HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

Autonomous universal Mapping and Navigation

Ausgeführt im Schuljahr 2020/21 von:

Themengebiet 1
Lukas Leskovar
5BHIF

Themengebiet 2
Fabian Kleinrad
5BHIF

Betreuer / Betreuerin:

MMag. Dr. Michael Stifter

Wiener Neustadt, am October 3, 2020/21

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 October 3, 2020/21

Verfasser / Verfasserinnen:

Lukas Leskovar

Fabian Kleinrad

Contents

Chapter 2

Acknowledgement

The authors would like to thank ...

Chapter 3

Kurzfassung

asdf

Chapter 4

Abstract

asdf

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, 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 not only allow for much more accurate procedures but also 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 at home to teach children topics in a playful and interesting way.

[1] **malone2011unimate**

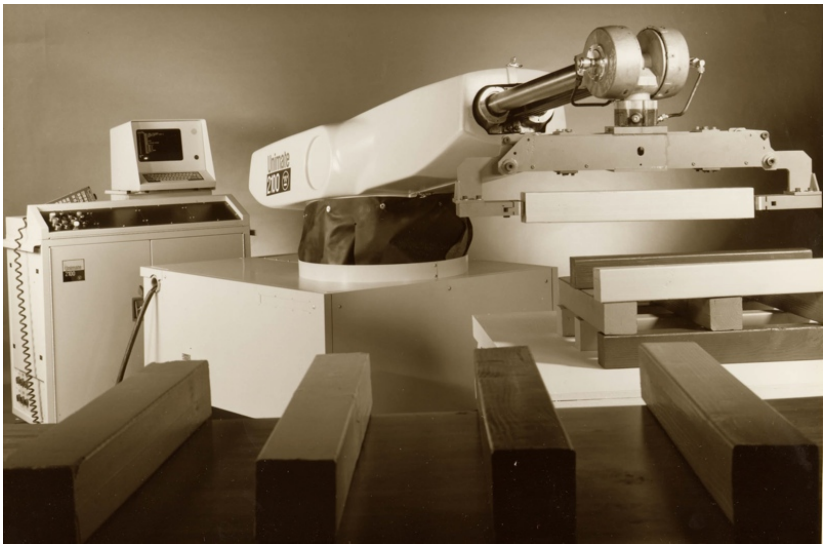


Figure 5.1: Picture of the first industrial robot. The Unimate developed by George Devol and Joseph Engelbert in 1961 was first used for hot die-casting and welding applications.[1]

5.3 Robots in hazardous environments

Robots do not only serve a purpose in a close-to-user work environment, they also ensure human safety by performing dangerous tasks in unsafe surroundings. Remotely controlled robots or drones can be used for inspecting 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 be used to navigate and organize warehouses, constructing parts in an assembly line, or map large areas for comprehensive calculations.

[2] **bekey2005autonomous**.

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

5.5 3D Mapping

5.6 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 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 Fig. 5.2.

5.7 Goals

The project associated with this thesis aims to implement a 3D Mapping system similar to the aforementioned solutions with limited financial, personnel as well as temporal resources. This goal forces the project team to primarily maintain an open-source approach during development.

The goal of this thesis is 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.

[3] **hovermap2021**.
[4] **exynAero2021**.
[5] **hovermapIMG2021**

5.8 Requirements

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

- The system is capable of generating a 3D Point-cloud of its surroundings
- All required hardware (e.g. sensors, computing boards, etc.) is mounted onto a drone-platform
- All calculations concerning the map-generation are run on a external server in direct communication with the drone
- The drone utilizes the Point-cloud to orientate in its surroundings and navigate one or multiple waypoints
- The drone is capable of avoiding obstacles as it is moving through a unknown environment

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

- A Web-App facilitating the usage of the system and enabling the user to create waypoints, monitor the drone and mapping algorithm as well as evaluate the Point-cloud
- The system is replicated within the gazebo simulator to simplify future development without direct access to its hardware

Chapter 6

Study of Literature

Author:

Chapter 7

System Architecture

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

7.1 First Prototype

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

- A DJI Matrice 100 functioning as the base of the system 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 to be improved where as the main issues were the lack of computational power and usage of non optimal hardware. A detailed explanation as well as a multitude of solutions to this problem are discussed in the following sections.

7.2 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 a major concern in many robotic applications that are powered by batteries or are restricted to low power inputs. Since Autumn is powered only by a drone battery this issue is groundbreaking during the development of this diploma thesis.

The central component of the system is a 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 9) 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 drones battery but also impairs the quality of the resulting 3D map as well as operating the drone.

7.3 Solving processing limitations

When dealing with load heavy computations one way to solve quality issues is to use higher performance hardware, however due to aforementioned power constraints and drone payload requirements this approach is not suitable for this diploma thesis. Another possible solution is to lower the computational load of the processors 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 lowered the amount of computation on the drone but required to use a high performance computer on site to wirelessly communicate with the drone. Furthermore the wifi-range became another limiting factor since with increasing range the latency increased thus impairing the result quality again.

- The second approach was to disable visual odometry using feature extraction and pattern matching as the most complex part of the algorithm and providing 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 **??**

7.4 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 as a great 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 greatly reduced the amount of computations performed on the NVIDIA Jetson TX2.

[1] **jetsonHardwarePageNoDate**.
[2] **nvidiaTegrastatsNoDate**.
[3] **jetsonPowerModesNoDate**.

- 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.

Figure 7.1: The Autumn drone with the NVIDIA Jetson TX2 on top as well as the Stereolabs ZED 2i mounted onto the drones Gimbal mounting plate using a custom 3D printed frame.

Chapter 8

Robot Operating System

Author: Lukas Leskovar
This chapters objective is to describe the basic concepts of the Robot Operating System (ROS) utilized 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 utilizing publisher-subscriber communication between different programs of such systems. Furthermore ROS provides a comprehensive set of tools enabling the compilation, operation as well as testing, visualization and debugging of robotic systems. With its vast amount of libraries and huge open-source community providing useful functionality ROS facilitates the development of robotic applications without having to reimplement standardized technology.[1]

8.1 Conceptual Overview

The Robot Operating System can be divided into three conceptual levels each contributing a 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 utilized within the graph by publishing and subscribing to topics. The concepts and technologies powering the computational graph are described in later in this chapter.

[1] **openSourceRoboticsFoundationDefinitionNodate**.
[2] **openSourceRoboticsFoundationConceptsNodate**.

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.

8.2 Naming

To aid the organization of programs, processes as well as resources ROS provides two naming schemes that are described in the following sections.[3]

Graph Resource Names

Graph Resource Names utilize a hierarchical structure to organize 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 packages name as well as the path to the desired resource within the package.

8.3 Packages

Software in ROS is organized 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 a slim as possible my 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 simply refer to directories. While most subfolders and files within a package depend on its 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] **openSourceRoboticsFoundationConceptsNodate**.
[4] **openSourceRoboticsFoundationPackageNodate**.
[5] **openSourceRoboticsFoundationBuildNodate**.
[6] **openSourceRoboticsFoundationMetapackageNodate**.

8.4 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 utilizing ROS consists of a fine-grained graph of processes called nodes. Each node provides computation on a single feature utilizing a ROS client library to communicate with others over a mesh-like peer-to-peer network.[7]
Exemplary for such as system would be one node running a LiDAR sensor, one responsible for localization, one performing motion planning, one controlling motor drivers and motors as well as one node running the robots main control loop.
This architecture allows for a 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 larger program.
Each node has a node type consisting of the package name it is located and as well as the nodes executable.

8.5 Communication

8.5.1 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 packages name as well as the name of the messages *.msg* file.

8.5.2 Topics

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

[7] **openSourceRoboticsFoundationNodesNodate**.
[8] **stephensBeginning2015**.
[9] **openSourceRoboticsFoundationMessagesNodate**.

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

8.5.3 Services

The communication architecture in ROS utilizing 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 procedures 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]

8.6 Master

One of the most important components of ROS is the Master. It tracks publishers and subscribers of topics as well as services and provides registration as well as 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 leat one subscriber and publisher the Master negotiates between the nodes so a peer-to-peer connection can be established using a Slave API provided by the nodes XML-RPC Server.[12] A simplified version of this procedure can be seen in Fig. 8.1
Besides registration and name resolution, the ROS Master also provides a Parameter Server used for globally storing static system parameters.[13]

8.7 Transform Library

A complex robotic system consists of multiple parts such as sensors, cameras, manipulators, etc. which are 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, not only the transform of that single frame but the composite transform of each frame in relation to the target has 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. Using the ROS Transform Library (tf) these complex calculations can be facilitated. To this end tf keeps track of each coordinate frame in a acyclic relationship tree where tf broadcasters then publish relative pose information and listeners query transforms between two coordinate frames. Because not all pose information in a robotic system is

[10] **rfc1831**.
[11] **openSourceRoboticsFoundationServicesNodate**.
[12] **openSourceRoboticsFoundationMasterNodate**.
[13] **openSourceRoboticsFoundationParameterServerNodate**.

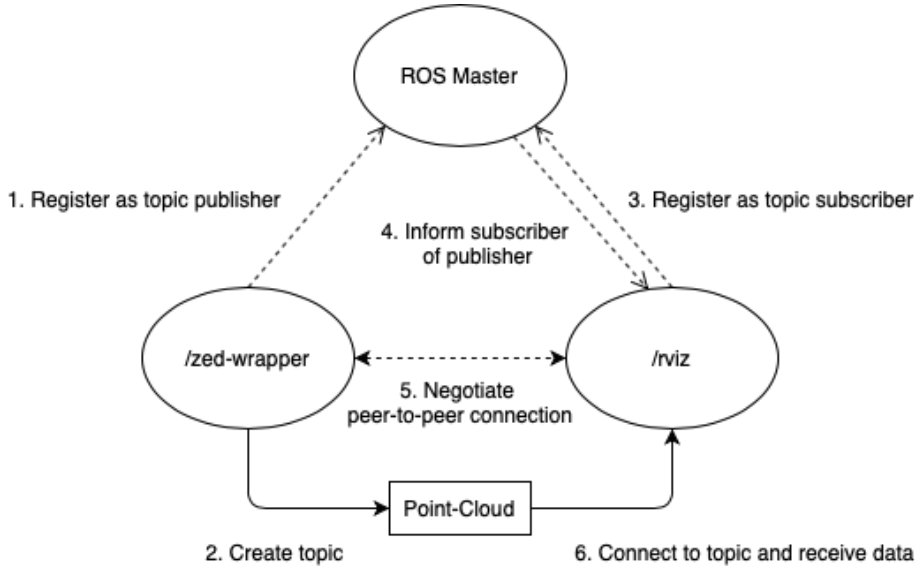


Figure 8.1: Diagram of the topic registration process in ROS at which the publishing node registers its topic before the subscriber tells the master its interest in the point-cloud topic. However a node can be registered as a subscriber to a specific topic without the topic existing yet.

instantly accessible, the tf saves this information for each frame over time. This means that transforms can be queried not just spacially but also temporally.

8.8 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 the Autumn robot utilises the representation and simulation technologies that are described in the following sections.

8.8.1 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 a XML format for representing a robot and its components as well as a *C++* parser and tools to convert and verify and visualize these models.[14] The *check_urdf* tool parses a URDF-File and returns the robots kinematic chain if successful. To visualize the robots frames and joints in a graphviz[15] tree the *urdf_to_graphiz* tool can be used. The resulting tree corresponds to the relationship tree tf uses to calculate transforms.

[14] **openSourceRoboticsFoundationURDFNodate**.
[15] **graphvizAuthorsAboutNodate**.

8.8.2 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 as well as comprehensive interfaces for controlling a simulated robot through ROS. Further advantages of Gazebo are its accurate sensor and sensor noise generation as well as its large community providing countless models of robots and sensors.

Chapter 9

Simultaneous Localization and Mapping

Author: Lukas Leskovar

The problem of localizing as well as navigating a system through a completely or partly unknown environment without any external coordinate system (i.e. GPS, Optical Beacon Tracking, etc.) has proven itself to be one of the most complex and yet fundamental topics in many scientific research fields with robotics being most prominent. The main 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 as well as mapping of such environment, different modern SLAM approaches and solutions utilized by the project team are discussed in this chapter.

9.1 Localization

Localization or state-Estimation aims to reconstruct the state of a system using interoceptive measurements (e.g. acceleration, velocity, etc.) as well as an exteroceptive model (e.g. position and orientation) of the system.[3] Put into the context of mobile robotics this means that in order to perform comprehensive localization of a mobile robot a sensor fusion between on-board sensors and an external coordinate system or map ought to be performed as solely relying on incremental sensors for odometry would quickly result in large accumulated errors. Exemplary for such a system would be an industrial robot utilizing a construction plan to

[1] **durrantSlam2006**.
[2] **cadenaSlamFuture2016**.
[3] **barfootStateEstimation2017**.

correct errors by wheel-encoded odometry as well as to navigate through a factory.

9.2 Mapping

Contrary to localization and state-estimation, 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 of environmental features as well as the systems pose to reconstruct aforementioned features in a global reference frame. Mapping applications often combine technologies typically found in scientific fields such as photogrammetry, computer vision or robotics. Another example would be a drone computing camera images and GPS positional data with a photogrammetric algorithm to reconstruct a 3D-Model of a building.

9.3 Localization and Mapping

The aforementioned technologies are considered rather simplistic problems as either the environments map is given or reliable pose-estimation can be provided. While most applications meet either of these criteria with pre-built maps or GPS being available, in some environments such as indoors, mineshafts or outer space both the systems pose and environment remain uncertain and need to be determined simultaneously hence simultaneous localization and mapping. To summarize, the crux of the SLAM problem is that localizations requires a map and mapping depends on pose estimates however neither are certain.
To approach this problem in a structured way a SLAM system usually comprises two main components:

- A front-end in charge of abstracting sensory input to be used in the back-end by performing feature extraction as well as data association.
- A back-end performing probabilistic estimation based on the abstracted data fed in by the front-end. The back-end also provides information to the front-end supporting loop closure detection.

The following sections will mainly focus on the SLAM back-end and dissect the problem by reviewing different probabilistic approaches.

9.4 Data Association

While many aspects of the front-end remain application specific each SLAM system typically contains modules for data association in the following ways. Short-term data association or feature tracking is responsible for associating corresponding features over multiple measurements while long-term data association aims to detect and link similarities between current measurements and previously observed features. These loops help correcting accumulated odometry errors as well as optimizing the global topology. A loop closure typically event occurs when revisiting previously mapped landmarks.

9.5 Map Representation

The way a robot perceives its environment and maintains a accurate map of its environment is directly dependant on many criteria such as complexity of tasks, size of its environment as well as measurement quality mainly influenced by sensor noise. Tailoring to benefit some of the aforementioned criteria 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 robots environment as occupancy grids, with each grid cell indicating the presence of an obstacle. The main benefit using metric maps is the facilitated construction of large-scale mappings as well as non-ambiguous determination of places. However such maps have significant drawbacks concerning space as well as 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 robots sensors. In contrast to metric maps they allow for comprehensive path planning and are significantly more compact as its resolution is directly proportional to the environments complexity.[4]

9.6 Problem Definition

9.7 Solution Paradigms

9.7.1 Kalman Filter

9.7.2 Particle Filter

9.7.3 Graph-based

9.8 Visual SLAM

9.9 Loop Closure for Visual SLAM

9.9.1 Appearance-based

9.9.2 Deep Learning

9.10 Map optimization

9.10.1 Bundle Adjustment

[4] **thrunMaps1998**.

Chapter 10

Methodology

Author:

Chapter 11

Implementation

Author:

Chapter 12

Experiment 1

Author:

Chapter 13

Lessons learned

Author:

Chapter 14

Experiment 2

Author:

Chapter 15

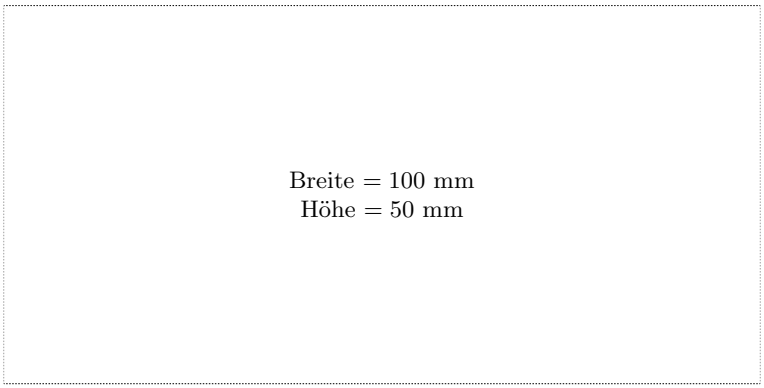
Conclusion

Author:

Index

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —