 14.3: Excerpt of parameters used for the Autumn camera mount design.

easily adjusted. This allows for corrections to be made after the design process is finished. Without linking, sketches to parameter alterations would be time-consuming and result in a model clouded with sketches overriding previous adjustments. Furthermore, changes affecting complex areas in the part lead to artefacts, which, when removed, take more time than redesigning the part, in the context of simple, one-component models, like the mount in autumn. An important property in the case of the Autumn camera mount is the angle of the camera relative to the drone. This design principle makes it possible that changing the angle only implies changing a number in a text field, which implies the change of several different dimensions of the part. Fusion 360 additionally comes with the capability of rendering the resulting model to get a feel for how it would look in physical form.

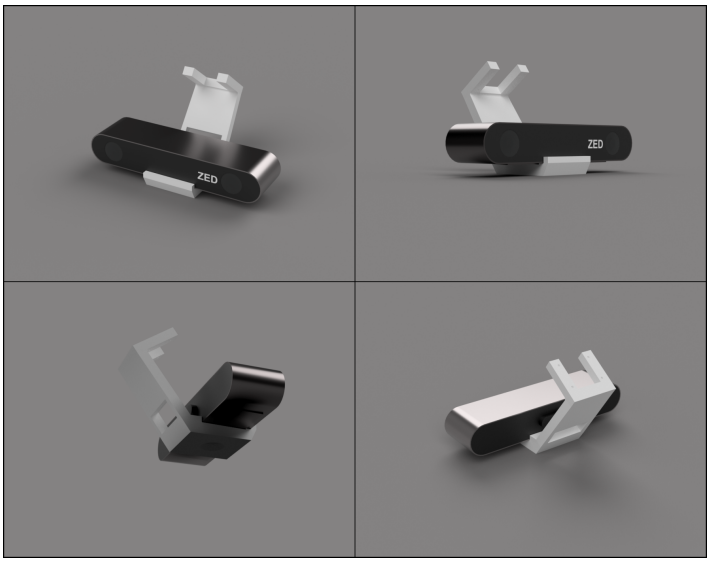


Figure 14.4: Render of the 3D-model for the camera mount attaching the ZED2i, also modeled and visible in the render, to the Matrice 100.

Manufacturing

For the purpose of converting the 3D model to a physical object, the most straightforward and most time-efficient method is to utilize 3D printing. Nowadays, 3D printing is a widespread technique to make ideas reality with little time and money. Therefore it is an optimal solution for the Autumn Project. When going from a 3D-modeled design to a 3D print, a slicer is needed to convert the 3D object to code the printer understands. With most 3D printers printing in 2.5 dimensions, meaning that only the x and y-axis move freely while the z only adjusts its height once a layer is finished. This principle is reflected in the resulting file once a 3D model is sliced. The gcode consists of a header, defining settings for the print and a sequence of lines following it. A line of this gcode file would look like this:

$$G1\ X120.95\ Y101.653\ E1.17355$$

The first part consists of the identifier G and a number, marks which command should be executed. For example, in the case of $G1$, it is "move in a straight line". Following the command, specifiers are parameters; the x and y end position is needed to draw a line. Additionally marked with the identifier E is the flow rate defining how much filament will be extruded when moving to the given position.
The final step to attaining the physical form of a gcode file is printing it using a suitable 3D printer. Autumn uses a Creality Ender 5 because of its readability and print quality. Furthermore, the choice of the correct filament for a successful 3D print is needed. For a part, like a camera mount, which is not under significant stresses and no large forces are applied, filament-like acrylonitrile butadiene styrene, commonly known as ABS, is not needed. Therefore the most widely used filament, polylactide, is most fitted for this print.

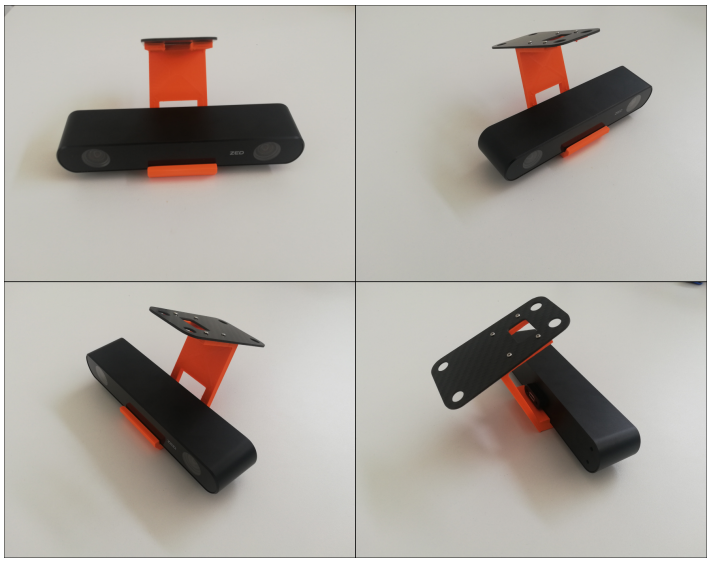


Figure 14.5: 3D-printed camera mount for the purpose of attaching the ZED2i to the Matrice 100. On top of the mount is the connection plate attaching on the drone.

14.2 CAD Software

This section is going to cover widespread CAD software solutions. Nowadays, computer-aided design is an imperative principle used in various industries. CAD software improves the efficiency and quality of the design workflow. With free solutions being available, this industry-standard has expanded to a clientele reaching from the leading companies to hobbyists, creating future innovation in the comfort of their homes. Therefore when choosing an apt software, a multiplicity of properties must be considered.

14.2.1 SolidWorks

SolidWorks being the industry-leading CAD solution, it is hard to overlook in a comparison. SolidWorks offers a wast array of functionality for CAD and offers capabilities for computer-aided manufacturing. The target audience for SolidWorks is experts in the field of modelling and design. Thereby is not only the UI designed to cater more to a clientele that is firm in their understanding of the computer-aided design process, but also it offers more in-depth functionalities such as dynamic loading, linear and non-linear analysis and composite materials.[1]

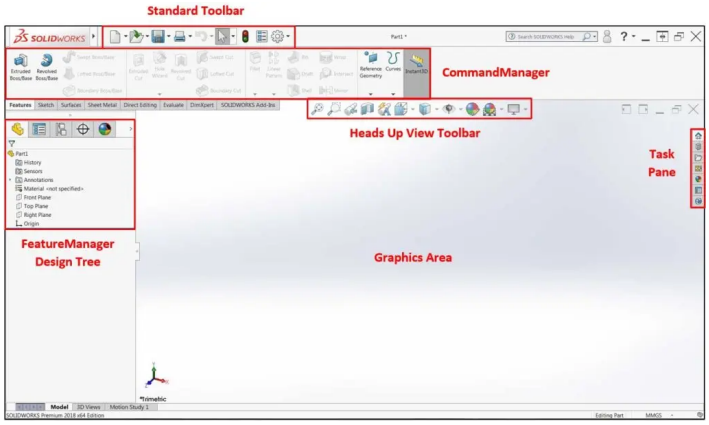


Figure 14.6: *SolidWorks user interface broken up into its different components.*

14.2.2 Fusion 360

With Fusion 360 offering a non-commercial, free license option with limited functionality, it is widespread amongst the hobbyists and with its beginner-friendly UI design and possibility for an educational license with the majority of the functional scope of the professional license. Features included in the professional subscription that is excluded otherwise are mostly cloud processing options and load and stress simulations.

[1] Lucas Carolo, *SolidWorks 2022 vs Fusion 360: The Differences*.

Fusion 360 Licenses		
License	Features	Price
Fusion 360 for personal use	Standard design and 3D modeling tools	free
Fusion 360	Standard design and 3D modeling tools, plus a fully featured CAM, CAE, and PCB development platform	495$ p.a.

Table 14.1: An overview of Fusion 360 licenses with their respective capabilities and prices.[3]

14.2.3 Autumn Choice

When comparing the two CAD solutions that are possibilities to be used in the Autumn Project, the most crucial characteristic is user-friendliness regarding the state of knowledge the user already has. Both SolidWorks and Fusion 360 offer more than enough functionality to fulfil the needs of autumn. Fusion has no competition with its free license for personal use when taking a look at the monetary dimension. Furthermore is, the Fusion 360 workflow tailored to the simple application, with a small number of components. SolidWorks follows an assembly based principle. In contrast, Fusion 360 is based on a multi-component architecture. In conclusion, due to the limited functionality required from CAD software in the Autumn Project, Fusion 360 is being used.

14.3 3D-Printing

This section will focus on manufacturing methods available in the Autumn Project and compare them. Nowadays, easy and fast prototyping is made possible using 3D-printing technology. 3D-Printing has evolved over the years, away from a science fiction concept to a widely used tool in part manufacturing, filling the gap left open by the geometrical limitations of conventional methods.
Autumn having an area of competence, split between skills when working with soft- and hardware, it is vital to have the means to manufacture custom parts tailored to the project's needs. With 3D-Printing being an affordable and accurate technology, it is an obvious choice to use for Autumn in terms of translating virtual concepts to a physical object. When considering using 3D printing in any project, it is essential to consider which method is best fitted for the application.

14.3.1 Fused Deposition Modeling

FDM printing is the most commonly used and widespread method of 3D printing. The principle behind fused deposition modelling comprises using an extrusion device to eject a thermoplastic polymer. This polymer is heated to the point of its glass transition and, consequently, extruded along a predefined path.

Movement Variants

The movement capabilities of 3D printers differ between two different approaches. First, most desktop 3D printers are Cartesian-style 3D printers, which means it moves linearly in the x, y and z dimension. Which parts of the printer move depend on the printer itself. Commonly the print-bed moves in the y dimension, while the print-head covers x and the vertical z movement. In Contrast, the delta-printing method uses three vertical rods aligned in a triangle shape to support the print-head and move it per individual movement of sledges connected to the print-head on the vertical rods.[4]

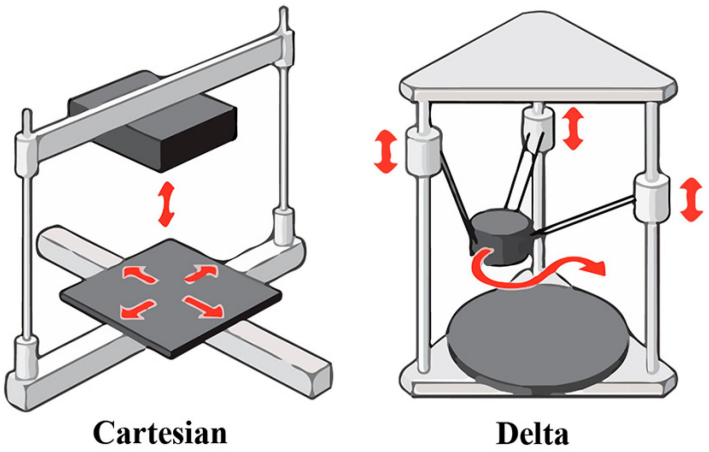


Figure 14.7: "A comparative study of cartesian and delta 3d printers on producing PLA parts".

Cartesian-style and Delta-style printers fundamentally contain the same parts. Both printer variants include a print-bed, print-head and stepper-motors. The difference is how they are arranged. Therefore the performance of these printing styles is similar and are more dependent on other factors than the movement system. The reason Cartesian-style printers are as popular as they are is due to the fact that the more straightforward Cartesian approach is user-friendlier and consumes less space than its delta-style equivalent.

Main Parts

In order to be able to distribute the filament according to the predefined path calculated by the slicing software, components are needed, namely an extruder, hot-end, and nozzle. The extruder is used the feed the polymer to the print-head. The print-head consists of a hot-end and a nozzle. The hot-end is used to heat the thermoplastic polymer to the state of glass transition, after which it is forced through a nozzle and applied to either the print-bed directly or on top of another layer.[5]

[4] Emmett Grames, *What is FDM 3D Printing? – Simply Explained*.
[5] Ibid.

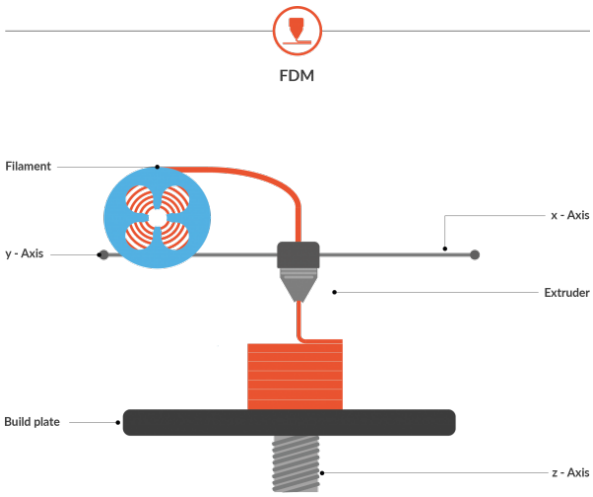


Figure 14.8: *Depiction of FDM printer parts, in an Cartesian-style design.*

Filament

The aspect of the filament used for a print heavily influences the success and usability of a part. The most common filaments include PLA, ABS and Nylon. PLA is the most frequently used. When needing a more robust material than PLA, a suitable polymer would be ABS. Cases where Nylon is used are when highly flexible parts are desired.[6]

14.3.2 Stereolithography

SLA is known for its accuracy and detail in conjunction with 3D printing. SLA-printers use a photosensitive resin combined with a laser to construct a part. This allows printers applying this method to keep mechanical components to a minimum. In contrast to FDM printing, SLA uses thermoset polymers, leading to irreversible parts once printed. FDM printed parts will melt when exposed to heat. SLA printed parts do not turn back to liquid. Instead, they start to burn.[7]

Printing process

At the start of every print, depending on the printer design, the build-plate is positioned so that one layer height of resin is present between the build-plate and laser. Thereafter the beam from the UV laser is redirected to cure a layer according to the information given by the slicer. In order to add the third dimension, the build plate moves a distance of one layer height every time a layer is finished. This process is repeated until the final layer.
After the part is finished, it is in a still not fully cured state. To fully cure a part, further exposure to UV light is needed. This step is happening in a so-called curing station.[8]

[6] Alkaios Bournias Varotsis, *Introduction to FDM 3D printing*.
[7] Alkaios Bournias Varotsis, *What is SLA 3D printing?*
[8] Ibid.

Design Variants

SLA printers are divided into two different designs. For desktop use, printers are built with a bottom-up architecture, which means the build-plate is attached to a so-called elevator. Therefore the light source is placed below the vat containing the resin. This design allows for affordable prices by sacrificing build volume.
In industrial applications, top-down designs are used. Placing the build plate into the vat enables larger build volumes.[9]

14.3.3 Autumn Choice

In Autumn, parts are designed to be practical, which entails a simple geometry. Therefore in terms of accuracy, it wouldn't be advantageous to use an SLA printing method. Furthermore, concerning structural integrity, both parts printed with FDM and SLA printers are sufficient for the stresses parts are exposed to in the Autumn Use Case. With Autumn being focused on fast prototyping and fast iteration, SLA with its extensive curing process would be unfit with no benefits gained from using a Stereolithography style printer.

[9] Ibid.

Chapter 15

Conclusion

Author: Fabian Kleinrad

15.1 Autumn Result

15.1.1 Goal

The objective of the Autumn Project is developing a fully autonomous drone capable of generating 3D scans of inaccessible areas. Additionally, it is possible to deploy the drone in regions lacking external localization based on the technologies used. Technologies realizing these features are already available. In contrast to these commercially available solutions, the autumn project aims to accomplish these feats using a low budget approach.

15.1.2 Design choices

Crucial to the whole operation are means to perceive objects surrounding the drone. To accomplish this, a stereo camera was chosen due the to low cost compared to sensors used for commercial autonomous solutions.
The drone used in the Autumn Project provides for a wide range of possible sensor configurations and a high payload capacity. This simplified prototyping and helped with the fast iterative approach used in this project. Using a smaller drone would have impeded tests due to the increased complexity of working with compact physical space and smaller weight margins.
To enable computation on the drone itself, an NVIDIA-Jetson TX2 is being employed. This solves the issue of latency problems, which are especially troublesome in processes that need to perform without interruption to guarantee the most optimal results. In the case of the Autumn Project, such a process would be the continuous mapping of the environment.

15.1.3 Result

The Autumn Drone performs mapping using stereo images provided by the stereo camera mounted on the drone. An RRT*-based path-planning algorithm then uses these mapping results to calculate a path originating from the drone's position to an selected end-point. The current position is also provided by the SLAM algorithm in the mapping stage. Using this route information, the drone can be controlled accordingly. In the current version of the drone, only semi-autonomous flight is supported.Furthermore, due to the lack of testing possibilities and the risk that accompanies testing autonomous drones, flight capabilities were only assessed using user input.

15.1.4 Mapping

With the focus of Autumn centring around creating 3D scans of environments, the aspect of mapping is a crucial factor upon which the quality of results depends on.
Mapping was realized by implementing a 3D graph-based SLAM algorithm, supporting stereo imagery. The algorithm determines the current position of the device that provides the images. SLAM generates a point cloud representing the environment the drone explores based on this relative position. All mapping logic is computed on the drone using an NVIDIA-Jetson TX2, which provides enough computational power to ensure the most optimal results. In order to transfer the point cloud data, an access point is being hosted, over which a client may supervise the result. This enables the user to get a real-time view of the model and how it is being constructed. The resulting point-cloud can then be used as reference material in numerous applications.
An example of its utilization would be the basis of a detailed render. In this scenario, it would simplify the process of measuring and conveying the composition into modelling software. The defining structure is already present using a point cloud, and only a few adjustments have to be made.

15.1.5 Path-Planning

Autonomy implies the means to navigate through an environment without external help. In autumn, this is realized using a path-planning algorithm. This algorithm is based on the RRT* algorithm, often employed in high-dimensional and dynamic domains. The algorithm plans a path through either a two-dimensional or three-dimensional representation of the environment surrounding the drone. In autumn, this model is being generated in the mapping phase using the SLAM algorithm.
The path is computed separately from the drone. The reason is that separating the path computation onto an external device allows the drone to work more efficiently and the path-planning algorithm not to be constricted by computational restriction present when computing on the drone itself.
The route is calculated between the drone's current position and a user-defined end-point. If newer mapping data is available, the previously generated path gets checked for possible collisions with newly scanned obstacles. Due to the single-query approach of the RRT algorithm, a new path has to be calculated in the case of the path being invalid. The resulting

path can be visualized for the user alongside the model generated in the mapping phase. This allows for early error detection through a human observer.

15.2 Outlook

15.2.1 Current problems

At current times, fully autonomous flight is not possible with the Autumn Drone. A significant risk factor in the design is the inability to monitor the space above the drone. This results in problems when using three-dimensional path-planning due to the uncertainty of what lies above. Without that information, the drone is unable to utilize the third dimension, which is the most crucial factor for using a UAV. In autumn, this was solved by focusing on a semi-autonomous approach. Another problem is the short battery life-time of the drone, which is due to the size of the drone used in the project. This leads to complications when scanning medium to large environments.

15.2.2 Future solutions

The modular structure provided by ROS allows the Autumn Project to be easily implemented using different hardware. It is possible to realize fully autonomous flight using a smaller drone and a simpler two-dimensional lidar for future projects. This results in the loss of three-dimensional mapping capabilities but enables a more reliable and easier environment exploration.
Furthermore, using ROS, every part of the Autumn Logic can be used in separate projects where needed. An example would be using the two-dimensional path-planning algorithm to plan the movement of a ground vehicle.

Index

Authors Index

Literature Index

State estimation for robotics, 29
std_msgs/Header Message, 45
Structured-light 3D surface imaging: a tutorial, 23

tegrastats Utility, 6

URDF- ROS Wiki, 14

WHAT ARE POINT CLOUDS?, 44
What is FDM 3D Printing? – Simply Explained, 73
What is Lidar?, 23
What is SLA 3D printing?, 74, 75

ZED Camera and SDK Overview, 25
ZED2 Camera and SDK Overview, 26

Bibliography

Adwait Naik. *Difference between Single-Query and Multiple Query Algorithms?* en. URL: https://robotics.stackexchange.com/questions/18433/difference-between-single-query-and-multiple-query-algorithms (visited on 09/20/2021).

Alkaios Bournias Varotsis. *Introduction to FDM 3D printing*. en. URL: https://www.hubs.com/knowledge-base/introduction-fdm-3d-printing/ (visited on 11/26/2021).

— *What is SLA 3D printing?* en. URL: https://www.hubs.com/knowledge-base/introduction-sla-3d-printing/ (visited on 11/28/2021).

Altuntas, C. "TRIANGULATION AND TIME-OF-FLIGHT BASED 3D DIGITISATION TECHNIQUES OF CULTURAL HERITAGE STRUCTURES". In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 43 (2021), pp. 825–830.

Amit Patel. *Introduction to A**. en. URL: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html (visited on 09/25/2021).

Autodesk Inc. *FUSION 360 FOR PERSONAL USE*. en. URL: https://www.autodesk.com/products/fusion-360/personal (visited on 11/14/2021).

Bailey, Tim and Hugh Durrant-Whyte. "Simultaneous localization and mapping (SLAM): Part II". In: *IEEE robotics & automation magazine* 13.3 (2006), pp. 108–117.

Barfoot, Timothy D. *State estimation for robotics*. Cambridge University Press, 2017.

Bekey, George A. *Autonomous robots: from biological inspiration to implementation and control*. en. MIT press, 2005. URL: https://books.google.at/books?hl=de&lr=&id=8MbxCwAAQBAJ&oi=fnd&pg=PR7&dq=autonomous+robots&ots=5EXAZx-UDf&sig=u_CyTA0fa4aMRFAAdnbXpKhaf94&redir_esc=y#v=onepage&q=autonomous%20robots&f=false (visited on 06/30/2021).

Bekris, Kostas E., Brian Y. Chen, Andrew M. Ladd, Erion Plaku, and Lydia E. Kavraki. "Multiple Query Probabilistic Roadmap Planning using Single Query Planning Primitives". In: *IEEE International Conference on Intelligent Robots and Systems* 1.March 2018 (2003), pp. 656–661. DOI: 10.1109/iros.2003.1250704.

Ben-Ari, Mordechai and Francesco Mondada. *Elements of robotics*. Springer Nature, 2017.

Cadena, Cesar, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332. DOI: 10.1109/TRO.2016.2624754. (Visited on 09/26/2021).

Conselho Nacional de Desenvolvimento Cientifico e Tecnologico. “O CNPq e a formacao de recursos humanos de C&T para o Brasil : estatisticas de bolsas no pais e no exterior, 1980-95”. In: (1995), 113 p.

Cyrill Stachniss. *Occupancy grid maps*. en. URL: http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam11-gridmaps-4.pdf (visited on 01/04/2022).

D.W. *A* graph search time-complexity*. en. URL: https://cs.stackexchange.com/questions/56176/a-graph-search-time-complexity (visited on 10/21/2021).

Do, Phuong Ngoc Binh and Quoc Chi Nguyen. “A review of stereo-photogrammetry method for 3-D reconstruction in computer vision”. In: *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*. IEEE. 2019, pp. 138–143.

DruckWege GmbH. *Depiction of FDM printer parts, in an Cartesian-style design.* en. URL: https://druckwege.de/home/technologie/fused-deposition-modelling-fdm (visited on 11/27/2021).

Durrant-Whyte, H. and T. Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022. (Visited on 09/26/2021).

Emesent. *Emesent_Hovermap.JPG (JPEG Image, 2018 × 910 pixels)*. URL: https://www.emesent.io/wp-content/uploads/2020/07/DSC_3700-1finalcrop.jpg?id=7339 (visited on 07/04/2021).

Emesent Hovermap. *Autonomy Level 2 for Emesent Hovermap*. URL: https://www.emesent.io/autonomy-level-2/ (visited on 07/04/2021).

Emmett Grames. *What is FDM 3D Printing? – Simply Explained*. en. URL: https://all3dp.com/2/fused-deposition-modeling-fdm-3d-printing-simply-explained/ (visited on 11/26/2021).

Exyn Aero. *Exyn Aero - Aerial Mapping Drone*. URL: https://www.exyn.com/products/exyn-aero-aerial-mapping-drone (visited on 07/04/2021).

Garrido-Jurado, Sergio, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Rafael Medina-Carnicer. “Generation of fiducial marker dictionaries using Mixed Integer Linear Programming”. In: *Pattern Recognition* 51 (Oct. 2015). DOI: 10.1016/j.patcog.2015.09.023.

Geng, Jason. “Structured-light 3D surface imaging: a tutorial”. In: *Adv. Opt. Photon.* 3.2 (2011), pp. 128–160. DOI: 10.1364/AOP.3.000128. URL: http://www.osapublishing.org/aop/abstract.cfm?URI=aop-3-2-128.

Gerkey, Brian. *Definition - ROS Answers*. en. URL: https://answers.ros.org/question/12230/what-is-ros-exactly-middleware-framework-operating-system/ (visited on 07/08/2021).

Gokturk, S Burak, Hakan Yalcin, and Cyrus Bamji. “A time-of-flight depth sensor-system description, issues and solutions”. In: *2004 conference on computer vision and pattern recognition workshop*. IEEE. 2004, pp. 35–35.

Grisetti, Giorgio, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. “A tutorial on graph-based SLAM”. In: *IEEE Transactions on Intelligent Transportation Systems Magazine* 2 (Dec. 2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.

Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard. “Improving Grid-based SLAM with Rao-Blackwellized Particle Filters By Adaptive Proposals and Selective Resampling”. In: Jan. 2005, pp. 2432–2437. DOI: 10.1109/ROBOT.2005.1570477.

Gutmann, Steffen and Kurt Konolige. "Incremental Mapping of Large Cyclic Environments". In: (Nov. 2003). DOI: 10.1109/CIRA.1999.810068.

Haenelt, Karin. *Hidden Markov Models (HMM)*. de. 2006. URL: https://www.cis.lmu.de/~micha/kurse/statistik-SS2008/begleitmaterial/HMM.pdf (visited on 10/09/2021).

Horn, Berthold, Hugh Hilden, and Shahriar Negahdaripour. "Closed-Form Solution of Absolute Orientation using Orthonormal Matrices". In: *Journal of the Optical Society of America A* 5 (July 1988), pp. 1127–1135. DOI: 10.1364/JOSAA.5.001127.

Islam, Fahad, Jauwairia Nasir, Usman Malik, Yasar Ayaz, and Osman Hasan. "RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution". In: *2012 IEEE International Conference on Mechatronics and Automation, ICMA 2012* (2012), pp. 1651–1656. DOI: 10.1109/ICMA.2012.6284384. URL: http://save.seecs.nust.edu.pk/pubs/ICMA2012.pdf.

Jacob Ames. *SolidWorks user interface broken up into its different components*. en. URL: https://hawkridgesys.com/blog/user-interface-basics-in-solidworks (visited on 11/14/2021).

Jan Faigl. *Robotic Paradigms and Control Architectures*. en. 2017. URL: https://cw.fel.cvut.cz/old/_media/courses/b4m36uir/lectures/b4m36uir-lec02-slides.pdf (visited on 03/08/2022).

Jia, Songmin, Zeling Zheng, Guoliang Zhang, Jinhui Fan, Xiuzhi Li, Xiangyin Zhang, and Mingai Li. "An Improved RANSAC Algorithm for Simultaneous Localization and Mapping". In: 1069 (2018), p. 012170. DOI: 10.1088/1742-6596/1069/1/012170. URL: https://doi.org/10.1088/1742-6596/1069/1/012170.

Karaman, Sertac and Emilio Frazzoli. "Incremental sampling-based algorithms for optimal motion planning". In: *Robotics: Science and Systems* 6 (2011), pp. 267–274. ISSN: 2330765X. DOI: 10.15607/rss.2010.vi.034. arXiv: 1005.0416.

Korovko, Alexander and Dmitry Robustov. "Partial Hierarchical Pose Graph Optimization for SLAM". In: *arXiv preprint arXiv:2110.08639* (2021).

Labbé, Mathieu and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation". In: *Journal of Field Robotics* 36.2 (2019), pp. 416–446.

Lucas Carolo. *SolidWorks 2022 vs Fusion 360: The Differences*. en. URL: https://all3dp.com/2/fusion-360-vs-solidworks-cad-software-compared-side-by-side/ (visited on 11/14/2021).

Malone, Bob. *George Devol: A Life Devoted to Invention, and Robots*. en. 2011. URL: https://spectrum.ieee.org/automaton/robotics/industrial-robots/george-devol-a-life-devoted-to-invention-and-robots (visited on 06/17/2021).

Montemerlo, Michael, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Aaai/iaai* 593598 (2002).

Naderi, Kourosh, Joose Rajamaki, and Perttu Hamalainen. "RT-RRT: A real-time path planning algorithm based on RRT". In: *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games, MIG 2015* July 2019 (2015), pp. 113–118. DOI: 10.1145/2822013.2822036.

Ng, Cherlyn J. and Bart Farell. "Solving the stereo correspondence problem with false matches". In: *PLOS ONE* 14.7 (July 2019), pp. 1–23. DOI: 10.1371/journal.pone.0219052. URL: https://doi.org/10.1371/journal.pone.0219052.

NVIDIA Corporation. *NVIDIA Jetson Hardware Page*. en. URL: https://developer.nvidia.com/embedded/jetson-modules (visited on 09/19/2021).

— *Power Management for Jetson TX2 Series Devices - Supported Modes and Power Efficiency*. en. URL: https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_tx2_32.html (visited on 09/19/2021).

— *tegrastats Utility*. en. URL: https://docs.nvidia.com/jetson/archives/l4t-archived/l%E2%80%A6 (visited on 09/19/2021).

Open Source Robotics Foundation. *Building Packages - ROS Wiki*. en. URL: http://wiki.ros.org/ROS/Tutorials/BuildingPackages (visited on 07/08/2021).

— *Concepts - ROS Wiki*. en. URL: http://wiki.ros.org/ROS/Concepts (visited on 07/08/2021).

— *Gazebo - Overview*. en. URL: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1 (visited on 02/22/2022).

— *geometry_msgs/Point32*. en. URL: http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Point32.html (visited on 01/06/2022).

— *Master - ROS Wiki*. en. URL: http://wiki.ros.org/Master (visited on 07/22/2021).

— *Messages - ROS Wiki*. en. URL: http://wiki.ros.org/Messages (visited on 07/09/2021).

— *Metapackages - ROS Wiki*. en. URL: http://wiki.ros.org/Metapackages (visited on 07/08/2021).

— *nav_msgs/OccupancyGrid Message*. en. URL: http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/OccupancyGrid.html (visited on 01/06/2022).

— *Nodes - ROS Wiki*. en. URL: http://wiki.ros.org/Nodes (visited on 07/09/2021).

— *Packages - ROS Wiki*. en. URL: http://wiki.ros.org/Packages (visited on 07/08/2021).

— *Parameter Server - ROS Wiki*. en. URL: http://wiki.ros.org/Parameter%20Server (visited on 07/22/2021).

— *sensor_msgs/PointCloud*. en. URL: http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/PointCloud.html (visited on 01/06/2022).

— *Services - ROS Wiki*. en. URL: http://wiki.ros.org/Services (visited on 07/09/2021).

— *std_msgs/Header Message*. en. URL: http://docs.ros.org/en/noetic/api/std_msgs/html/msg/Header.html (visited on 01/06/2022).

— *URDF- ROS Wiki*. en. URL: http://wiki.ros.org/urdf/Tutorials (visited on 07/28/2021).

Pan, Tong and Shuk Ching Pun-Cheng. "A Discussion on the Evolution of the Pathfinding Algorithms". In: August (2020), pp. 1–20. DOI: 10.20944/preprints202008.0627.v1. URL: www.preprints.org.

Sathyaraj, B. Moses, L. C. Jain, A. Finn, and S. Drake. "Multiple UAVs path planning algorithms: A comparative study". In: *Fuzzy Optimization and Decision Making* 7.3 (2008), pp. 257–267. ISSN: 15684539. DOI: 10.1007/s10700-008-9035-0.

Schmitt, Betina Madeira, Christiano Fraga Zirbes, Cassiano Bonin, Daniel Lohmann, Diovani Castoldi Lencina, and Aurélio Da Costa Sabino Netto. "A comparative study of cartesian

and delta 3d printers on producing PLA parts". In: *Materials Research* 20 (2017), pp. 883–886. ISSN: 15161439. DOI: 10.1590/1980-5373-mr-2016-1039.

Siciliano, Bruno, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*. Vol. 200. Springer, 2008.

Srinivasan, Raj. *RPC: Remote Procedure Call Protocol Specification Version 2*. RFC 1831. Aug. 1995. DOI: 10.17487/RFC1831. URL: https://rfc-editor.org/rfc/rfc1831.txt.

Stachniss, Cyrill, John J Leonard, and Sebastian Thrun. "Simultaneous localization and mapping". In: *Springer Handbook of Robotics*. Springer, 2016, pp. 1153–1176.

Stephens, Rod. *Beginning Software Engineering*. en. Wiley, 2015. ISBN: 9781118969168. URL: https://books.google.at/books?id=SyHWBgAAQBAJ (visited on 07/09/2021).

Stereolabs Inc. *ZED Camera and SDK Overview*. en. URL: https://www.stereolabs.com/assets/datasheets/zed-camera-datasheet.pdf.

— *ZED2 Camera and SDK Overview*. en. URL: https://www.mybotshop.de/Datasheet/zed2-camera-datasheet.pdf.

Szudzik, Matthew P. "The Rosenberg-Strong Pairing Function". In: (2017), pp. 1–27. arXiv: 1706.04129. URL: http://arxiv.org/abs/1706.04129.

TECH27 SYSTEMS. *WHAT ARE POINT CLOUDS?* en. URL: https://tech27.com/resources/point-clouds/ (visited on 01/05/2022).

The Graphviz Authors. *About - Graphviz*. en. URL: https://graphviz.org/about/ (visited on 07/28/2021).

Thrun, Sebastian. "Learning metric-topological maps for indoor mobile robot navigation". In: *Artificial Intelligence* 99.1 (1998), pp. 21–71.

— *Probabilistic robotics*. Vol. 45. 3. ACM New York, NY, USA, 2002.

Velodyne Lidar, Inc. *What is Lidar?* en. URL: https://velodynelidar.com/what-is-lidar/ (visited on 12/31/2021).

Wang, Chieh-Chih, C. Thorpe, and S. Thrun. "Online simultaneous localization and mapping with detection and tracking of moving objects: theory and results from a ground vehicle in crowded urban areas". In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 1. 2003, 842–849 vol.1. DOI: 10.1109/ROBOT.2003.1241698.

Wolfram Burgard. *Introduction to Mobile Robotics - Robot Control Paradigms*. en. 2020. URL: http://ais.informatik.uni-freiburg.de/teaching/ws17/mapping/pdf/slam03-bayes-filter-short.pdf.

Zammit, Christian and Erik Jan van Kampen. "Comparison between A* and RRT algorithms for UAV path planning". In: *AIAA Guidance, Navigation, and Control Conference, 2018* 210039 (2018). DOI: 10.2514/6.2018-1846.

Zingl, Alois. "A Rasterizing Algorithm for Drawing Curves". In: (2012), p. 98.

Chapter 16

Appendix

Meetingprotokoll

Autumn

Lukas Leskovar & Fabian Kleinrad

Oktober 2021

Abstract

Die Liste an Meetings für die Autumn Diplomarbeit und Projekt.

Contents

2.4.2021

- Entscheidung Thema
- Recherche
- Organisation

7.5.2021

- Use-Cases (Tunnel, Brücken, Gebäude)
- Ziel festlegen (Autonomes Mapping)

14.5.2021

- Leskovar: Jetson Nano konfigurieren, ROS
- Kleinrad: Management Tool

28.5.2021

- Leskovar: Kamera/LiDAR testen + Mapping Algorithmen
- Kleinrad: Algorithmus für Pfadfindung auswählen

10.6.2021

- Leskovar: Remote Mapping auf Jetson Nano testen
- Kleinrad: Umsetzung eines Occupancy grid basierenden RRT Algorithmus

30.7.2021

- Leskovar: Jetson TX2 konfigurieren
- Kleinrad: Fehlerbehebung der Umsetzung

20.8.2021

- Leskovar: Remote Mapping auf TX2
- Kleinrad: Implementierung von RRT* auf basis des RRT

10.9.2021

- Diplomschrift: Leskovar Introduction & ROS
- Kleinrad: Problembehebung der RRT* Implementierung

24.9.2021

- Leskovar: Remote Mapping auf TX2
- Kleinrad: Collision Avoidance in der 2D RRT* Implementierung

1.10.2021

- Leskovar: Zed 2i zur verbesserung
- Kleinrad: Refactor 2D Algorithmus um für 3D Umsetzung

15.10.2021

- Leskovar: Performance Probleme lösen
- Kleinrad: Implementierung des Algorithmus auf basis einer Point-Cloud

22.10.2021

- Diplomschrift: System Architecture (Leskovar) + Pathplanning (Kleinrad)

29.10.2021

- Leskovar: Aufteilung Mapping (auf Drohne) & RVIZ (auf Computer)
- Kleinrad: Collision Avoidance basierend auf Point-Cloud

12.11.2021

- Leskovar: Aufteilung Mapping (auf Drohne) & RVIZ (auf Computer)
- Kleinrad: Aufteilung der Algorithmen in selbständige ROS Packages

25.11.2021

- Leskovar: Remote Verbindung (Hotspot von Jetson hosten) + testen/optimieren
- Kleinrad: Testen der 2D & 3D Algortihmen
- Diplomschrift: SLAM (Leskovar) + Custom Parts (Kleinrad)

3.12.2021

- Leskovar: Remote Verbindung (Hotspot von Jetson hosten) + testen/optimieren
- Kleinrad: Refactor/Fehlerbehebung 2D Algorithmus

10.12.2021

- Leskovar: TX2 + Drohne verbinden
- Kleinrad: Fehlerbehebung 2D Algorithmus

14.12.2021

- Diplomschrift: Representation of Environments (Kleinrad)

21.1.2021

- Leskovar: Experiment Mapping
- Kleinrad: Refactor/Fehlerbehebung 3D Algorithmus

11.2.2022

- Leskovar: ZED Experiment
- Kleinrad: RRT* Collision Avoidance Tests

24.2.2022

- Diplomschrift: Sensors (Leskovar) + Collision Avoidance (Kleinrad)

4.3.2022

- Leskovar: ZED Experiment
- Kleinrad: RRT* Genauigkeits/Laufzeit Tests

27.3.2022

- Diplomschrift: Abstract (Kleinrad) + RRT* Tests (Kleinrad) + Conclusion (Kleinrad) + Autonomous Navigation (Leskovar)

Arbeitsprotokoll

Autumn

Lukas Leskovar

Oktober 2021

Abstract

Die Liste an Arbeitsstunden für das Autumn Diplomarbeitsprojekt von Lukas Leskovar.

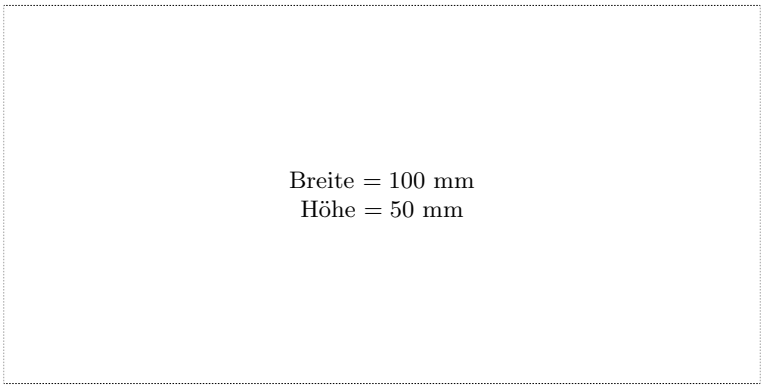
Contents

Datum	Dauer[h]	Aufgabe	Anmerkung
18.05.2021	4	Projekterstellung	
23.05.2021	5	Packages inkuldieren	
24.05.2021	4	Test ZED 1 maping	
25.05.2021	2	Netzerkerbndung kabelgebunden	
25.05.2021	4	ROS Network communication	
26.05.2021	5	Zed Topic throttle	
28.05.2021	4	Topic cleanup	
27.06.2021	4	Fix remote mapping	
28.07.2021	3	remove topic throttles	
29.07.2021	5	configure Jetson TX2	
30.08.2021	4	implement mapping on drone	
31.08.2021	5	implement mapping on drone	
06.09.2021	6	implement remote client	
07.09.2021	4	implement remote client	
10.09.2021	4	refactor client	
13.09.2021	4	refactor drone	Projektstunde
14.09.2021	4	test zed2i	Projektstunde
16.09.2021	3	zed2i parameter tuning	
17.09.2021	4	zed2i parameter tuning	
18.09.2021	6	configure zed2i mapping on laptop	
20.09.2021	6	configure TX2 for new camera	
21.09.2021	4	change camera topics to zed 2i	Projektstunde
22.09.2021	6	implement visual/inertial odometry	
23.09.2021	4	implement visual/inertial odometry	
24.09.2021	6	tune odometry	
28.09.2021	4	tune mapping	Projektstunde
05.10.2021	4	cleanup	Projektstunde
12.10.2021	4	tune mapping	Projektstunde
19.10.2021	4	tune mapping	Projektstunde
09.11.2021	4	host network on drone	Projektstunde
23.11.2021	4	host network on drone	Projektstunde
30.11.2021	4	host network on drone	Projektstunde
07.12.2021	4	host network on drone	Projektstunde
09.12.2021	5	ROS Network communication	
10.12.2021	5	ROS Network communication	
12.12.2021	6	ROS Network communication	

Datum	**Dauer[h]**	**Aufgabe**	**Anmerkung**
14.12.2021	4	connect TX2 to drone	Projektstunde
21.12.2021	4	connect TX2 to drone	Projektstunde
11.01.2022	4	connect TX2 to drone	Projektstunde
18.01.2022	2	connect TX2 to drone	Projektstunde
21.01.2022	2	connect TX2 to drone	Projektstunde
25.01.2022	2	read drone data topics	Projektstunde
28.01.2022	2	read drone data topics	Projektstunde
29.01.2022	2	experiment mapping	
30.01.2022	3	experiment mapping	
01.02.2022	4	experiment mapping	
04.02.2022	2	improve mapping	Projektstunde
21.02.2022	2	improve mapping	Projektstunde
25.02.2022	2	improve mapping	Projektstunde
01.03.2022	2	improve mapping	Projektstunde
04.03.2022	2	experiment zed1 vs zed2i	Projektstunde
08.03.2022	2	experiment zed1 vs zed2i	Projektstunde
11.03.2022	2	experiment zed1 vs zed2i	Projektstunde
15.03.2022	2	experiment zed1 vs zed2i	Projektstunde
18.03.2022	2	experiment zed1 vs zed2i	Projektstunde
22.03.2022	2	Video über Projekt	Projektstunde
29.03.2022	2	Video über Projekt	Projektstunde
Summe	**209**		

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —