Konstantin Lassnig, BSc

An Autonomous Robot for Campus-Wide Transport Tasks

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Gerald Steinbauer

Institute for Software Technology

Graz, September 2016

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Graz, ________________ ____________________________

Date Signature

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, ________________ ____________________________

Datum Unterschrift

Acknowledgements

First I would like to thank my supervisor Gerald Steinbauer, who always found the time for discussing ideas and methodologies. I hold him in high regard because he spend a lot of time for supporting me during the time of my master thesis, even in stressful periods.

Special thanks to Clemens Mühlbacher, he invested a lot of time by supporting me not only with many helpful contributions but also by assisting me when assembling or transporting the robot. Moreover I am thanking the group of the *Autonomous Intelligent System* laboratory of the *Institute of Software Technology* for their support. Furthermore I have to thank the company *incubedIT* for funding this thesis.

Many thanks to the whole *Tedusar* Team for their great support. We made such a good job in bringing ideas to life and solving challenging problems together. We worked really hard, but made sure we had fun doing it!

I like to thank my girlfriend for always supporting me throughout my whole life. She reminds me to enjoy life, either by doing sports or by drinking wine together and have a good talk. Finally I want to thank my whole family as well as all my friends for believing and supporting me and my ideas.

Konstantin Lassnig
Graz, Austria, August 2016

Abstract

In this thesis we present a navigation system for an autonomous transport robot. The system is able to operate in outdoor as well as indoor environments. The navigation system is specially designed for urban areas, such as campus sites which raise numerous challenges. The system allows reacting autonomously on dynamic changes of the environment such as blocked sideways and roads while trying to stay on preferred paths like sidewalks. In this thesis we present the core modules of the system: (1) mapping, (2) localization and (3) path planning and navigation. Techniques developed within this thesis, contribute to the area of large-scale graph-based mapping by fusing information of several sensors to one efficient and consistent map representation. Moreover, algorithms which translate this representation into a hybrid representation comprising a roadmap and local grid maps are presented. The topological roadmap allows easy access to the underlying local grid maps. Thus is a practical solution to treat large-scale maps. Moreover existing technologies for solving local localization or path planning and navigation can be reused. The planning architecture is based on separated abstraction levels and comprises a high-level module which is able to generate paths in real-time for large outdoor environments. The high-level planner uses the roadmap with a graph topology to take the global structure of the environment into account. Theses global plans are refined by a local planner using the connected local grid maps. Furthermore the planner also incorporates the traversability and type of the actual terrain into planning. The proposed navigation system was implemented and realized with a mobile robot prototype to allow extensive testing and evaluation in a real world scenario. The evaluation verified the technical feasibility and functionality of the proposed algorithms as well as the implementation.

Contents

List of Figures

1 Introduction

This thesis presents the design and implementation of a fully autonomous transport system that can be used indoor and outdoor.

With increasing need for flexibility and automation fully autonomous mobile robots are increasingly deployed in industrial faculties and demonstrate their usefulness in a variety of different tasks. This new kind of robots do not require any kind of modification to the environment, like magnets in the floor or painted lines. Another big improvement is the new gained ability to work side by side with humans, without cages or restrictions. This offers hole new perspectives. Nowadays the transportation task is more often transferred to robots due to flexibility reasons. The clear majority of mobile robots are designed for indoor scenarios only but with rising popularity and acceptance by the industry robots begin to expand their working fields. As a consequence outdoor robots are becoming essential for short and intermediate transport services.

Autonomous driving is a new upcoming field in the automotive industry and will have profound effects on our daily life and society. Mobility and the way we travel is transforming whenever new technologies come available. This technology offers a lot of benefits in terms of safety and will prevent a lot of human caused accidents. In the next subsequent years such systems will first appear on highways due to complexity reasons but the goal for the future is clear, driving wherever a human is able to drive. These highly automated cars have the same underlying technology as robots and rely on algorithms earlier investigated in the field of robotics. Prospective developments in this industry will again heavily depend on achievements firstly made in robotics.

1.1 Motivation

Transportation is one of the major problems of our current economic system. Almost every industry needs to transport objects from point A to point B.

It is estimated that at least 70 million people[1] worldwide are working directly in the transportation industry. When talking about transportation, mobility of individuals is very similar and deals with the same needs.

Autonomous robot delivery systems can combine several advantages at once. Instant or unscheduled delivery will increase the productivity by delivering faster. The transportation of goods can be accomplished with a fraction of the costs compared to conventional systems. Furthermore autonomous robots can easily adapt to a large variety of tasks or external influences. Only very flexible, cost-efficient and smart solutions will master the challenges of ever-changing customer needs for logistic industries.

1.2 Goals and Challenges

In this section we further describe the goals and challenges of such a fully autonomous transport system. The overall goal is a functional navigation system which is able to travel autonomously in a campus like environment. The campus of the Graz University of Technology at the Inffeld site is a typical example. An overall schematic view of the campus can be seen in Figure 1.2 and in Figure 1.1 a typical surrounding is shown. Such regions are especially challenging because autonomous driving is only very limited possible. The optimization of intra-logistics on similar sites will require autonomous robot solutions in the near future.

[1] *http : //www.ilo.org*

The environment is changing to some extent, either because of coming and going construction sites or because of very crowded and populated pedestrian zones. Urbane environments are very complex, dynamic and thus challenging scenarios with a lot of potential pitfalls, like negative obstacles as depressions, ditches or potholes. A lot of moving objects, not only people but bicyclists and cars which disturb sensor measurements just like constantly changing weather conditions violate global positioning system (GPS) readings.

Next we point out the key aspects of this thesis.

1. Producing a **global consistent map** offline, which is then further used for localization and path planning purposes. The map needs to be consistent and precise enough so that the final localization accuracy is sufficient.
2. **High-level navigation** which is fast and efficient enough to work on large-scale environments and flexible to avoid blocked routes. Planning routes for several hundreds of meters works in real-time.
3. The system is capable of operating **in- and outdoors** and is able to switch instantaneously between both scenarios.
4. It incorporates **reasonable routes** into path planning and chooses sidewalks and pedestrian-zones instead of moving straight to the goal.
5. **Traversability analysis** is necessary to avoid negative obstacles like curbstones but also depressions and ditches. Furthermore unwanted terrain like vegetation should be bypassed as long as possible.

1.3 Contribution

This thesis presents a complete setup, comprising hardware and software solutions, of a fully autonomous transport system for outdoor and indoor usage. Descriptions of complete setups like this one are rather rare and most of the time only sub features are

Figure 1.1: Typical day at the Inffeld campus of the TU Graz.

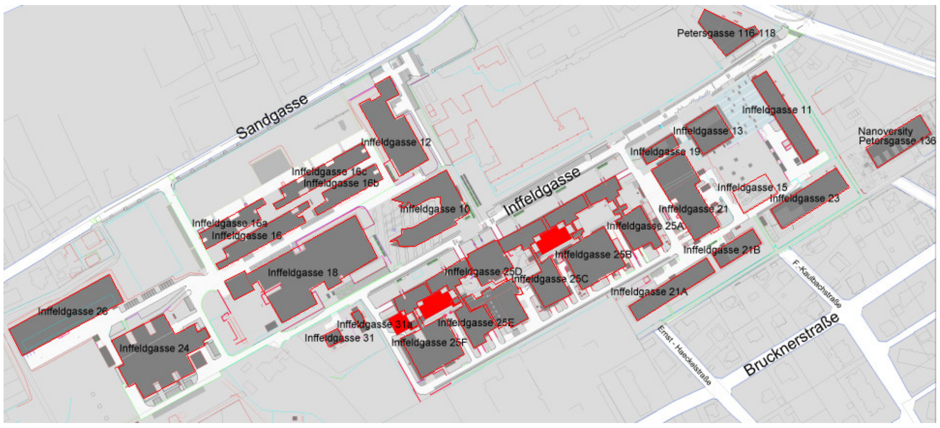


Figure 1.2: Inffeld campus of the Graz University of Technology. The campus is on the longest side approximate 700m long and 320m wide (map image © TU Graz).

discussed. Therefore, we point out all relevant challenges, solutions and future work of a complete autonomous transport system setup for operating in urban areas. We show an offline mapping solution by utilizing pose-graphs with additional sensor processing, to build a global consistent map. The constructed pose-graph builds the foundation for an extended topological graph which is used for localization and in conjunction with a roadmap for navigation. These environment representations are very useful, particular in dealing with very large-scale maps. It has computational benefits and this is important when coping with insufficient processing power, a common problem for mobile robots. Furthermore we propose navigation procedures on different stages, which make path planning on large scale maps tractable but still incorporates reasonable routes into planning. This is possible by building roadmaps with unique graph topologies beforehand and applying them when necessary. The navigation execution is therefore processed by a high-level planner which is then triggering the low-level modules.

1.4 Outline

The following chapters are organized as follows. Chapter 2 gives a formal representation of the transport task. Related research with link to this thesis is presented in Chapter 3, Chapter 4 contains necessary basic prior knowledge. The overall concept and several design decisions for hardware and software as well as algorithm explanations are described in Chapter 5. Implementation details can be found in Chapter 6. The results of all the different parts are then evaluated and discussed in Chapter 7. Finally we conclude with a discussion the outcomes, lessons learned, additional improvements and propose future work in Chapter 8.

2 Problem Formulation

This thesis describes an autonomous transport system with one robot $\mathcal{R} := \{R_0\}$. In an environment $\mathcal{E}$ the robot R_0 is moving and $\mathcal{E}$ is defined within a two dimensional space as $\mathcal{E} := \{x \in \mathbb{R}^2\}$.

The robot is traveling from the starting point s to the goal g ($s, g \in \mathbb{R}^2$), for every navigation task $T \in \mathcal{T}$. Navigation tasks $\mathcal{T}$ are tuple of $\langle s, g \rangle$.

To accomplish any navigation task a map like environment representation is necessary. We then use this abstract model for determine the robot position and furthermore creating plans.

The main map structure is formulated with a pose-graph P and is defined with $P := (N, C)$.

- N is a set of nodes. Where nodes are similar to robot poses and every node n is fully described with $n_i = (x_i, y_i, \theta_i, S_i)$ whereby each node can carry informations from arbitrary many sensors, $S_i := \{s_i^1, s_i^2, \ldots, s_i^j\}$. In our particular situation nodes are carrying s_{Laser} and s_{GPS} measurements.
- C is a set of edges and every edge is connecting two nodes and is defined by $c := (n_1, n_2, k)$. Where n_1 and n_2 are nodes and k relates to the numerical measurement error.

For solving the mapping problem and creating a pose-graph we acquire a sequence of odometry $u_{1:T} = \{u_1, \ldots, u_T\}$, laser $l_{1:T} = \{l_1, \ldots, l_T\}$ and GPS $gps_{1:T} = \{gps_1, \ldots, gps_T\}$ measurements. The odometry measurement comprises the translational velocity v^t and the rotational velocity w^t at time t. The odometry measurement is defined with $u_i = (v_i^t \; w_i^t)^T$. GPS measurements include information about longitude, latitude and altitude and is thus described with $gps_i = (long_i \ lat_i \ alt_i)^T$. Laser measurements consist of multiple range measurements $l_i = \{r_1, \ldots, r_j\}$. The unknown variables trajectory $x_{1:T} = \{x_1, \ldots, x_T\}$ and the map m can than be estimated. This map representation can further translated into local occupancy grid maps. Grid maps discretize the world into cells and each cell is assumed to be occupied or free space. Let $\mathbf{cell}_i$ denote individual grid cells, hence occupancy grid maps are formulated in Equation (2.1).

$$grid\_map = \sum_i \mathbf{cell}_i \tag{2.1}$$

Occupied cells are notated with $p(\mathbf{cell}_i = 1)$ and free cells with $p(\mathbf{cell}_i = 0)$, thus $p(\mathbf{cell}_i)$ describes the probability if a grid cell is occupied.

Instead of modeling a complete 3D representation of the environment we define a two-dimensional grid with three-dimensional information. The elevation map discretizes the world into cells (Equation (2.2)).

$$elevation\_map = \sum_i \mathbf{ec}_i \tag{2.2}$$

Each cell $\mathbf{ec}_i$ can be described as a tuple of $\mathbf{ec}_i = \langle o_i, \mu_i, \sigma_i^2 \rangle$. Where o_i is the occupancy probability, μ_i is the elevation and σ_i^2 is the variance of the estimation.

Determine the own position is called localization and a necessary precondition for planning a path toward a goal. Localization in this thesis is the process of estimating the robot position x_t against a known map m. The robot position is defined with $x_t = (x \ y \ \theta)^T$. For that purpose we keep a probability density $p(x_t | l_{1:t}, u_{0:t-1})$ of the position at time t given all laser measurements $l_{1:t}$ and control inputs $u_{0:t-1}$. The resulting probability distribution suffices to approximate a state hypothesis of x_t.

We utilize a navigation graph for planning, defined of the form $\mathcal{N} := \{P, G, E, O\}$. This graph represents navigability, so edges only connect grid maps if and only if a possible path exists between them.

- P is a pose-graph.
- G is a set of local grid maps.
- E is a set of edges connecting neighboring grid maps. Edges are of the form $e := (g_1, g_2, t)$ where g_1 and g_2 are grids and t is the relative cost.
- Every grid maps g owns a set of overlaying nodes so that $g, O_g \subset N$. O_g is therefore a set of nodes inside the covered area of g.

A path is defined as a continues mapping $\pi^{(R_0)} \colon [t_{R_0}, t_{R_0} + \Delta t_{R_0}] \rightarrow \mathcal{E}$ which is further described in the book of Choset [1]. The start of the path is constrained to $\pi^{(R_0)}(t_{R_0}) = s_{R_0}$ and the goal with $\pi^{(R_0)}(t_{R_0} + \Delta t_{R_0}) = g_{R_0}$. Planning on the navigation graph is then separated in three sub-tasks. Selecting the start way-point n_s and the goal way-point n_g closest to the robot and the goal respectively. The topological path can then be computed with $astar(n_s, n_g)$.

We further assume the low-level planning and execution as solved.

3 Related Research

This chapter gives an overview of relevant research which is related to this thesis. The first part describes relevant mapping techniques and optimization procedures which have already successfully demonstrated in real scenarios. This is followed by complete outdoor navigation systems dealing with similar applications and presenting already some effective solutions. The last part discusses some literature on dealing with terrain classification to allow robots to navigate on save ground.

3.1 Mapping

Map creation of the environment is an essential skill for mobile robots. Robots use these maps for localization and navigation. The process of map building is called simultaneous localization and mapping (SLAM) [2], while the robot tries to localize itself inside the current environment model it successively integrates more measurements to the map and thus extends it. It is a fundamental and complex problem, because acquiring a consistent map requires a good localization estimate and vice versa.

The most famous map creation paradigms in literature are Kalman filter, particle filter or graph-based. For example `GMapping` [3] provides a highly efficient SLAM solution. It implements Rao-Blackwellized particle filter to create maps. Each particle carries therefore an individual map of the environment as well as the trajectory of the robot. The map strongly depends on the trajectory, which can be efficiently estimated first. The sampling and thus the next generation of particles is obtained by applying a proposal distribution, whereby an odometry motion model is typically used. Each particle gets an individual importance weight assigned and then resampled. Only individual particles are selected for resampling, depended on their importance weight. It takes odometry and laser range measurements as input and outputs an occupancy grid map.

3.2 Graph-Based SLAM

The simultaneous localization and mapping can also be represented by a graph. This technique is even older than the particle filter but became popular in 2006 due to several new mathematical toolkits, which made the computation practicable. These algorithms solve nonlinear error functions in an efficient manner and can be adapted to a wide range of other problems. Graph-based slam state of the art techniques are able to deal with very large maps and are highly customizable and process all kind of sensor inputs. New enhanced algorithms encouraged map sizes even beyond several kilometers by introducing hierarchical approaches [4] and maintaining different level of detail. Due to its versatility it can be used for optimizing high dimensional problems and therefore able to build large-scale 3D maps.

One famous and widely used open source framework is g^2o [5], presented and developed by Kümmerle and Grisetti et al. The constructed graph consists of nodes (vertices) which represent robot poses at a certain time and edges which connect corresponding poses due to same measurement observations. Each node carries information about it sensor readings and edges can then measure the correlation between each pose. When the graph is constructed the optimization routine finds the most consistent representation. For more detail information see Section 4.3.

3.3 Navigation Systems

Navigation is a core functionality for all mobile robots. This combines the ability to estimate the current position and simultaneously plan a path to a goal. Most state of the art solutions require a prerecorded map which was taken in forehand either by exploring autonomously or by controlling the robot remotely. In indoor scenarios autonomous exploration [6] is very popular to automate the mapping process. This technique has the major advantage of creating maps without any supervision. For that reason, commercial systems like modern robot vacuum cleaner for example the iRobot Roomba 980, already offer this functionality. However the larger the operating area for navigation becomes, the more suitable technique is remote controlled [7, 8] or GPS guided [9]. These systems rely on a previously taught map for reaching a specific goal location within the recorded area. When controlling the robot remotely the operator can exactly specify which part of the environment should be learned. In some cases the robot is even restricted to drive the exact same route. The majority of navigation systems so far focused on indoor scenarios or in comparable small operating areas, apart from a few exceptions which are described next.

One of the few systems special designed for the outdoor navigation task is the autonomous city explorer (Bauer et al. [10]). It successfully took a longer tour through Munich without any map knowledge or GPS information, solely from information given from passers-by. The robot frequently interacted with pedestrians which were the primary source for orientation and direction information. It approached humans autonomously and recognized a number of basic commands by using a speech recognition system.

Kümmerle et al. [11] describes a complete system for mobile robots navigating in urban scenarios. Obelix the robot of the University of Freiburg managed to navigate autonomously in the city center for a distance of more than 3km. This system uses accurate maps of the city center in conjunction with GPS. The maps where generated with a SLAM procedure while someone manually drove the robot along a route. The system contains a dedicated map data structure for dealing with large-scale maps and a variant of Monte-Carlo localization realized with a particle filter. What makes this work interesting is the way how the system is designed to deal with very large maps. Their dedicated map data structure allows the robot to use large amount of information in an effective manner. Instead of computing and working with one large map they computed a lot of small ones locally. Due to their graph-based SLAM solution another graph consisting of nodes and edges is constructed, where each node comprises a local map computed with the information stored in the first graph. Furthermore this data structure allows different level in abstract planning. The topology of the graph connecting the local maps is considered by the high level planner for computing a path towards a goal location. Therefore each edge of the topological graph represents navigable paths.

Košnar and Krajnık [12] introduced the large map framework (LaMa). This framework is intended to handle spatial knowledge of large environments and also utilizes a topological map to acquire and store all necessary data. They used this framework together with vision algorithms to keep the robot on the road and detect crossings. Crossings create a vertex inside the topological graph and then get connected with neighboring vertices by edges which again represent paths.

3.4 Terrain Analysis

Navigation systems need to analyze the surrounding terrain to ensure a robot's safety on slopes and uneven surfaces. This skill is further necessary to robustify navigation but also useful to just take an easier path to the goal. Many decisions depend on robot capabilities, like traversing certain obstacles like curbstones or on desired behaviors such as avoiding vegetation to keep the robot on a track.

Common vision based system [13, 14] try to differentiate between paths (drivable) and non-drivable terrain. The image is therefore classified in regions, which are either drivable or not. This procedure is rather difficult because the quality is affected by plenty of factors such as lighting or surface material of the road.

In research point clouds, elevation maps [15, 16] and multi-level surface maps [17] became common terrain map types. All of these map types require some sort of 3D measurements of the surrounding, which is typically achieved with laser scanners or depth cameras. Elevation maps or height maps for example serve as a very good terrain representation because they are memory efficient and satisfy all required information to distinguish different terrains or all kind of obstacles. In many cases a local window around the robot even suffices, this saves memory and computational effort. Such a map stores an estimated height for each grid cell and is also referred as a 2.5D representation of the environment. For further information see Section 5.7.

Successful robot configurations in the past, like Stanley [18] from the Stanford University, not only used a single laser scanner to acquire the surface data but also a computer vision system. This was especially necessary when driving speeds exceeded 40 km/h and the laser scanner would not measure far enough.

4 Prerequisites

This chapter serves as an overview for all basic prerequisites, which are essential for this work. It contains principle algorithm designs as well as software frameworks.

4.1 Robot Operating System

The Robot Operation System (ROS) [19] is a popular robotic framework which serves as a basic communication layer. ROS mainly requires an operating system as a host and runs on Linux-based platforms. This framework delivers easy to use, out of the box features and tools for package management and inter process communication. Due to its heterogeneous multi-computer design it is able to spread tasks on multiple on- and off board machines. The communication is based on TCP/IP sockets for transporting message over networks. It supports C++, python and several other programming languages and is thus suitable for a wide variety of uses. ROS is fully open source and comes with a diversity of packages which have been released from a big community. Additional documentation can be found at `http://wiki.ros.org`.

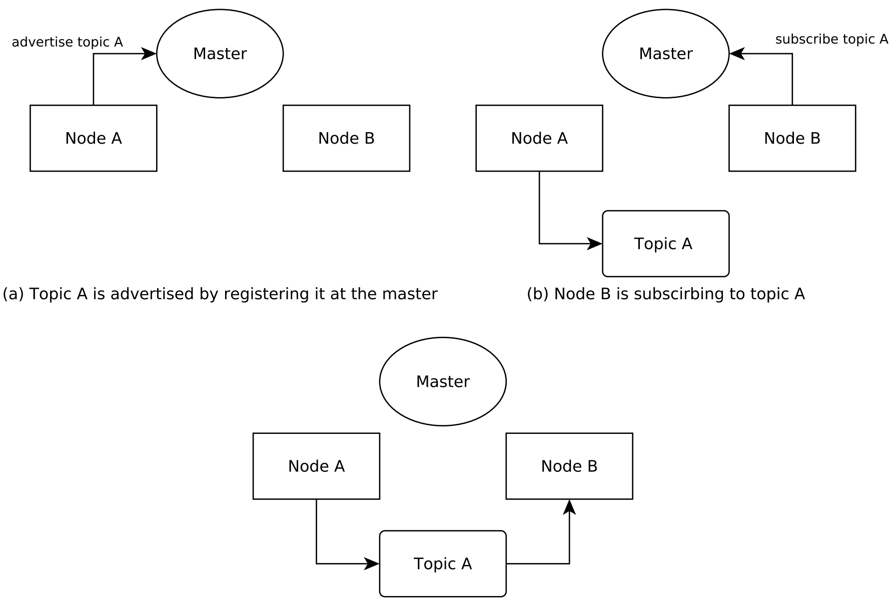
In the next paragraph the main architecture is described in more detail. Processes are called nodes inside ROS and the communication between them is handled with predefined messages. Each single node performs individual tasks and computation, like driver module or perception. For reasons of flexibility, nodes have unique names for identification and can run simultaneously. Predefined messages are described through the interface definition language

(IDL) and are build up with primitive data structures. The transaction of messages is done over topics (Figure 4.1), which are similar to uniquely named channels. Within one topic only one message type can be exchanged. When one node is sending data, it publishes the message to the topic and then other nodes can subscribe to the topic to receive the message. Nodes can subscribe and publish on several topics at once but communication partners are never aware of their opposite. In some cases the communication needs to be synchronized and this is achieved with so called services. Services are provided by certain nodes, following the same principle as web services and use well defined messages similar as topics. The service design is shown in Figure 4.2. To manage all the nodes and communication a core process is used, the ROS master. This process keeps track of all registered nodes, every topic and service. The ROS master is responsible for setting up a peer-to-peer connection between appropriate nodes. Furthermore the master includes a parameter server, where nodes can either share or look up specific parameters.

The actionlib library introduced another missing concept, which enables ROS to cope with long lasting tasks and asynchronous communications. With this library it is possible to receive feedback while task execution or even stop it entirely. The action concept is illustrated in Figure 4.3. The actionlib implementation is build up on services provided by ROS.

4.2 Move-Base

One of the most known ROS packages is the `Move-Base` [20]. Developed by the community to tackle the indoor navigation task and features a wide set of customizable settings to fit a lot of different robots. First the package receives goals in real world coordinates and plans global paths to the desired locations, this is called the `GlobalPlanner`. As it can be seen in Figure 4.4 the `GlobalPlanner` requires a so called costmap, for computing valid paths. Costmaps



(a) Topic A is advertised by registering it at the master

(b) Node B is subscirbing to topic A

(c) Messages are then directly transmitted from Node A to B

Figure 4.1: Topic communication concept in ROS, with publisher and subscriber node. Adapted from `http://wiki.ros.org`

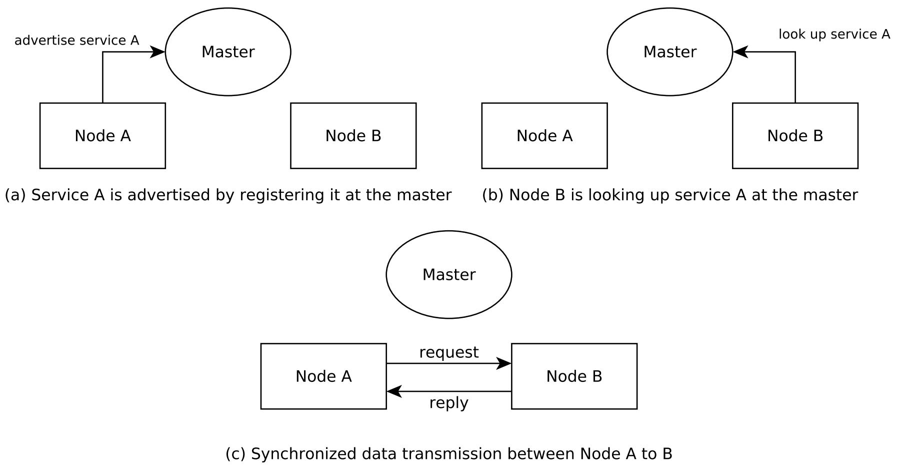

advertise service A
Master
Node A
Node B
(a) Service A is advertised by registering it at the master
Master
look up service A
Node A
Node B
(b) Node B is looking up service A at the master
Master
Node A
request
reply
Node B
(c) Synchronized data transmission between Node A to B

Figure 4.2: Synchronized service communication concept in ROS, with advertiser and look up node. Concept taken from `http://wiki.ros.org`

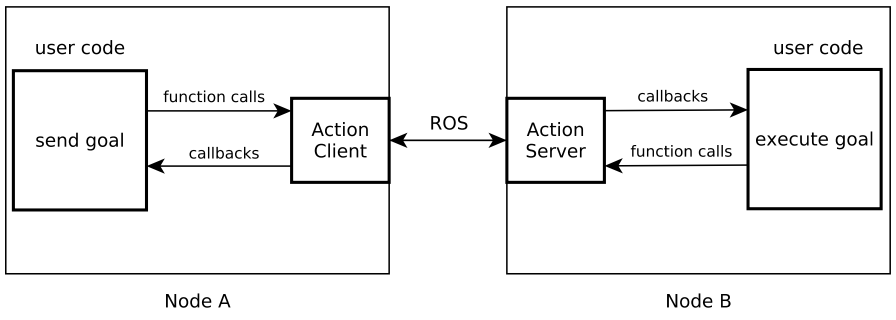


Figure 4.3: Concept of the action-server communication provided by actionlib. Figure adapted from `http://wiki.ros.org`

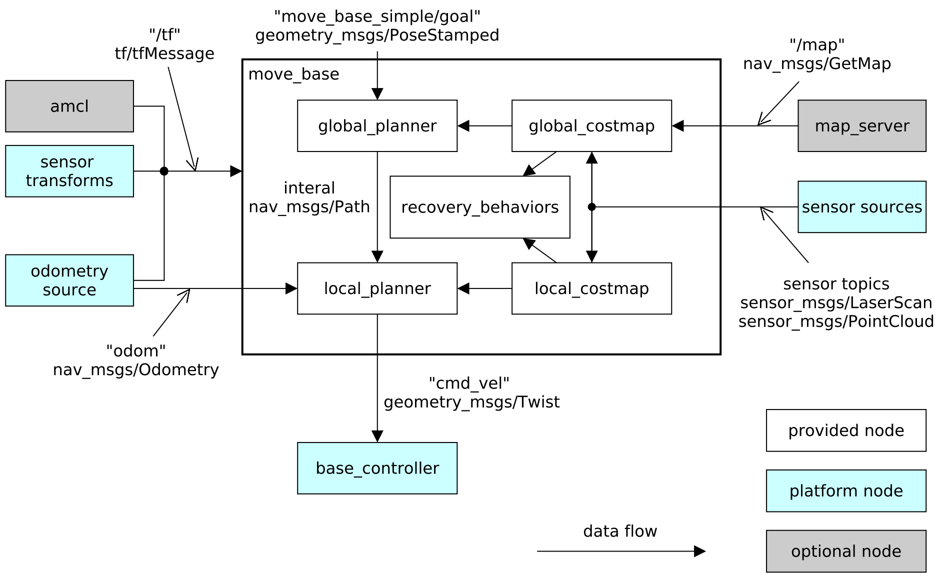


Figure 4.4: Concept of the navigation stack together with `Move-Base`. Figure adapted from `http://wiki.ros.org`

are basically grid maps but include further information which are relevant for navigation. Costmaps distinguish between free and occupied cells and apply costs for every grid cell. This is necessary for computing paths. The global costmap is typically created with an environmental map and some additional sensor sources. In the second stage it computes trajectories and finally velocities for every motor controller with the `LocalPlanner`. The `LocalPlanner` requires also a costmap similar to the `GlobalPlanner` but only of the near local neighborhood.

4.3 Graph-Based SLAM

As mentioned in Section 3.2 the SLAM problem (Section 3.1) can be formulated within a graph structure, like illustrated in Figure 4.5. These graphs are also called pose graphs because robot poses (circles colored in cyan and labeled with x^S denote robot poses) at different measurement times, are connected to each other. Usually robots are equipped with an odometry sensor, this enables the robot to measure relative movements in a time interval. This relative information is used to connect subsequent robot poses within the graph and labeled as z^S.

In addition some landmark detection is required, whereby a landmark can be any object in the environment. In Figure 4.5 diamond shapes in green represent landmarks and are marked as x^L. Typically, when using laser scanners corners and edges are treated as landmarks. Due to the noisy odometry measurements, landmark detections serve as a correction of robot positions and thus can interconnect multiple robot poses at different timestamps. The sensing of landmarks is also affected by noise and therefore edges between robot poses and landmarks also introduce some measurement error z^L.

Graph-based SLAM decouples the problem in two sub tasks which are separately solved in the front or back-end part. This showed to be very effective for dealing with noisy measurements. First the graph needs to be constructed from the raw measurements, this is done in the front-end part. Then the graph optimization is within the back-end part, here the most likely configuration of the graph will be determined.

The front-end is responsible to compute a consistent estimate of the robot trajectory. Although some measurement could induce multiple resulting edges to different poses, it is advisable to restrict the process only to the most likely topology. The back-end is then trying to find a spatial configuration which satisfies the earlier constructed model, by minimizing the edge errors based on a least-squares solution. Every single edge is characterized with an error function and an uncertainty matrix, this formalizes the quality of each connection. In Figure 4.6 the edge error definition is illustrated.

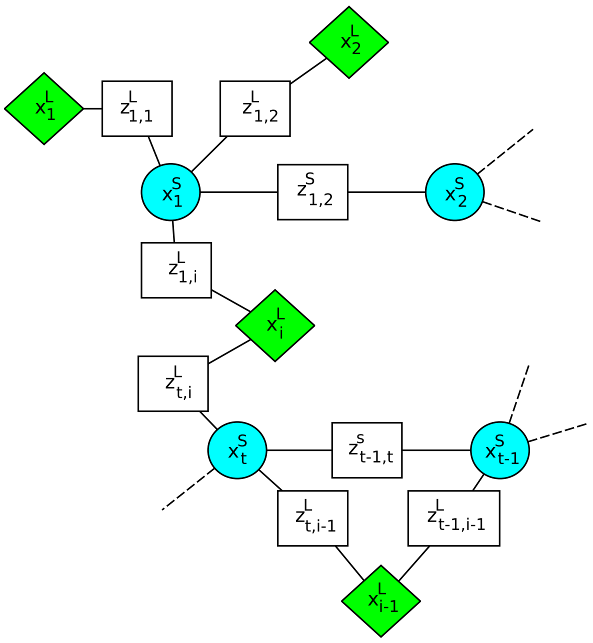


Figure 4.5: SLAM process represented as a graph. Circular nodes represent robot poses x^S and green color denotes landmarks x^L. Edges between consecutively robot poses are odometry measurements z^S and measurements which connect landmark to robot poses are labeled with z^L. Figure adapted from `https://openslam.org/g2o.html`

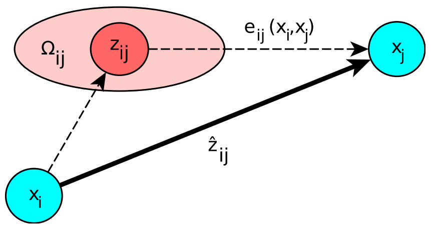


Figure 4.6: The error $e_{ij}(x_i, x_j)$ results from the offset between the real z_{ij} and expected measurement $\hat{z}_{ij}$. Measurements are typically gathered from odometry sources when solving the SLAM problem. Every pose graph edge is specified by its error function and uncertainty matrix Ω_{ij}. Figure adapted from `https://openslam.org/g2o.html`

4.4 Adaptive Monte Carlo Localization

The Monte Carlo localization algorithm as proposed by Fox et al. [21] is a probabilistic approach to solve localization problems for mobile robots. The algorithm estimates the current position and orientation of the moving robot by taking current measurements and fitting them against a known map. It uses particles which can be somehow distributed and therefore fit every arbitrary distribution in reality. Each particle represents a possible state. If there is no information about the start position given, the particles are uniformly distributed over the full map, so that every position is evenly likely. Every time the robot moves, all particles are moved the same way and when receiving a measurement the particles are resampled accordingly. After each newly received input the algorithm predicts the new robot state.

The most crucial part is the number of used particles. While too many particles could slow down the computation and one would risk its real time capabilities but when using too few the model is not capturing the reality good enough and this can cause localization failures. To circumvent this problem adaptive particle numbers had been introduced. Such adaptive methods use adequate particle amounts, depending on the current localization quality. ROS already provides a localization package which implements an adaptive particle filter with many different algorithms described in the probabilistic robotics [2] book. In this thesis we utilize this package which is called adaptive Monte Carlo localization (`AMCL`).

4.5 Planning

Path planning is described as the process of searching for the shortest route between the start and goal point in principle. It can be distinguished between path finding on metrical maps and graph traversal. Both use cases rely on an environment model to carry out the planning procedure. One of the most famous search algorithm is A*. Hart and Nilsson et al. [22] described it in 1968 and although some hybrid variants emerged, it is still a state of the art solution. The algorithm requires a heuristic to guide the search to the goal and that is the reason why it always picks the most favorable variant among all possible paths. So A* always explores those paths first which appear to have the lowest costs.

For path finding or metrical planning a grid model of the search space is utilized, which is an obvious choice because todays mapping (Section 3.1) processes already output occupancy grids. Graph traversal, in contrast to metrical planning, uses an abstract graph model of the environment to find the shortest route between two points. To finally reach the desired goal, the robot has to visit each vertex of the computed route and this could sometimes lead to suboptimal routes compared to the grid-based metrical planning approach. Very effective solutions combine both methods to speed up the hole planning procedure. Planning on vast areas becomes tractable with graph traversal and then the grid-based search is further used to fine grain the route.

5 Concept

This chapter gives an overview of all relevant design decisions taken during each step of the system development. It includes hardware choices as well as software design patterns. Some concepts are described in more detail for a deeper understanding of the basic design, which is necessary for being able to follow Chapter 6 about the implementation. The first section presents the hardware used as a base for experiments. The overall software design is discussed next and then we provide details about individual software modules used for solving particular problems.

5.1 Overview

In the early design phase we had to choose the best robot configuration to fit for an autonomous transport robot. The first explicit choice we had to make was the robot base. It needed to be able to drive outdoors and to have a reasonable size to fit all the needed sensors but nevertheless it should work for indoor tasks as well.

After the available transport capacity was known the necessary sensor set can be put together. For outdoor tasks GPS was an obvious choice to become aware of the global position. Due to several reasons, including privacy and security concerns we decided not to use any camera systems. Cameras have very good future prospects for autonomous vehicles because they have a very low-price and cover a wide range of applications. But considering how strong light influences the capturing quality, especially in the evening or dawn, a good sensor combination is mandatory for robots.

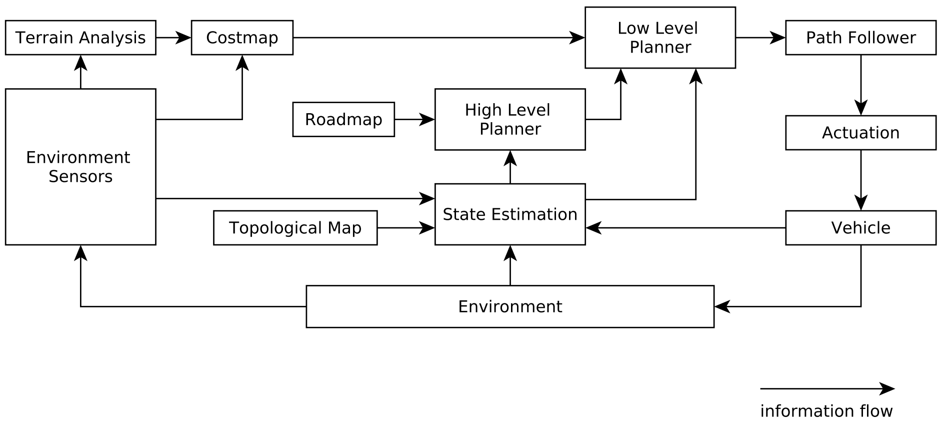


Figure 5.1: Overall system architecture.

Laser scanners are the most valuable sensors for us to pick because these types of sensors are very precise, easy to use and not very prune to measurement errors. Finally we added an inertial measurement unit (IMU) to the sensor setting, which provides indispensable orientation measurements.

Figure 5.1 shows our general system architecture. Environmental sensors (see following Section 5.1.1 for more details) are used to estimate the current state, detect obstacles and to analyze the terrain. The state estimation is receiving information from multiple sources and delivers a position estimate within the environment for both planners. The high level planner assigns goals to the low level planner which furthermore computes paths for the path follower and finally moves the vehicle.

In Figure 5.2 the overall architecture for data acquisition concept is illustrated and is further described in Section 5.1.2. The robot operator is manually driving the robot. Then the system is building all the necessary environment representations.

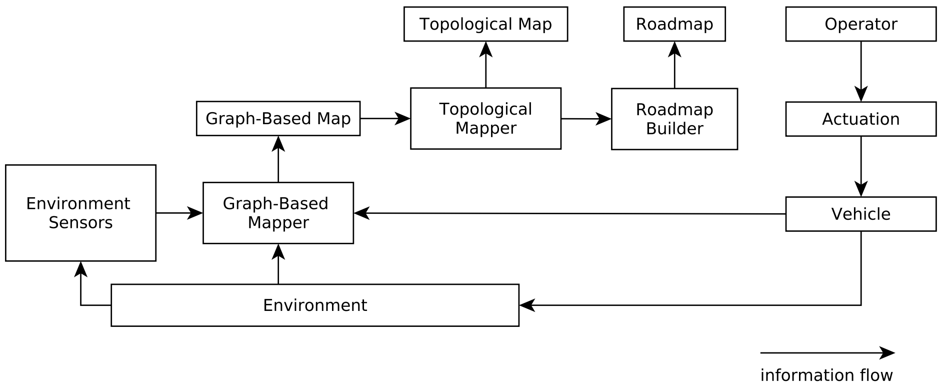


Figure 5.2: System architecture for data acquisition.

5.1.1 Hardware

The robot used for field testing is a Pioneer 3 AT with a custom assembly. This model was the most promising choice for us, as it is able to drive outdoors and has a reasonable size as mentioned in the previous section. The robot has a skid steering drive with a maximum forward speed of 0.7 m/s. This robot base is designed for asphalt, sand, dirt as well as carpet and in indoor use. The maximum traversable step height is about 10 cm but with all sensors installed the full robot weight increased from 12 kg to approximately 23 kg and for this reason we decreased the traversable step height to 5 cm. The footprint of the complete robot is 0.65 m x 0.5 m and has a height of around 0.8 m (Complete robot assembly can be seen in Figure 5.3 and Figure 5.4).

On the top center a Lenovo notebook is mounted for running the entire software and to visualize the planned route and further information about the navigation system. The notebook is powered by an Intel Core i5 processor with a clock speed of 2.53GHz and has 4GB DDR3 RAM.

Additionally a front facing webcam is attached to the robot but only for capturing the tests and easier troubleshooting. The environment measurements are very long processes. If any unpredicted effect occurs, e.g. very erroneous sensor behavior, we can have a look at the captured images of the webcam afterwards and explain the results.

The main sensors are laser range finders (LIDAR). In our setup we use three LIDARs, whereby two SICK LMS 200 are mounted in front and back of the robot to capture the horizontal surrounding. The third laser, a Hokuyo UTM 30LX is added on top the front facing SICK. The Hokuyo laser is declined by 20° to detect obstacles and to identify the terrain (Figure 5.5). The chosen angle of 20° provided be the best translation between detecting obstacles in the distance and loosing information near the robot.

Another reason for this configuration is the terrain classification which distinguishes between vegetation and drivable roads. Classification with laser measurements shorter than roughly 1 m are rather difficult and would lead to incorrect results, which is further explained in Section 5.7.

The horizontal laser range finders are configured to measure up to 80 m with a frequency of 10 Hz. This configuration does not allow very fast rotational movements but only long distance measurements gather enough localization features in vast open outdoor environments. The scanning angle is 180 °, thereby appears a lack of coverage at each side of the robot. This should not be an issue because the robot is not able to drive sideward. This can be seen in Figure 5.6.

As for the IMU we used a XSense MTi-G which provides GPS functionality with an external antenna. The IMU is appropriate orientated, so that the sensor uses the same x-axis, y-axis and z-axis direction as the robot. In our configuration the driving direction is the x-axis. It is positioned inside the body center as close as possible to the center of rotation. The GPS antenna is located at the top of an extra pole on the robot to obtain additional space between antenna, ground and robot. In Figure 5.3 the sensor layout is shown.

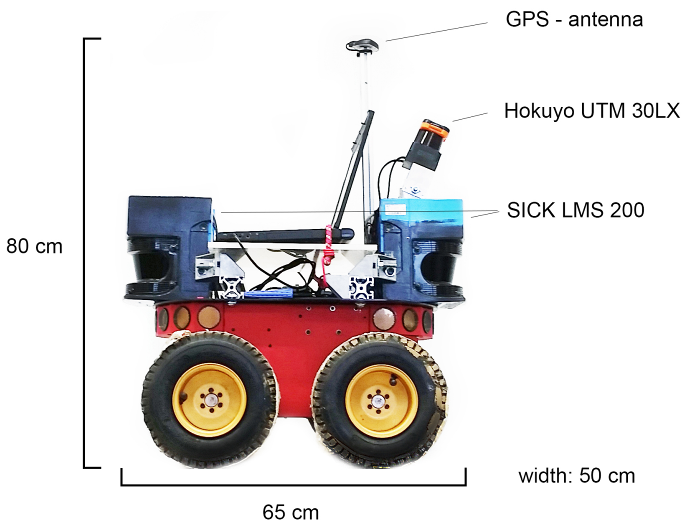


Figure 5.3: Complete robot setup in side view, with all comprising sensors labeled.

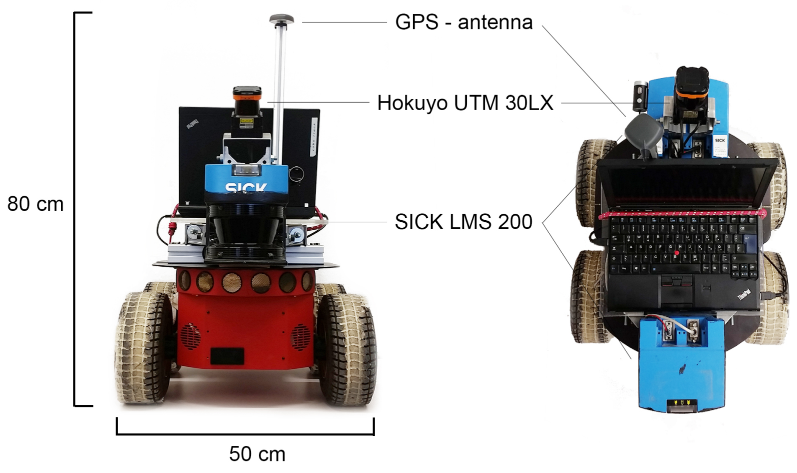


Figure 5.4: Robot setup in front and top view with all visible sensors marked.

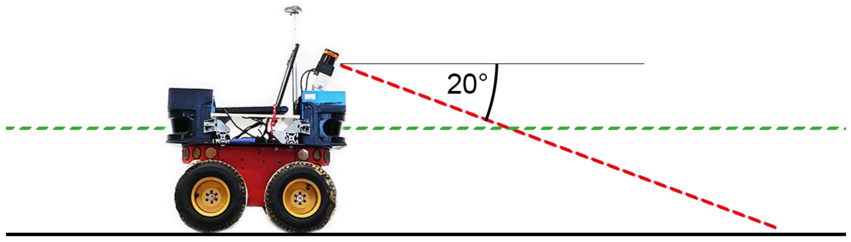


Figure 5.5: The laser sensor setup from the side view. Green lines mark the horizontal lasers (SICK LMS 200) and the red one shows the terrain analyzing laser (Hokuyo UTM 30LX).

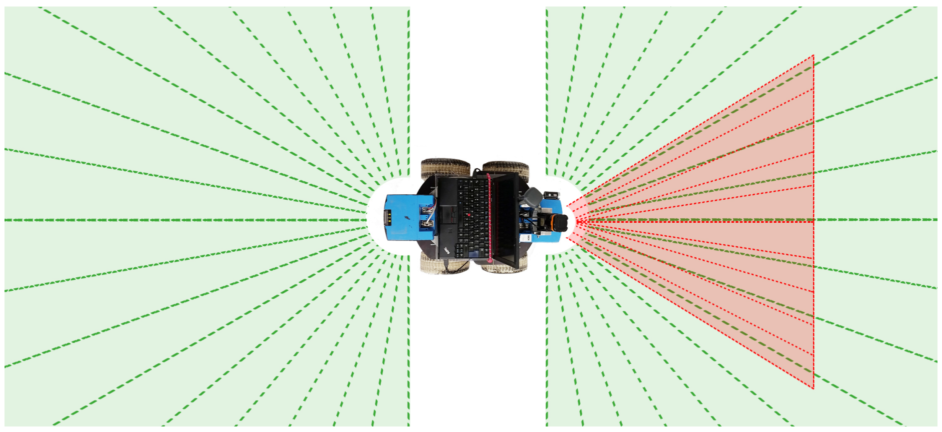


Figure 5.6: Laser sensor setup from the top view. Green areas show the covered area from the horizontal lasers and red areas denote the downwards facing laser.

5.1.2 Software

The software concept is illustrated in two figures. Figure 5.7 shows the mapping process while Figure 5.8 depicts the autonomous planning procedure.

For mapping the environment we have to carry out the following five steps. The process begins by taking measurements of the operating environment. Which is as easy as driving the robot around manually. This step requires recording only the raw sensor information but we suggest to include the preprocessing as it saves computational time later. In the preprocessing step all incoming sensor measurements are prepared for further computation. Beside some basic filtering (described in Section 5.2) of each laser scan we had to deal with both horizontal laser scans simultaneously and therefore introduced a laser scan chain, which is described in Section 5.2. After preprocessing the sensor data we perform a graph-based mapping (see Section 5.4.1) which outputs a global map representation. This information is then fed into the topological hybrid mapping module (more information in Section 5.4.2) for creating local maps and building a map graph. Finally a road map is build using the previous computed topological hybrid map.

In contrast to the mapping process, the planning is almost entirely online. As illustrated in Figure 5.8 the topological planner (see Section 5.6.2) receives the final goal as well as the current localization and outputs intermediate goals for the low-level planner (further details in Section 5.6.3). The localization module is described in Section 5.5 and uses the active local map and the topological hybrid graph to obtain a pose of the robot.

5.2 Preprocessing

After the experimental robot setup was finished (see Section 5.1.1), we had to prepare all the different sensor sources to gather reliable measurements. Therefore we first decided to introduce a `laser`

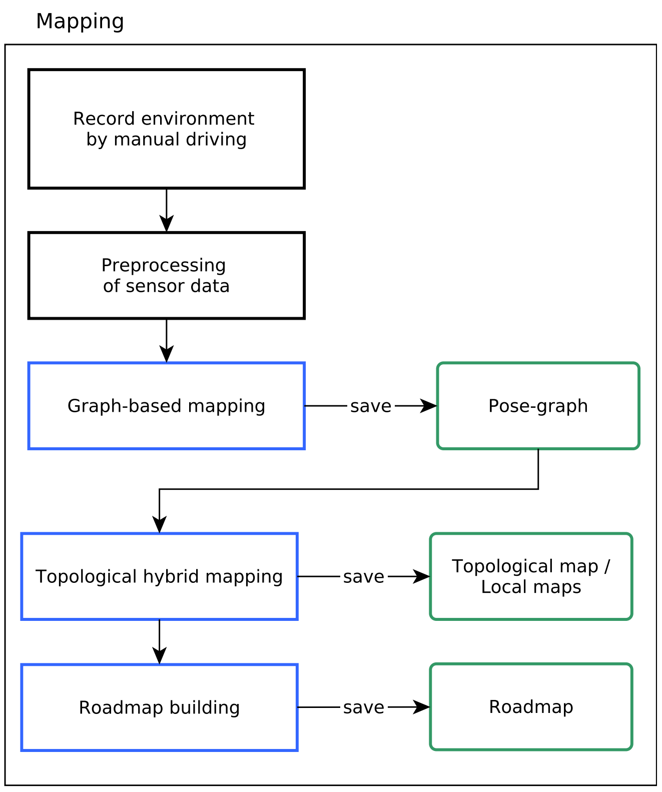

Mapping
Record environment by manual driving
Preprocessing of sensor data
Graph-based mapping
save
Pose-graph
Topological hybrid mapping
save
Topological map / Local maps
Roadmap building
save
Roadmap

Figure 5.7: Overview for the mapping process. Components framed in blue color are computed offline and green color denotes data structures.

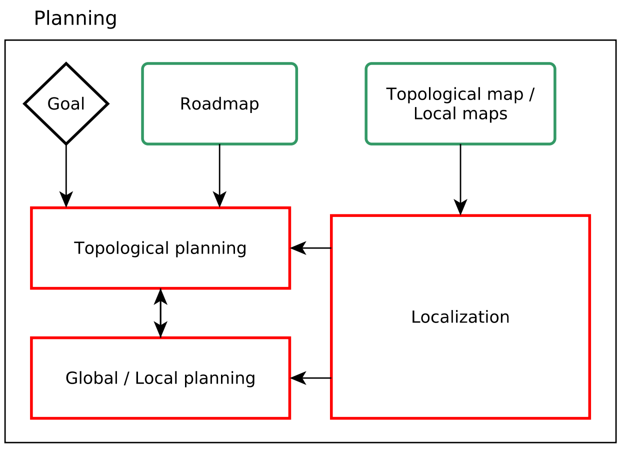


Figure 5.8: Workflow concept for planning. Green color indicates data structures and red components require an online computation.

`scan chain` for easier maintenance of both horizontal laser. The data flow chart is shown in Figure 5.9. Providing a noiseless laser scan due to preprocessing is not only essential for detecting reliable obstacles while operating autonomously but also saves unnecessary post-processing at map creation. As illustrated in Figure 5.9 the scans are merged at the beginning and then passed through some basic filtering operations offered by the point cloud library [23]. One basic filtering mechanism is the statistical outlier removal [1] which analysis the mean distance for every point and distances or standard deviations outside an interval are considered as outliers. The box filter [2] afterwards removes erroneous measurements which are too short. While recording data used for map generation, the box filter can even be set to two meters so that the operator following the robot is removed.

Due to missing 3D sensing capabilities and the assumption of driving on flat terrain we implemented a very simple ground

[1] *http : //pointclouds.org/documentation/tutorials/statistical_outlier.php*
[2] *http : //docs.pointclouds.org/trunk/classpcl_1_1_crop_box.html*

removal filter (Figure 5.10). Although the robot never updates its vertical believe, it suffices in many situations to transform the scan by using the orientation of roll and pitch from the IMU. We therefore receive a relative height estimate of each measurement (shown in red color) and then simply remove scans exceeding certain thresholds (thresholds are colored in green).

After removing any possible ground measurements the laser scan is finally outputted as laser and point cloud for easier subsequent use. This state of the preprocessed scan should be used for planning purposes. Otherwise more advanced filtering mechanisms like removing moving objects would cause the robot to ignore critical aspects for planning. The filtering of complex objects, such as people or cars is very helpful and measurements are then perfectly usable for localizing or mapping purposes. We therefore added a people filter within the scan chain overview but only to point out the different outputs. Such an advanced implementation would be out of scope of this thesis.

5.3 Odometry

The concept behind odometry is rather simple. The wheel rotations can be translated into linear and angular displacements, so that the offset to a known starting position can be approximated. Odometry has a very good short-term accuracy but in the long run the error is increasing significantly because errors get accumulated. Nevertheless it serves as a very good initial guess for mapping algorithms and is therefore advisable to be included. Hence it is useful to fine tune the odometry as good as possible.

Unfortunately odometry is altered by many factors such as encoder resolution, wheel diameter, ground slippery and transport load to name just a few. The raw odometry measurements are directly outputted from the pioneer base and are basically not calibrated to fit the momentary robot load and tire pressure.

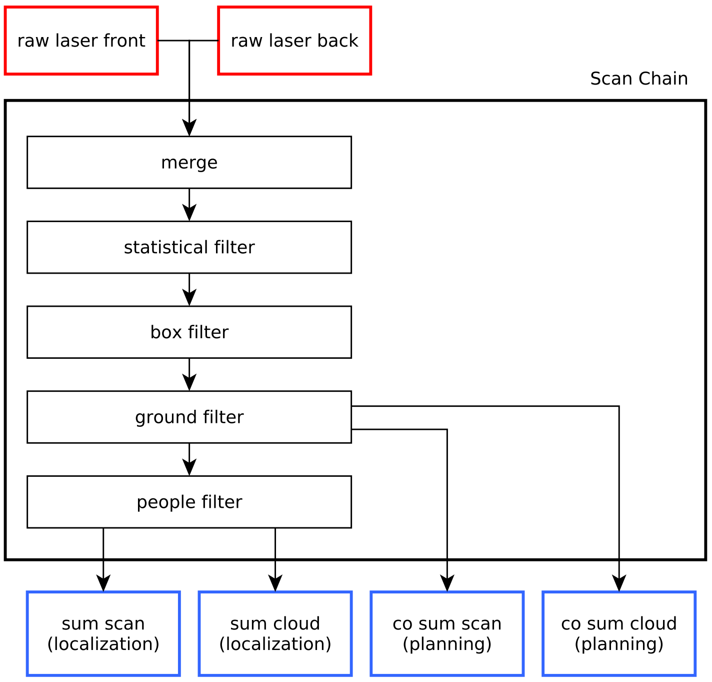


Figure 5.9: Scan chain data flow chart. Combines two laser scans and outputs one merged scan for localization and another for planning.

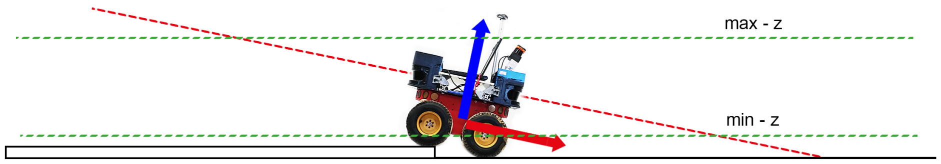


Figure 5.10: Concept of ground removal implementation. Green lines denote thresholds to remove horizontal laser measurements (red color). For a better visualizing of the ground removal filter only the effect of pitch orientations is shown. The final filter takes roll orientations into account as well.

We therefore applied a calibration technique, namely the square path experiment referring to the book of Borenstein et al. [24]. This method is able to deal with systematic errors by determine correction factors for an uncertain wheelbase and unequal wheel diameters. Within this experiment the robot has to drive a square path by using the odometry sensors only. The position at the end is then compared to the start position. This test should be repeated multiple times in clockwise and counter clockwise direction. The correction factors can then be estimated because the repetitions limit the influences of non-systematic errors. After the successful calibration we included both correction factors for improving the raw odometry.

Our complete robot setup comprises multiple other sensors which also can be used as an odometry source. The current implementation employs an unscented Kalman filter (UKF) [25] to provide a nonlinear state estimation and thus a better odometry estimation. T. Moore and D. Stouch et al. [26] already offer a ROS package, namely the `robot localization` which includes various Kalman filter. This UKF implementation is capable of using an arbitrary number of input sources and sensors types. One input source is provided by the earlier corrected wheel odometry and the second input source is the IMU. We fed the velocity given by the odometry and only the orientation measurements from the IMU into the filter.

5.4 Mapping

Map creation is essential for any navigation system, either to provide routing information or to estimate the current robot position. A very common map representation is introduced by Elfes et al. [27] and are called occupancy grids. Every grid cell can directly be transformed into real world coordinates. Each cell marks an area as free or occupied by an obstacle. This environmental model showed to be very effective and is still a state of the art solution for two dimensional maps. As described in Section 3.1 we could have also used GMapping for map generation but it turned out that graph-based mapping methods have far more benefits for reasons of different sensor support and easier maintenance of larger maps. Maps in the size of e.g. our campus will lead to relative large memory footprints and algorithms which refer to this map will suffer in processing speed and probably lose their real-time capabilities. Due to evaluation reasons, we decided to record all sensor information and run the mapping process with different settings offline.

5.4.1 Graph-Based Mapping

Graph-based mapping as described in Section 3.2 and Section 4.3 solves the problem of simultaneous localization and mapping. The Algorithm 1 shows the workflow on how the input data consisting of odometry, laser scan and GPS measurements are processed until a final pose-graph is returned. The input data must be in chronological order, so that the algorithm can take care of neighborhood relations. Furthermore related measurements are picked by minimizing the time interval between the recording times. The current odometry measurement always serves as the reference time and corresponding laser or GPS readings are then chosen depending on their recording time (Line 13, 20).

At the beginning we initialize the pose-graph by adding the first odometry and GPS measurement (Line 4). This very first pose is furthermore fixed and never changed throughout the whole mapping process. New poses are always added whenever either the distance or the angle to the previous pose in the graph is larger than a certain threshold (Line 10). For monitoring the offset we estimate the new pose by adding the odometry difference to the latest pose (Line 8), which was added to the graph (Line 7). Then an edge is added to the pose-graph to connect them. The quality of the edge and thus the bond strength between those poses is described by the corresponding uncertainty matrix. If the scan matching succeeds a smaller uncertainty matrix is selected, otherwise larger values are preferable.

Then the corresponding GPS reading is checked if it fulfills the validation criteria and if this is the case GPS edges are added to the graph. Validating is done by first checking the status information of the measurement and secondly by analyzing the first error and if it is within a threshold. This is necessary because the signal could be lost at any time or measurement errors with unnatural jumps might occur. GPS edges only capture the error between the Euclidean distance and the computed GPS distance by further using the *haversine formula*[3] [28], splitted in three Equations (5.1, 5.2, 5.3). The *haversine formula* calculates the great-circle distance between two points, hence the shortest distance over the earth's distance. In Equation (5.1) and Equation (5.2) auxiliary variables are computed. Almost all used variables belong to the GPS domain (latitude, longitude) or mentioned separately. It should be pointed out that GPS related variables need to be in radians. The final GPS estimated distance is then calculated together with the earth's radius in Equation (5.3).

The error function of GPS edges is therefore only a subtraction of both distances. Edges connecting odometry measurements belong to the group of 2D transformations with an error vector to incorporate position and yaw information. Whereas GPS edges only consider a numerical error, the difference of two distances. Every new and valid GPS reading is connected to the initial GPS pose. Moreover some additional GPS edges are created to support the mapping process by removing degrees of freedom of the overall graph.

Algorithm 1 also ensures the optimality of the constructed graph by iteratively optimizing (Line 27) it via Iterative Local Linearizations [29]. Additionally it constantly performs constraint tests (Line 25 and Algorithm 2), to find potential loop closures or to filter out misleading edges.

[3] $https://en.wikipedia.org/wiki/Haversine\_formula$

Possible loop closures are detected within the constraints check and rely on finding good potential poses. Poses near to the latest pose, but neither directly connected nor by traversing the graph and visiting some vertices, are considered potential loop closing vertices. After some graph related neighbors, of each potential pose, are added to the list we have to group them in reasonable sets by using the k-nearest neighbor algorithm. Finally we can obtain a loop closing result by running a more extensive scan matching search. The GPS outlier removing procedure is rather straight forward. First all edge errors are computed by considering a robust cost function (Pseudo Huber as suggested by Kümmerle et al. [11]) and then additionally normalized to their length. The errors are then comparable, otherwise very long ranges will more often have the largest error and are wrongly removed. We filter out a small percentage of the worst GPS edges after a certain amount of new edges are collected.

$$a = sin^2(\frac{\Delta lat}{2}) + cos(lat_1) \cdot cos(lat_2) \cdot sin^2(\frac{\Delta lon}{2}) \tag{5.1}$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)}) \tag{5.2}$$

$$d = R_{earth\_radius} \cdot c \tag{5.3}$$

5.4.2 Topological Hybrid Mapping

In this section we look into topological hybrid mapping or sometimes referred as map-graphs. This method combines the functionality of topological graphs (see Section 4.5 for additional information) but also incorporates occupancy grid maps (Kurt Konolige et al. [30] described this idea for the first time). In this thesis we suggest a modified version, namely an extended topological hybrid map. Our extended graph divides individual maps into multiple one due to accessibility. This means that every known pose can be reached from any other known pose position inside one map.

Algorithm 1: Graph-Based Mapping

```
Data: odometry_list ... all odometry recordings
Data: gps_list ... all GPS recordings
Data: laser_scan_list ... all laser range recordings
Result: pose_graph ... fully constrained pose-graph
begin
    initial_pose $\longleftarrow$ firstOdometry(odometry_list)
    initial_gps $\longleftarrow$ firstGPS(gps_list)
    graph.addInitial(initial_pose, initial_gps)

    foreach odometry in odometry_list do
        last_pose $\longleftarrow$ graph.getLastPose()
        new_pose $\longleftarrow$ ComputeEstimate(odometry)

        if (DistBetween(last_pose, new_pose) $>$ dist_tresh) or
        (AngleBetween(last_pose, new_pose) $>$ angle_tresh) then

            graph.addNewPose(new_pose)
            new_pose_laser_scans $\longleftarrow$ getPoseScans(laser_scan_list)

            matching_succeeded $=$ false
            if ScanMatchingSuccessful(new_pose_laser_scans) then
                matching_succeeded $=$ true
            end
            graph.addEdge(new_pose, new_pose_gps, matching_succeeded)

            new_pose_gps $\longleftarrow$ getPoseGPS(gps_list)
            if IsGPSValid(new_pose_gps) then
                graph.addGPSEdges(new_pose, new_pose_gps)
            end

            graph $\longleftarrow$ checkConstraints(graph)

            graph $\longleftarrow$ optimize(graph)

        end
    end
end
```

Algorithm 2: Check Pose-Graph Constraints

```
Input: pose_graph ... pose-graph
Output: pose_graph ... pose-graph
Function checkConstraints(pose_graph)
/* check for loop closures */
last_pose ⟵ graph.getLastPose()

poses ⟵ potentialLoopClosurePoses(graph)

poses ⟵ addNeighbours(poses, graph)

p_groups ⟵ findGroups(poses, graph)

closest_poses ⟵ closestPoses(p_groups, graph)

foreach pose in closest_poses do
    if ScanMatchingSuccessful(pose, last_pose) then
        graph.addLoopClosingEdge(pose, last_pose)
    end
end

/* check for GPS outlier */
gps_edges ⟵ getAllGPSEdges(graph)

gps_edge_errors ⟵ computeGPSEdgeErrors(graph)

gps_edges, gps_edge_errors ⟵ sort(gps_edges, gps_edge_errors)

if gps_edges.size() > threshold then
    for i ← 0 to gps_percentage do
        graph.removeGPSEdge(gps_edges[i])
    end
end
```

Therefore we distinguish between occupancy grid maps which can be fully traversed and grid maps which can be only partially visited by the robot in reality.

To visualize the difference Figure 5.13b shows an ordinary local grid where all poses within the grid are used for rendering the occupancy grid map. In contrast Figure 5.13c relates to the extended map and Figure 5.13d is the resulting new grid. For building such a graph a fully consistent pose-graph is essential, failures or inconsistencies will cause severe problems.

Algorithm 3 captures one approach to produce our modified topological hybrid map. The initial step is responsible for providing at least one grid for every pose-graph pose. The iteration sequence through the pose-graph is prescribed and mandatory by the previous creation process. The initial covering process is described in more detail by Algorithm 4 and is rather straight forward, except the maintenance of an additional **covered list** which functions as a map for saving correspondences between poses and grids.

Now the topological hybrid graph can be extended by checking the accessibility (Algorithm 5). While processing through the whole topological graph we render a complete temporary map for each grid (Line 4), on which we perform all accessibility checks. These grid maps can easily be created with the earlier constructed **covered list**. The grid map is computed by a standard ray tracing procedure using all the laser scans gathered from all covered poses.

To validate traversability we need to look at each covered pose from the **covered list** and test if planning to the pose at the grid center is possible. This is further visualized by Figure 5.12. The blue rectangle represents for the original local grid and the dashed one is newly created. Green circle denote poses and number 4 to 6 are truly related to the grid, however 34 to 36 are not accessible poses and therefore do not relate to that grid.

We now maintain two separated map structures. One is saving poses responsible for the final map creation and called **used list**. The second map (**unconnected poses**) is a temporary list for all poses which are neither connected to the grid center nor to any other grid. From Line 17 to 26 in Algorithm 5 we process through the **unconnected poses** list until every pose is validity used by a local map. Then we need to find all covering poses for all newly created grids (Line 28).

Due to the nature of the pose-graph, which is per definition continuously connected, a resulting graph can always be created. There exists only one exceptional case, where the size of the local grid is large enough to enclosure the complete pose-graph.

After all local grids are ready for deployment the process of edge creation is clean and easy. The process is described by Algorithm 6 in more detail. Here we iterate over all grids and compare their **used list** to one another and if a similar pose occurs in both lists we add a new edge connecting these grids to the topological hybrid graph. At the end of the graph creation process we render and save all occupancy grid maps and output the final topological hybrid graph.

Algorithm 3: Topological Hybrid Mapping

```
Data: pose_graph ... fully constrained and consistent pose-graph
Result: tp_graph ... topological graph
Result: local_maps ... local occupancy grid maps
1  begin
2      tp_graph ⟵ CoverPoses(pose_graph)
3
4      tp_graph ⟵ CheckAccessibility(tp_graph, pose_graph)
5
6      tp_graph ⟵ CreateEdges(tp_graph)
7
8      BuildAndSaveMaps(tp_graph)
9      SaveTopologicalGraph(tp_graph)
10 end
```

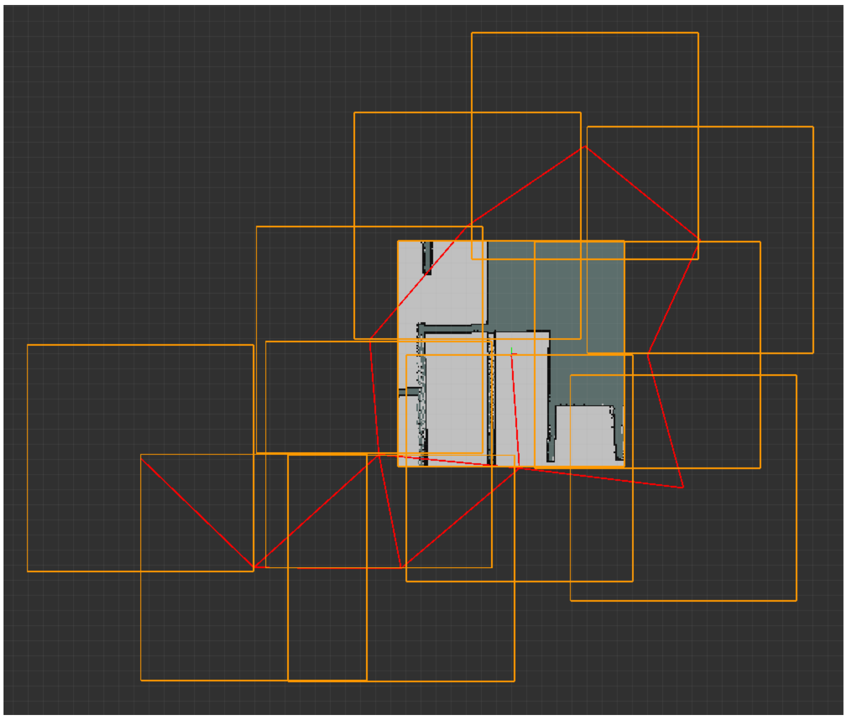


Figure 5.11: Sample topological hybrid map. Orange rectangles outline local grids, edges which visualize connections between grids are colored in red. In the center one sample local grid is rendered.

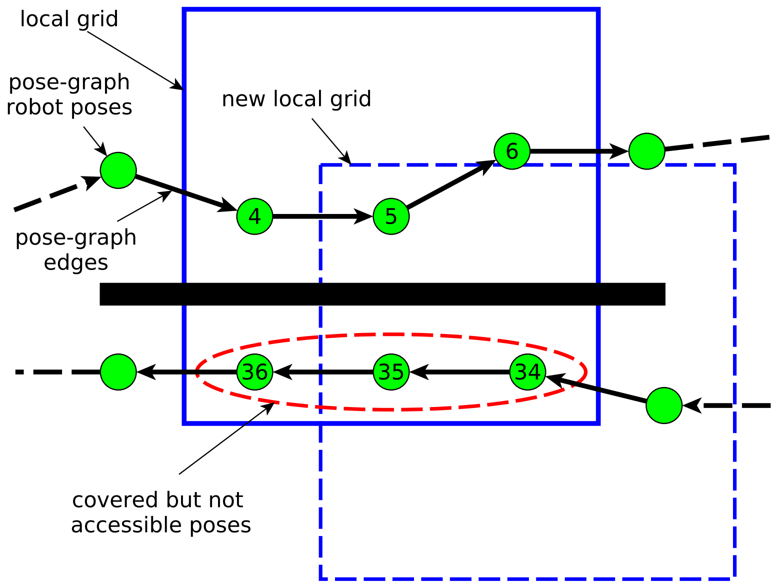
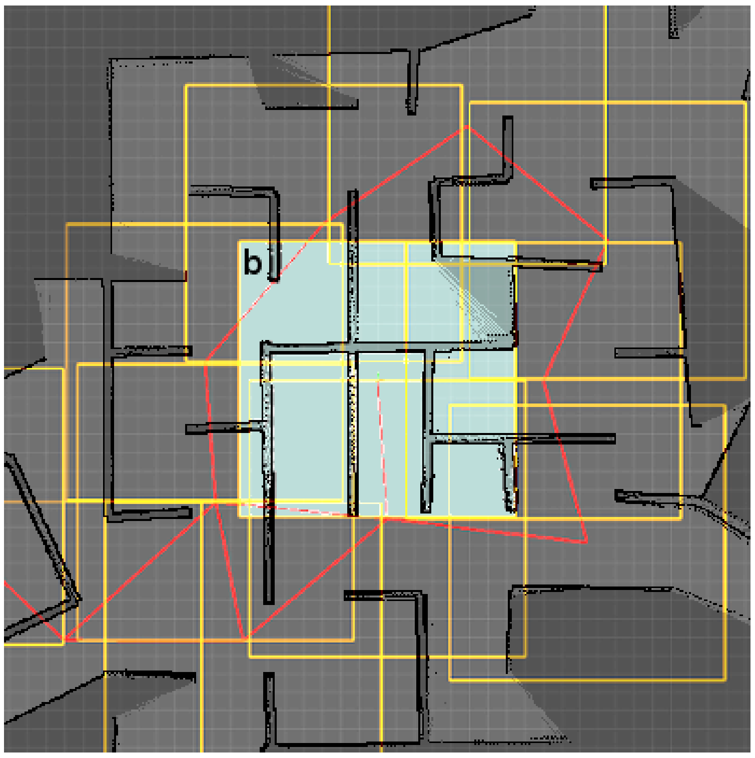
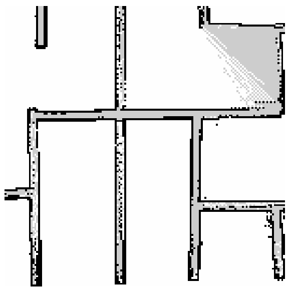


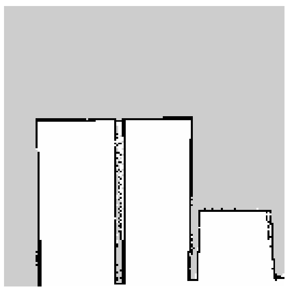
Figure 5.12: Accessibility check of all covered poses within a local grid of an early topological graph. A new local grid is created because covered but inaccessible poses where found (no valid path between pose e.g. 35 to pose 5, which is the center of the local grid).



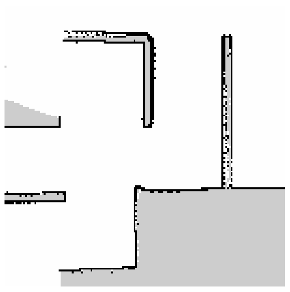
(a) Topological hybrid graph overlays the map.



(b) Local grid, ordinary graph.



(c) Local grid, extended graph.



(d) One newly created local grid.

Figure 5.13: Topological hybrid graph with underlaying map. Difference between ordinary and extended method is visualized in Figure 5.13b and Figure 5.13c.

Algorithm 4: Cover Poses

```
Input: $pose\_graph$ ... pose-graph
Output: $topological\_graph$ ... topological graph
Function CoverPoses($pose\_graph$)
topological_graph ⟵ []
foreach pose in pose_graph do
    not_covered = true
    foreach grid, center in topological_graph do
        if $distance(center, pose) < threshold$ then
            not_covered = false
            topological_graph.insertCoveredList(grid, pose)
        end
    end

    if not_covered then
        topological_graph.createGrid(pose)
    end
end
```

5.5 Localization

For localization we use a sample-based approach known as the particle filter as explained in Section 4.4. As explained before, such a particle filter is able to cope with all kind of non-Gaussian error hypothesis unless enough particles are involved.

The current setup uses 1,000 to 5,000 particles due to its adaptive character. As for the observation model, which is crucial for computing the likelihood for each particle given the laser measurement, we decided to pick the `range finder model`. This model can explicitly deal with unexpected objects, failures and random measurements.

The other measurement source is the combined odometry, provided by the UKF (Section 5.3) and relevant for moving all particles accordingly to the motion model. AMCL applies the `sample motion model` for adding odometry measurements to the hypothesis, so noise and movement characteristics are better captured.

Algorithm 5: Check Accessibility

```
Input: pose_graph ... pose-graph
Input: tp_graph ... topological graph
Output: tp_graph ... topological graph
Function CheckAccessibility(tp_graph, pose_graph)
foreach grid, center in tp_graph do
    covered_list ⟵ tp_graph.getCoveredList(grid)
    map ⟵ buildCompleteMap(grid, covered_list)

    unconnected_poses ⟵ []
    foreach covered_pose in covered_list do
        if planningPossible(map, covered_pose, center) then
            tp_graph.insertUsedList(grid, covered_pose)
        else
            if tp_graph.findInUsedList(covered_pose) $\neq$ true then
                unconnected_poses.insert(covered_pose)
            end
        end
    end

    while unconnected_poses $\neq \{\}$ do
        pose ⟵ unconnected_poses.pop()
        grid, center ⟵ tp_graph.addNewGrid(pose)
        foreach pose in unconnected_poses do
            if planningPossible(map, pose, center) then
                tp_graph.insertUsedList(new_grid, pose)
                unconnected_poses.remove(pose)
            end
        end
    end

    tp_graph ⟵ GetNewGridsCoveredList(tp_graph, pose_graph)
end
```

Algorithm 6: Create Edges

```
   Input: $topological\_graph \ldots$ topological graph
   Output: $topological\_graph \ldots$ topological graph with edges
1  Function $CreateEdges(topological\_graph)$
2  foreach $grid\_1$ in $topological\_graph$ do
3      $used\_list\_1 \longleftarrow topological\_graph.getUsedList(grid\_1)$
4      foreach $grid\_2$ in $topological\_graph$ do
5          if $grid\_1 \neq grid\_2$ then
6              $used\_list\_2 \longleftarrow topological\_graph.getCoveredList(grid\_2)$
7              if $(used\_list\_1 \cap used\_list\_2) \neq \{\}$ then
8                  $topological\_graph.addEdge(grid\_1, grid\_2)$
9              end
10         end
11     end
12 end
```

The particle filter requires a map to estimate the robot pose against it and one is provided by picking the most probable local grid given the topological hybrid graph. Algorithm 7 is responsible for finding the most likely local grid. At the start we assume a known robot position and thus a specific local grid. Together with the current pose estimation we constantly check if the robot is leaving the local grid. Only a limit set can be considered as candidates, this is because of the given graph topology and illustrated in Figure 5.14.

The next local grid is then picked depending on the shortest Euclidean distance (colored in red). In our implementation the distance to the new local grid has to be one fourth, of the distance to the current grid closer. This ratio assumption proved to work pretty well in our field tests. Within the Algorithm 7, this is notated as the `relation` in Line 8. This prevents too early map switches, especially when the robot is driving along the switching boarder.

It is advisable to do the map change for a particle filter only if the majority of the localization distribution is covered by the next grid. Overlapping between neighboring grids is therefore an important property for a topological hybrid graph.

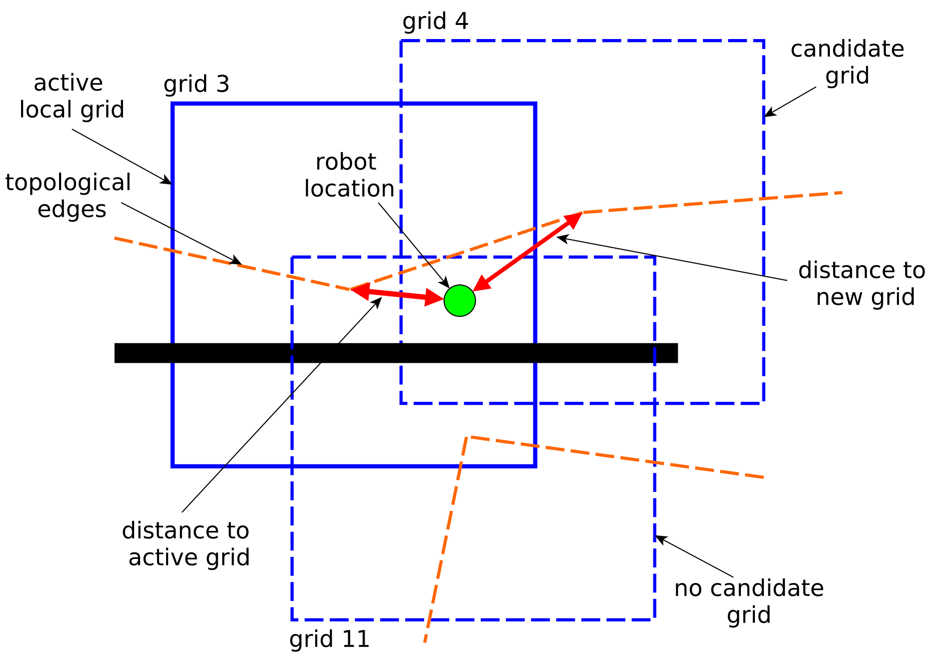


Figure 5.14: Map switching by choosing candidates due to the connectivity (dashed orange lines) of the topological graph. Grid 4 is the only candidate and finally picked if the distances, highlighted in red, meet a certain relation.

In our current implementation we considered a local grid size of 60 by 60 meters mainly because localization depends on enough local features and the average measurement distance of the laser range finders do not exceed 30 meters.

5.6 Planning

The planning architecture is build up on three abstraction level. The lowest and second lowest level is covered by the `Move-Base` (see Chapter 4 for more information) and the concept is described

Algorithm 7: Get Best Grid

```
Input: $tp\_graph$ ... topological graph
Input: $robot\_pose$ ... current robot localization
Output: $best\_grid$ ... best grid
Function GetBestGrid($tp\_graph, roadmap, robot\_pose, goal$)
shortest_dist ⟵ computeDistance(tp_graph, robot_pose)
best_grid ⟵ tp_graph.getCurrentGrid(robot_pose)
neighboring_grids ⟵ tp_graph.getAdjacentGrids(robot_pose)

foreach neighbor in neighboring_grids do
    distance ⟵ computeDistance(robot_pose, neighbor)
    relation = distance/(shortest_dist + distance)

    if insideGrid(robot_pose) and relation < threshold then
        shortest_dist = distance
        best_grid = neighbor
    end
end
```

in Section 5.6.3. The highest level employs A* search on a precomputed roadmap, which is an abstracted model of the environment. This layer outputs a way-point sequence to the goal and hands each point over to the next lower planning level, the `Move-Base`. Due to the layered architecture we unfortunately compute sub-optimal paths (approximately 10% longer, while using A* search algorithm on the complete occupancy grid map). But as Konolige et al. [30] stated, the calculation time is in exchange several orders of magnitude faster.

5.6.1 Roadmap

In our current design we decided to introduce an abstracted planning layer to simplify and thereby speed up the whole planning processing. We therefore implemented a roadmap, similar to on proposed by Kurt Konolige et al. [30]. Algorithms like A* require a significant amount of time (several seconds and even longer) on very large sized grid maps for planning, which is not always sufficient for real-time solutions. As earlier mentioned in Section 4.5 hybrid systems benefit from locally navigating on the occupancy grid maps and producing near-optimal plans on the topological graph. The roadmap serves as a connection port between these planning layers and provides way-points as well as edges with adequate traveling costs, so that planning on this type of graph is possible. One essential assumption is that two way-points are only connected if and only if there exists a possible path in the map between them. The main difference to the implementation of Kurt Konolige et al. [30] is the way how he offered the locations of every way-point. He suggested providing way-points, only related to a specific grid, whereas in our implementation way-points belong to the world and consist of global coordinates. This property as some benefits and will be discussed later.

The development of the roadmap is described in Algorithm 8. When building the roadmap one has to decide to either use, to build up on, or to exclude the earlier created pose-graph. All decisions have particular benefits like when using the pose-graph only method the robot will stay on an already familiar terrain which is especially helpful in dangerous environments. Whereas the hybrid variant allows more flexible routing and is useful in open areas with a lot of environmental changes. The main roadmap development is presented in Algorithm 8 and passes the simple sequence of first placing all way-points (whether with or without pose-graph) and then connecting those way-points with edges.

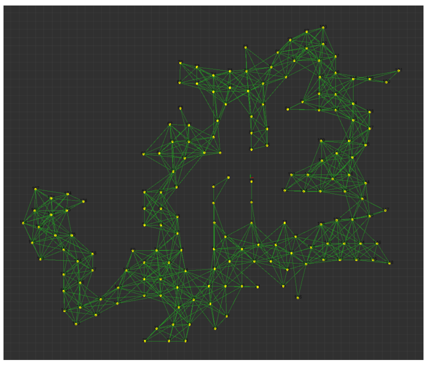
In Algorithm 9 the way-point placement based on the pose-graph is described. While iterating through the pose-graph new way-points are always added whenever the current location near that suggested position is still way-point free. This complies with a minimum spread out distance. Every way-point relates to a local grid and we therefore need to find the correct corresponding one. This is achieved by looking at all covering grids, but because we assume an extended topological hybrid map (see Section 5.4.2) we have further to assure that the relevant position is within a known area in the local occupancy grid.

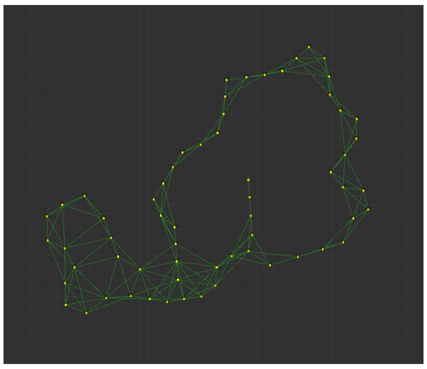
The layout process without any pre-knowledge, except the previous rendered occupancy grid maps is as follows. Every single local grid is separately loaded from the hard disk and then identically processed. First potential way-points are spread out with a grid pattern. The local occupancy map is then conservatively inflated, so that obstacles appear larger then in reality but still leave enough space for the placement. Next we look at every potential point and validate if this position is obstacle free given the inflated map. If that's the case, this point becomes a valid new way-point, otherwise we perform a spiral search around that point until we either find free space or we reach a cut off distance and dismiss it.

The final step is to connect way-points by edges comprising of proper cost values. To start the edge generation every local occupancy grid is loaded again and the obstacles are more radically inflated than before. Then we look at all grid relevant points including way-points from neighboring grids unless they are not covered. The local A* planner is then called for every potential pair (points within a certain distance), to find a path between them. The traveled distance is then taken as the cost for the newly created edge. Although A* is rather fast on such short distances it takes some time to compute. However this is the only way how to guarantee that a valid path exists. The computed edge cost relates to the real travel cost. This time consuming step can be computed completely offline and saves a lot of processing power while running online (see Section 7.4 for an evaluation).

5.6.2 Topological Planning and Execution

The topological planner is the highest level in our planning hierarchy. It is responsible for computing a path sequence to the goal using the topological hybrid graph and the roadmap (Figure 5.16). In Algorithm 10 the planning execution is outlined. The user has to input a specific goal way-point and the planner is then executing as long as the goal is not reached.


(a) Generated with maps only.


(b) Only using the pose-graph.

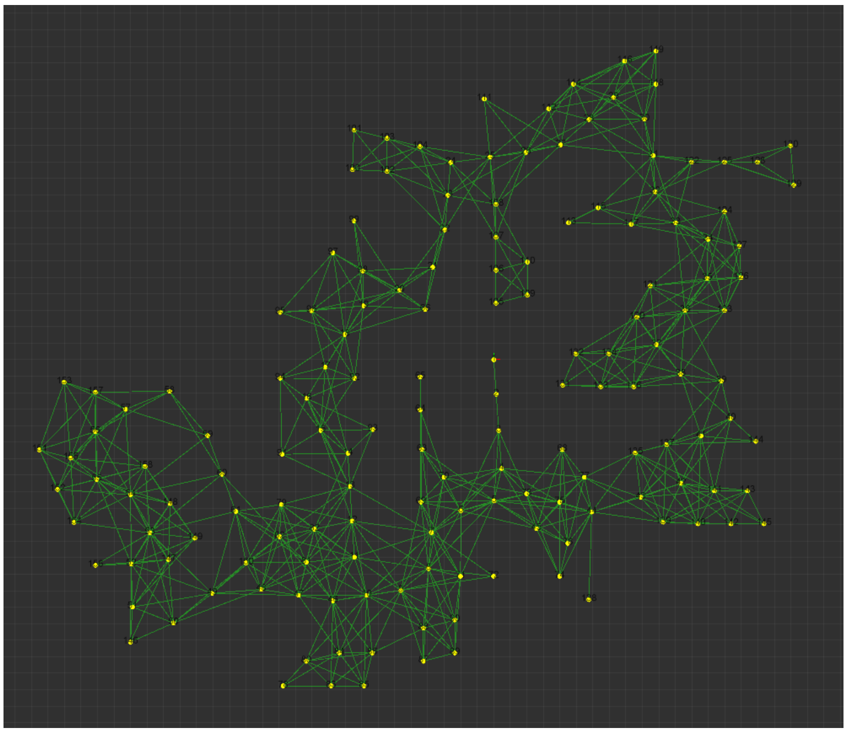

(c) Created with both, pose-graph and maps.

Figure 5.15: Roadmap generation with different inputs. Figure 5.15a creates a roadmap by using the maps only, whereas Figure 5.15b is just utilizing the pose-graph. Finally a combination of both is visualized in Figure 5.15c.

Algorithm 8: Roadmap Generation

```
Data: pose_graph ... pose-graph
Data: tp_graph ... topological graph
Data: local_maps ... local occupancy grid maps
Result: roadmap ... roadmap
begin
    roadmap ⟵ []
    if pose_graph_enabled then
        roadmap ⟵ WithPoseGraph(tp_graph, pose_graph, local_maps)
    end

    if pose_graph_only_disabled then
        roadmap ⟵ CreateWaypoints(roadmap, tp_graph, local_maps)
    end

    roadmap ⟵ CreateEdges(roadmap, local_maps)

    SaveRoadmap(roadmap)
end
```

Algorithm 9: Way-point layout with pose-graph

```
Input: pose_graph ... pose-graph
Input: tp_graph ... topological graph
Input: local_maps ... local occupancy grid maps
Output: roadmap ... roadmap
Function WithPoseGraph(tp_graph, pose_graph, local_maps)
foreach pose in pose_graph do
    if NoPointNearPosition(pose, roadmap) then
        foreach grid, center in tp_graph do
            if CoversPoint(pose, center) and
            InsideKnownArea(pose, local_map) then
                roadmap.addWaypoint(pose, grid)
                break
            end
        end
    end
end
```

The *CreatePlan* routine (in Line 4) first searches for the start and end point within the roadmap and then extracts a way-point sequence to the goal by applying A* search.

The plan execution is now processing every way-point of the plan until the goal is finally reached. For every point we have to check if the current local grid is still covering it, otherwise the `GlobalPlanner` of the lower-level is not able to come up with a valid plan. At that point we pick a grid which is covering both, the current robot pose and the next way-point. In the case the local grid is valid for that specific way-point, it can be submitted to the next planning layer, the `Move-Base`. While waiting for the low-level planner to finish, we can frequently check the distance between the robot and the temporary goal and simply proceed with the next way-point if the robot is close enough. This speeds up the path execution a lot.

5.6.3 Global/Local Planning

This layer of the planning hierarchy comprises two stages, the so called global and local one. This layer depends on the high-level planner from Section 5.6.2 for receiving goals. In this thesis we make use of the well-known `Move-Base` to solve the lower navigation task at that level (described in Chapter 4). The navigation sequence is executed as follows: The high-level planner is publishing a goal within the active local grid, then the `GlobalPlanner` computes a path which is then fed to the `LocalPlanner`.

The `GlobalPlanner` requires additional knowledge in the form of a costmap. To come up with an useful and risk free plan. Costmaps are similar to normal occupancy grid maps except that they include more information about robot or maneuvering relevant obstacles. In our case the costmap is additionally informed about vegetation and relatively small sized obstacles and gaps. Important for making planning decisions and traversing.

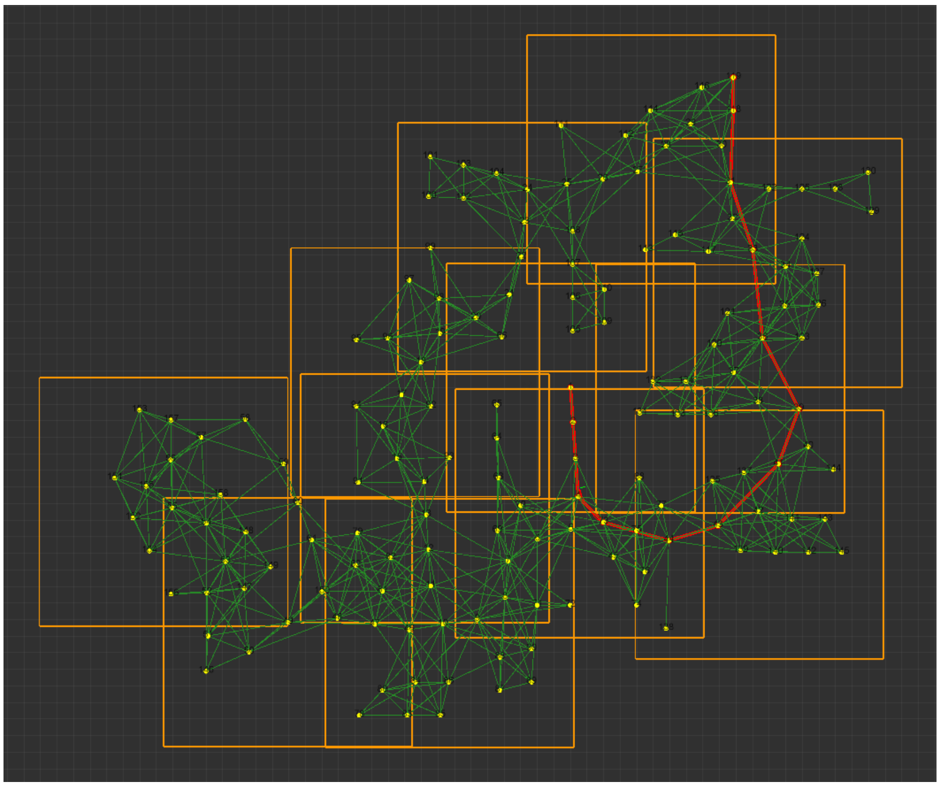


Figure 5.16: Planned path from roadmap point 1 to 139 in red color. Green and yellow relate to the roadmap and orange rectangles symbolize grids from the topological hybrid graph.

Algorithm 10: Execute Topological Plan

```
Data: pose_graph ... pose-graph
Data: tp_graph ... topological hybrid graph
Data: roadmap ... roadmap
Data: localization ... current robot localization
Input: goal ... goal way-point
begin
    goal_reached ⟵ false
    while goal_reached = false do
        plan ⟵ CreatePlan(tp_graph, roadmap, localization, goal)

        foreach way_point in plan do
            if insideGrid(way_point) = false then
                grid ⟵ GetBestGrid(way_point, localization)
                switchGrid(grid)
            end

            move_base_client.SendGoal(way_point)
            while move_base_client.CheckState = pending do
                if distance(way_point, localization) < threshold then
                    break
                end
            end

            if way_point = goal then
                goal_reached ⟵ true
            end
        end
    end
end
```

Costmaps are build up in a layer like structure and the lowest layer usually is a static map but in our case this map can change at any time. The second layer maintains an already provided voxel grid, to deal with obstacles recognizable through the horizontal lasers. The last layer is responsible for inflating everything from layers below, so that the robot keeps a reasonable distance to obstacles and is therefore the last layer. The planner is then using this costmap and applying A* search as described in Section 4.5 to determine the route to the goal.

The last planning level is called `LocalPlanner` and used to follow a generated path by computing possible trajectories and translating them into velocity commands. We decided to pick the dynamic window approach (DWA) for computing trajectories. It is the recommended choice, because it consumes less computing power than similar methods. This method requires a costmap for computation and we applied the same layer topology as before. The horizontal lasers as well as the terrain analyzing module are providing additional information to two separated layer. The only difference is the used map size. Whereas the `GlobalPlanner` depends on the complete local grid, the `LocalPlanner` only needs a small window of about 6 by 6 meter to function properly. This allows the planner to speed up the map building process which then enables real-time planning capabilities.

5.7 Terrain Analysis

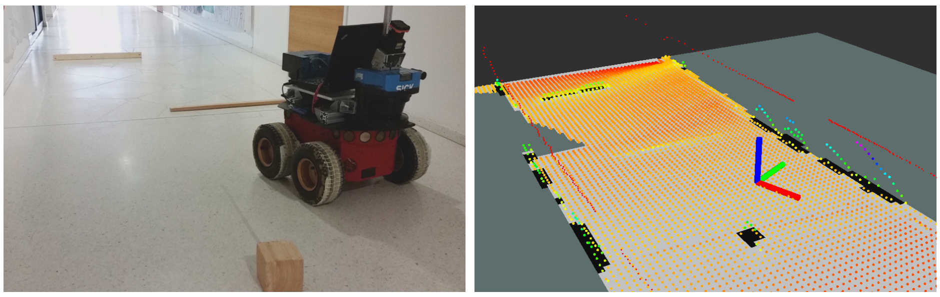
Terrain analysis supports two tasks. The first task copes with obstacle avoidance. In order to prevent damages of all kind. The second task deals with terrain classification. To distinguish between "good" terrain where the robot is permitted to drive or "bad" terrain which should be possibly avoided. As already discussed in Section 3.4, these skills heavily depend on the working environment as well as the robot capabilities.

In the overall design a reliable obstacle identification beyond the horizontal laser scanners is mandatory for any outdoor navigation. For that reason the down facing Hokuyo (more information on hardware decisions in Section 5.1.1) fills the important role to gather enough information to create a detailed elevation map. An elevation map or height map is a 2.5D dimensional representation of the environment. The map maintains a flat cell grid and every cell comprises a height estimation of the terrain. An extended elevation map was proposed by Pfaff et al. [15], which is able to detect tunnels e.g. passages under bridges. In Figure 5.17 sample measurements were taken indoor, to see how the system reacts to various obstacle heights. The resulting costmap using only the elevation map as input is visualized in Figure 5.17b. Black denotes non-traversable obstacles and bright gray color marks free space. The color reflects the estimated elevation for every point of the height map.

To obtain the map we utilize a Kalman filter to estimate the height and its standard deviation in each cell. When adding a new measurement z_t with standard deviation σ_t we use Formula (5.4) and Formula (5.5) to update the corresponding cell. The measurement z_t is produced by the declined laser scanner and further transformed with the orientation provided by the IMU. We additionally apply a sensor model, in which the variance increases linear to the measurement distance. This smooths out measurement errors, due to traveling on uneven terrain. The gradient of the height map is constantly updated and if cells exceed a maximum value it is treated as an insurmountable obstacle, similar to walls.

$$\mu_{1:t} = \frac{\sigma_t^2 \cdot \mu_{1:t-1} + \sigma_{1:t-1}^2 \cdot z_t}{\sigma_{1:t-1}^2 + \sigma_t^2} \tag{5.4}$$

$$\sigma_{1:t}^2 = \frac{\sigma_{1:t-1}^2 \cdot \sigma_t^2}{\sigma_{1:t-1}^2 + \sigma_t^2} \tag{5.5}$$



(a) Picture of robot while analyzing the terrain. (b) Result of the computed elevation map.

Figure 5.17: Creation of an elevation map. Figure 5.17b shows the resulting map and Figure 5.17a the corresponding scene in reality.

The second terrain analyzing task takes care of classification of the ground. In this thesis it sufficed to distinguish between vegetation (especially flat vegetation like grass) and non-vegetation representing the road surface. Luckily plants exhibit specific laser reflection properties different to streets. This is further explained by Wurm et al. [31]. Therefore the classification can easily be done by making use of the laser reflectivity which is additionally measured by the down facing Hokuyo laser.

Figure 5.18 presents the correlation between range and intensity measurements for grass and streets. The closer the measured distance, the harder the distinction. The blue separation line was conservatively placed, so that street detection is more preferred. This classifier can also be trained by applying machine learning tools like explained in the book of Barber et al. [32].

There are several things which can be done to boost up the classification accuracy. But the far simplest one is to increase the range measurements by reducing the angle of the down facing laser. Although we obtain many wrong classifications, simple post-processing steps like applying morphological operators strongly improve the result.

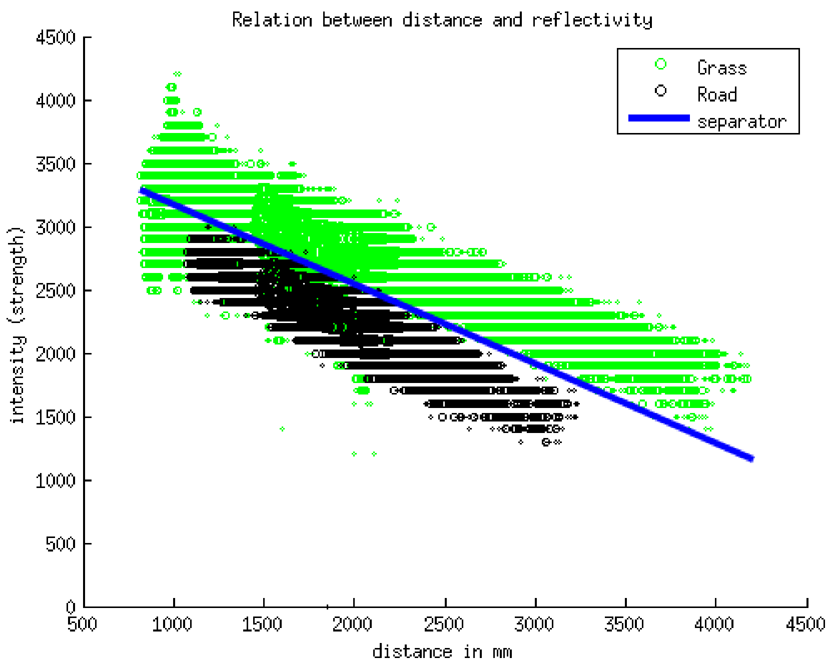


Figure 5.18: The diagram shows the relation between range and intensity measurements for grass and road. Green dots represent grass measurements and black correspond to the road. The incidence is visualized by point density. The blue line shows a possible separation line for classification.

Beside the elevation map we maintain a second grid for classification. When building the costmap layer for navigation (see Section 5.6.3) the previous explained gradient map and the classification is used. The costmap layer is build up by analyzing every cell individually. What causes the highest costs at one point counts for the costs inside the costmap layer (for more details see Section 6.3.2).

6 Implementation Details

In this chapter we talk about implementation details related to the concept presented in the previous chapter. The overall software runs on a Linux operating system, with ROS Indigo as middle-ware, which is further described in Section 4.1. At the very beginning of this chapter we examine the graph-based mapping as well as the topological hybrid mapping. Then we show some details on our planning approach utilizing the roadmap. Finally we conclude with information about the terrain analysis module.

6.1 Mapping

The process of map creation is split in two separated tasks. The first one is graph-based mapping, described in Section 5.4.1 and responsible for directly interpreting the sensor data to construct a consistent pose-graph. This was mainly achieved with g^2o. The second task is then capable of forming the pose-graph into a topological hybrid graph, the key concept behind is formulated in Section 5.4.2.

As mentioned in Section 5.4 we recorded all sensor measurements for various reasons. This can easily achieved by using ROS integrated solutions the `ros bag file system`. For that reason some implementations might also work in an online manner, as long as enough computational power is available.

It should be mentioned that we decided not to utilize any database, although we heavily thought about it. The data is now saved to text files because then the information can easily be modified with any simple editor. For the final distribution these files are loaded into the memory in no time. In addition this technique makes the need for any other software dependency obsolete. The constructed pose-graph, the topological graph and the roadmap can be separately saved. Further details on the data structures can be looked up in the following matching sections.

6.1.1 Graph-Based Mapping

The main principle behind building a pose-graph is already described in Section 5.4.1 and in this section we will only discuss details of the realization. The graph-based mapping functionality is fully implemented within one ROS package (*pose_graph_mapping*). For constructing the graph we need two main class types, namely the `VertexSE2` and the `EdgeSE2`. Both classes are type of SE2 and furthermore a hypergraph element (see Figure 6.1 for details) to simply provide a dimension of three to include information on Δx, Δy and $\Delta\Theta$. The g^2o framework already provides several SLAM relevant and optimized object types. In addition tutorials on how graph-based slam should be implemented are prepared. The original framework is linked to the ROS package and new class implementations which rely on framework types must be additionally registered by calling a factory class provided by g^2o. Otherwise the framework does not support these new defined class types.

Figure 6.1 shows the relationship of the individual class types directly used with g^2o. For this thesis we added a new `RobotGPS` type for saving GPS measurements to every vertex. Due to the slow measurement frequency of the GPS not every graph vertex will have a valid reading. Thus only the chronologically closest vertex has the valid measurement attached and following vertices get the same measurement but with an invalid status information added. So the status information is only true (which denotes a valid reading) if it was initial and it is a new measurement. In our current implementation `RobotGPS` only comprises the GPS

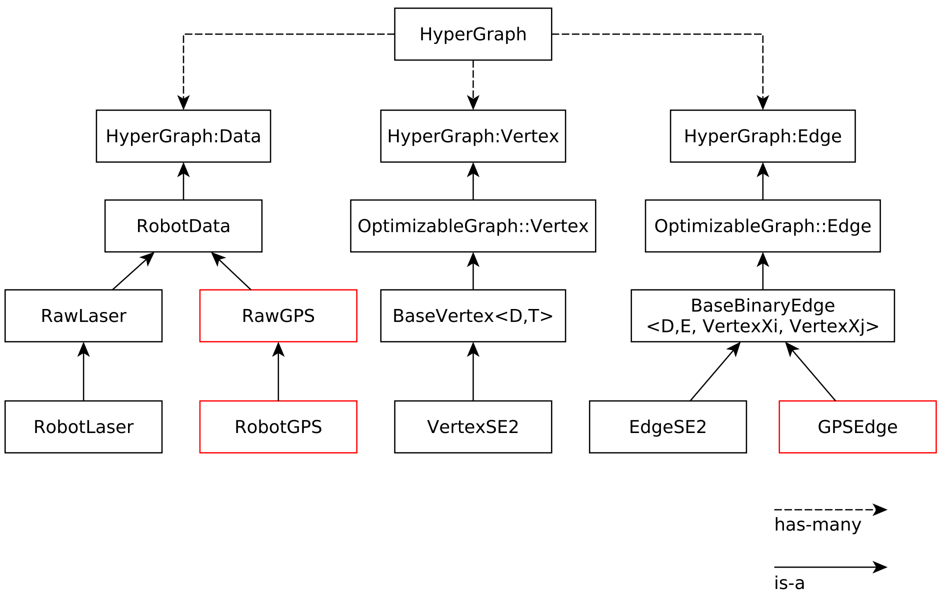


Figure 6.1: Class diagram of existing and custom (colored in red) types for *g*^{2}*o*.

information longitude, latitude, altitude and the `status` bool to specify the validity of the data.

We also introduced a `GPSEdge` for establishing the connection between vertices with valid GPS measurements. This edge is also based on the `BaseBinaryEdge` but only features one numerical value and is thus one dimensional in contrast to the three dimensional `EdgeSE2`. Within this class we added the functionality to directly compute the distance between two GPS readings with the *haversine formula* which was explained in Section 5.4.1.

The node consists of only one process which contains the main thread and the previous described *g*^{2}*o* relevant classes. Moreover some helper classes are implemented for the sake of clarity. The main loop is running with a frequency of 10 Hz and is successively building up the pose-graph by receiving odometry *nav_msgs::Odometry*, laser scan *sensor_msgs::LaserScan* and GPS messages in type of *sensor_msgs::NavSatFix*. Odometry informations are published with a frequency of 50 Hz even though laser scans are only published with 10 Hz. Therefore running the main loop at the same frequency as the laser scans causes new scans for at least every vertex. Furthermore we acquire the difference between two successive odometry messages and valid GPS message are only attached to appropriate poses. For this reason we can neglect any data synchronization. These messages are received via ROS topics and for better debugging we output every change of the pose-graph on a topic with the type *visualization_msgs*::*Marker*. This allows us to see all vertices with connections. Especially GPS related edges are colored differently. The graph optimization is running pretty fast and can therefore be done for every newly added vertex. At the moment we run the optimizer for five iterations.

The pose-graph can easily be saved at the end of the process by using the provided functionality by *g^2o*. Then the graph will be saved in a text file like shown in Listing 6.1.

Listing 6.1: Saved pose-graph

```
VERTEX_SE2 id x y theta                          # robot pose vertex
ROBOT_GPS longitude latitude altitude            # vertex related GPS
ROBOTLASER laser_data                            # vertex related laser
...                                              # all vertices
EDGE_SE2 vertex_id_1 vertex_id_2 error variance  # normal edge
... or ...
GPS_EDGE vertex_id_1 vertex_id_2 error           # GPS edge
...                                              # all edges
```

6.1.2 Topological Hybrid Mapping

After building the pose-graph we can start creating the topological hybrid graph. The *topological_mapping* package provides the functionality to create, save, and maintain such a graph. In Figure 6.2 an overview of the individual module classes and how they are interconnected and share common resources is visualized. The mapping process is loading the previous saved pose-graph by *g^2o* and is therefore only able to build the graph once. The map rendering process is realized by utilizing the *occupancy_grid_utils* package. This graph is then distributed over ROS by means of visualization tools (*visualization_msgs*::*Marker*) and a special ROS message (Listing 6.2). The message contains not only the graph but additional information about the grid size and the resolution.

If desired, the graph can then be saved similar to what *g^2o* provides. The saving module is subscribing to the topic where the graph message is distributed and saves it to a text file in the same structure as the message. Each created occupancy grid is numbered in ascending order and saved as an image, together with a corresponding parameter text file. This technique is equivalent to the functionality of the *map_server*, another ROS package and can therefore be used to load individual maps.

Listing 6.2: ROS topic message of the topological hybrid graph

```
topological_nav_msgs/TopologicalMapNode[] nodes # list of all grids
  uint32 id
  float32 x_size
  float32 y_size
  float32 resolution
  geometry_msgs/Pose center                     # position of grid origin
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
topological_nav_msgs/TopologicalMapEdge[] edges # list of all edges
  uint32 id
  uint32 src
  uint32 dest
```

For using the topological hybrid graph we offer a maintenance module (*TopologicalManagerNode*), which loads the graph and publishes it as well as the local grids frequently. Additionally it manages the local grids by loading it directly from the memory and provides a service for changing. The service only requires the grid number and returns an error if the grid is not available due to various reasons.

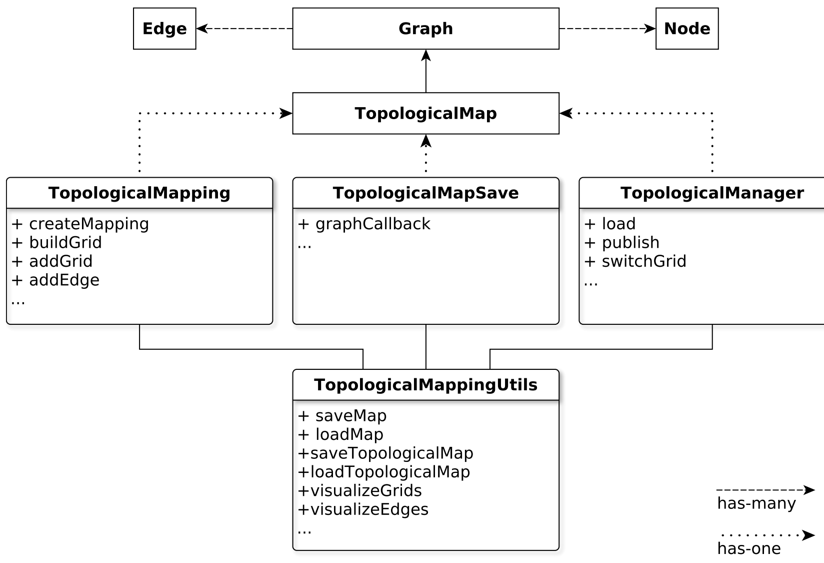


Figure 6.2: Overview of the available topological mapping classes.

6.2 Planning

This section comprises only the high-level planning implementation, whereas the low-level hierarchy planner was not part of this thesis. The real planning work is done inside the topological planning section (Section 6.2.2) but in order to work, the robot position (see Chapter 5 for more information) and a pre-computed roadmap is required, which is described next.

6.2.1 Roadmap

The roadmap is also implemented within one ROS package, called the *topological_roadmap*. The created roadmap is published with a special designed message which is described in Listing 6.3 and can be visualized as well. This message comprises all way-points and their related grid. This is later useful to pick the correct local grid. As described in Section 5.6.1 every way-point is located in the same global frame (in our case it is called the map frame). This has the advantage that every way-point could directly be transformed to real world GPS coordinates.

The actual generation process is not started until receiving the topological hybrid graph for the first time. Then, depending on the user input, it loads the corresponding pose-graph file and adds all necessary way-points. The program needs to load individual maps from time to time and therefore refers to the library provided by the topological hybrid mapper. The way-point generation with maps strongly builds up on the functionality offered by the *occupancy_grid_utils* package. This package is able to adapt maps on the fly like inflating and also includes basic planning capabilities. For example the process of spreading out potential way-points is checking reasonable positions on inflated grids, this ensures a safety distance so that points are far enough away from obstacles.

The final roadmap can be saved, loaded and then used very similar to the topological hybrid graph implementation (Section 6.1.2).

Listing 6.3: ROS topic message of the roadmap

```
topological_nav_msgs/RoadmapNode[] nodes       # list of way—points
  uint32 id
  uint32 grid                                  # related to that grid
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
topological_nav_msgs/RoadmapEdge[] edges       # list of edges
  uint32 id
  uint32 src
  uint32 dest
  float32 cost
```

6.2.2 Topological Planning

The topological planning or high-level planner is providing a way-point sequence to the global goal for the low-level planner `GlobalPlanner`. Therefore it requires the current robot localization and the roadmap. At the moment the user assigns the goal by using a defined action comprising a goal way-point id or a roadmap independent pose. Furthermore it is responsible for switching the maps accordingly and thus depends on the topological hybrid manager or mapper.

The planning procedure is illustrated in Figure 6.3. The planning module is only enabled if `Move-Base` is ready and the topological hybrid map as well as the roadmap has been received. Then the planning thread is initiated and incoming goals are further transferred to the thread execution. After the planner receives the robot position a plan to the goal is computed by utilizing the roadmap and the Boost graph library[1]. The actual execution of the plan is done in a straight forward manner. The topological planner thread is sending every way-point successively to the `Move-Base`. The `Move-Base` provides an action *move_base/goal* for receiving navigation goals. The topological planner always sends the following way-point shortly before the robot is reaching the current way-point. Hence `Move-Base` is only finishing the last and final goal.

The map switching is one crucial part of the whole system. To perform reasonable decisions, Algorithm 7 described in Section 5.5, needs information about the robot position, the current and the following low-level goal. All this is within the scope of the topological planner. The `GetBestGrid` algorithm ideally picks a local grid where the low-level goal and the robot are covered. In some cases this is not always possible. If for example the way-point space of the roadmap is too large in proportion with the overlap of the local grids. Then the second best choice, at least the robot is covered, is chosen. After a short amount of time the robot has then driven a little further and a better grid is available. Nevertheless we recommend changing such configurations. In the current setup we assume the first grid is the start but this is easily changed by launch parameters.

[1] *http://www.boost.org/doc/libs/1_61_0/libs/graph/doc/index.html*

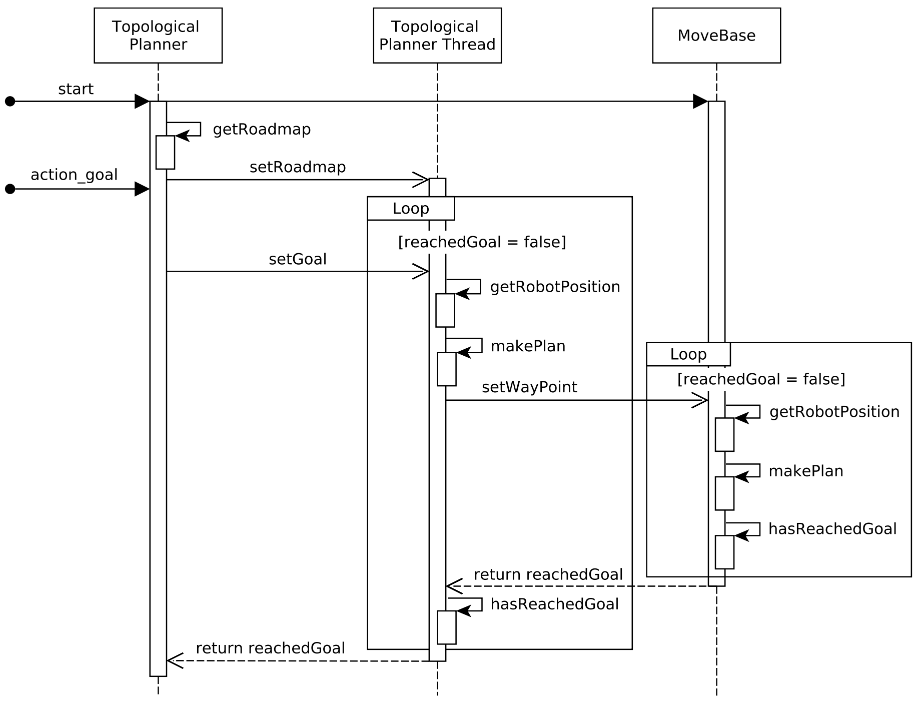


Figure 6.3: Illustration of the topological planning procedure interacting with the `Move-Base`.

6.3 Terrain Analysis

This last section explains implementation decisions for the elevation and vegetation mapping and gives some insight on the structure of the module. Both tasks are combined within one ROS package *elevation_mapping* because they depend on the same data-flow, which is laser scan measurements as input and a costmap layer as output.

The mapping can either work in a rolling window mode or with a complete persistent map. As it is preferable to keep the data transfer as small as possible we picked the rolling window functionality. The related costmap has always the same size as the elevation map. If it exceeds the size of the global costmap only information within these bounds are transferred.

In Figure 6.4 the involved classes for mapping are shown. The module *ElevationMappingNode* is carrying the elevation and intensity map. The other two modules are sharing these data by either modifying or reading it. The measurement module owns an additional variance map, which is in our implementation only used when updating existing measurements and thus only relevant for that module. The module *ElevationMaplayer* employs a costmap layer, which can be integrated like any other costmap layer (e.g. inflation layer) to the `Move-Base`.

The implementation uses the provided `pluginlib`[2] of `Costmap2DROS` to make arbitrary changes to the costmap. This plugin offers a C++ interface and already implements basic methods for accessing the costmap. Therefore the *Costmap 2D Layer* is derived from the *costmap_2d* :: *Costmap2D* class.

[2]*http : //wiki.ros.org/costmap_2d*

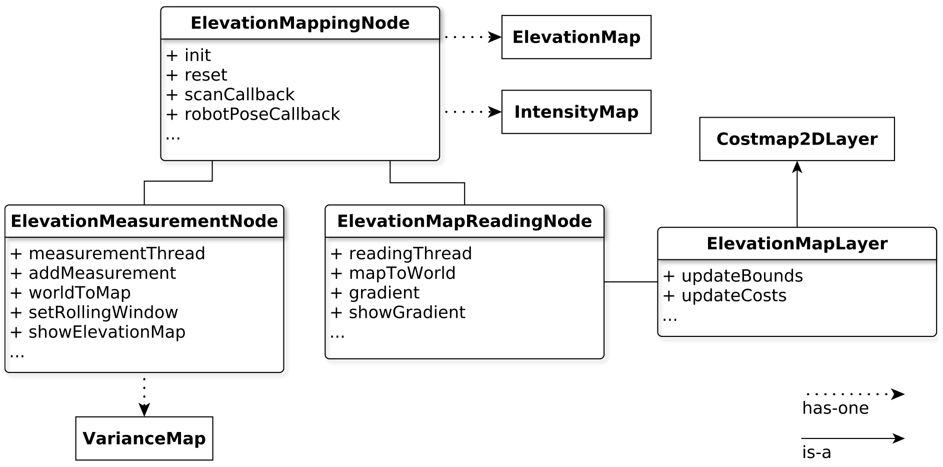


Figure 6.4: Acting classes of the terrain analysis.

6.3.1 Terrain Mapping

The terrain mapping process is primary done in the *ElevationMeasurementNode*. This process is running an external measurement thread for registering new sensor readings. This is especially necessary for depth cameras or laser sensors with a high update rate because in both cases the data transfer is rather high. Unfortunately ROS is known for having delivery troubles when callbacks are blocked for too long.

In our configuration the process receives laser scan and odometry messages. The message types are *sensor_msgs::LaserScan* and *nav_msgs::Odometry* respectively. The odometry information is used to trigger the rolling window and to integrate the readings on the right position in the map. The process has access to the elevation and intensity map, carried by the *ElevationMappingNode* module. Before computing the elevation, the sensor data is filtered with a voxel grid and some statistical filter. Then the measurement is transformed from world to map and finally the height is computed as described in Section 5.7. When performing the vegetation classification we need to look at the incoming range measurements and their corresponding intensity value. For that reason it has to classify each ray right at the start and write the result to the intended intensity map cell.

6.3.2 Terrain Analyzing

The evaluation module is called *ElevationMapReadingNode* and has also access to the elevation map but only for reading purposes. It employs an extra thread for updating the gradient or computing the costs in order to create the costmap layer. This costmap is then handed over to the *ElevationMapLayer*, for making it globally available as a valid costmap layer. In Figure 6.5 the costmap with all individual layer is presented. The implementation features four different costmap layers however the resulting costmap is only maintained in the master. The local occupancy grid map functions as the static layer. Then obstacles visible in the horizontal lasers are inserted and removed in the next layer. Obstacles or terrain relevant information is covered in layer number three followed by an inflation procedure.

The gradient of the elevation map can easily obtained by utilizing OpenCV [3] functions or just by looking at adjacent cells in x and y direction and computing it by hand. Currently the vegetation costs almost comply with a fixed value because any vegetation cell gets the same cost assigned. Only boarder cells have lower costs, due to applied blurring operations. For the final cell cost, the highest value of the elevation or vegetation cost is used.

The generated elevation map is rendered as a color image or in an appropriate ROS point cloud message, this makes evaluation and parameter calibration easier.

[3] *http : //docs.opencv.org/*2.4*/index.html*

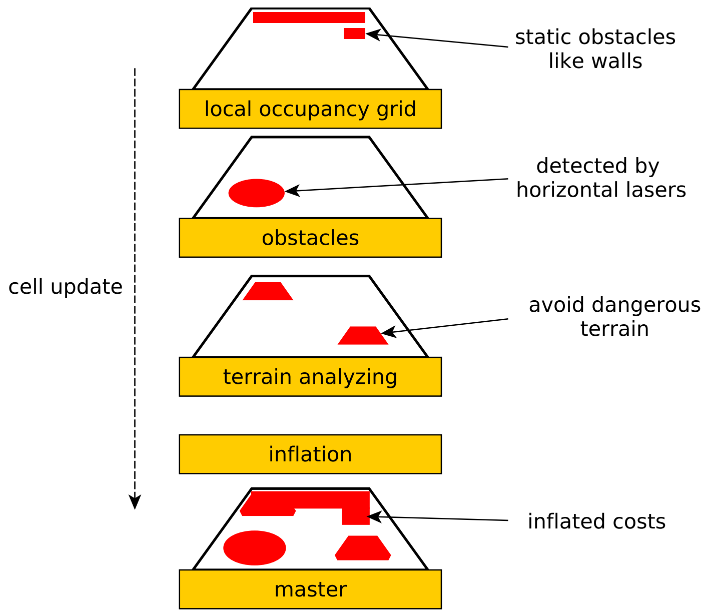


Figure 6.5: Layered costmap illustration with all four individual layers. Yellow rectangles denote a costmap layer and the color red indicates lethal obstacles within the costmap.

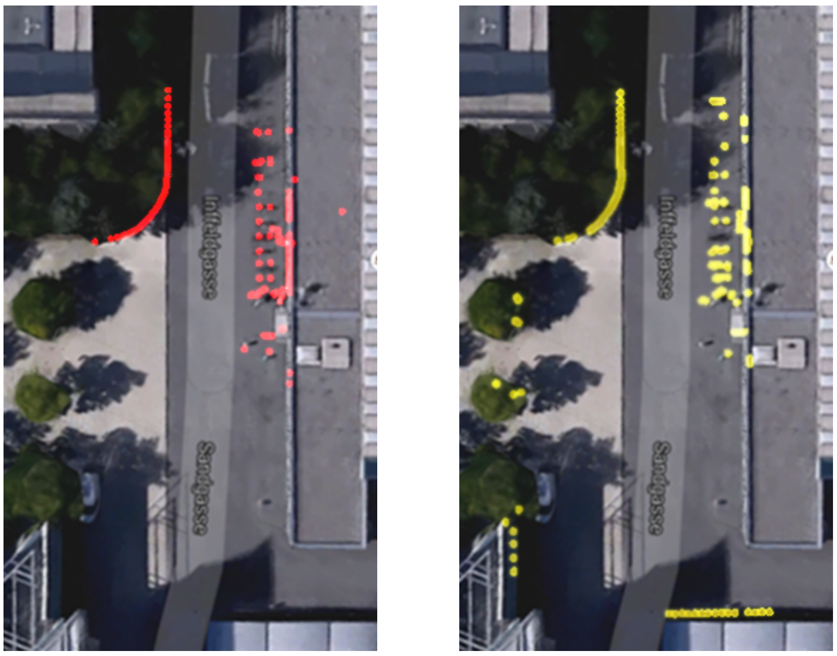
7 Evaluation

The purpose of this chapter is to evaluate the proposed algorithms and concepts. The first part discusses the quality of the used sensors and compares the quality of sensors based on the same technology. In part two the results of graph-based slam and the topological hybrid mapping are presented. Moreover, we investigate the quality of the localization based on these maps. In the next section the planner as well as the roadmap are evaluated. Finally we present some results of the terrain analyzing module.

7.1 Sensors

7.1.1 Laser Range Finders

The SICK LMS 200 laser range finders can be configured for several different operation modes. The measuring distance as well as the frequency depends on the chosen configuration. Due to the outdoor task the longest measuring configuration with a distance up to 80 m was picked. In Figure 7.1 two laser measurements are compared because of varying sunlight impacts. They share an identical location but are measured at different times a day. The measurement at night (Figure 7.1b) has clearly more far reaching scans, especially at the lower part of the image. This is probably due to bright sunlight which is reducing the laser's maximum effective range by several meters. This extra intense light overwhelms the beam originating from the laser and therefore confuses the sensor. Thus scanning at night is the optimal situation.



(a) Laser data captured at 10:00 am.

(b) Laser readings at 11:00 pm.

Figure 7.1: Laser range finder comparison between day and night measurement (underlying map image © Google). Captured data at the same position but at different times a day. A substantially difference in long range measurements is visible. The red and yellow color denotes different measurements.

7.1.2 GPS

At the beginning of the development we quickly realized that the accuracy of GPS in urban environments is very poor, especially in urban canyons. First of all we evaluated the used GPS sensor by picking a fixed track around the campus. The robot was remote controlled and three measurements were recorded at different days and times. The result is presented in Figure 7.2. Due to missing ground truth measuring capabilities we estimated the path on the basis of aerial imagery provided by Google.

Typical reasons for poor measurements are multi path readings, too few satellites, or weather conditions. Tall buildings and urban canyons in general disturb GPS very much. These effects are framed with black rectangles in Figure 7.2. Rectangle **a)** indicates GPS errors due to a block of houses and **b)** show GPS disturbances caused by tall buildings. The measurement accuracy within those places was always relatively worse than the remaining track. We even encountered jumps of up to 50 meters. But the overall poor quality in this sample is caused due to too few satellites.

GPS sensors often require a warm-up time to initiate the GPS-fix. This can take up to several minutes. In our experiments we started with a warm-up time of about 15 minutes. Another improvement brought an adaption, by mounting a metal plate beneath the GPS antenna. The size of that plate is approximately 10 by 10 cm and boosts up the sensor capabilities by preventing multi path effects caused by reflected GPS signals, e.g. from the ground.

Due to very bad GPS readings we decided to compare our IMU-GPS with GPS from different manufacturers. An obvious choice was to test the tracking capability of modern smart phones and additionally we picked the Garmin Edge 705, as it is a typical consumer product for tracking purposes.

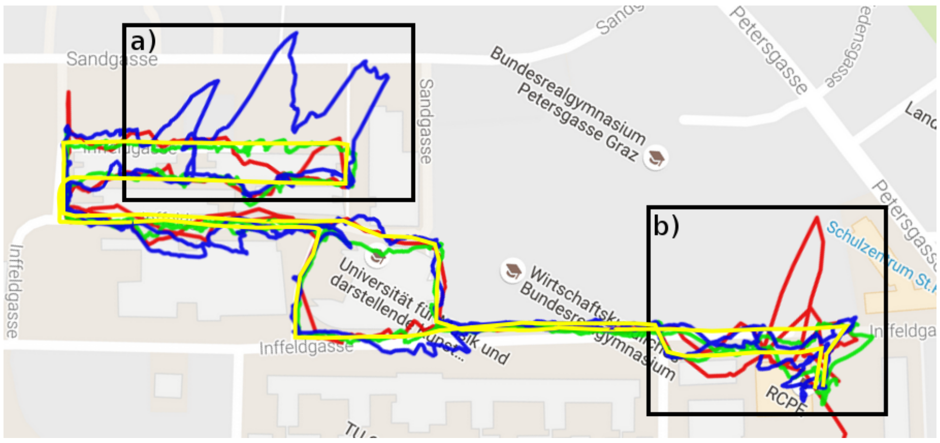

a)
b)
Sandgasse
Sandgasse
Inffeldgasse
Inffeldgasse
Petersgasse
Bundesrealgymnasium Petersgasse Graz
Universität für darstellende Kunst...
Wirtschaftskundliches Bundesrealgymnasium
Schulzentrum St.
RCPE

Figure 7.2: Sensor evaluation of the IMU-GPS measured on three different days and times (map image © Google). The ground truth is emphasized in yellow. The green, red and blue colored path corresponds to an individual measurement. Measurement errors due to a block of houses are labeled with **a)** and errors because of tall buildings are labeled with **b)**.

The output of every sensor is shown in Figure 7.3 and in Figure 7.4 the IMU-GPS is additionally shown together with the ground truth. This sample route was taken on a sunny midday and during a favorable time with many satellites. At the first glance it is noticeable that all sensors have errors of at least five meters in some spots. In this particular run the IMU-GPS has striking accuracy than the competitors. It only remains weak at some spots, which are described in more detail in Figure 7.5b. The worst performance was delivered from the mobile device which is colored in blue.

In Figure 7.5a a close up of the independent measurements is presented. One can see that the mobile device suffers from a sparse measurement frequency, in contrast to the other two devices. This low frequency causes fewer measuring points and thus long straight lines appear inside the interpolated path. Whereas the IMU-GPS achieved outstanding results on that day. Beside some rare spots it only diverges from the ground truth by estimated 3 meters.

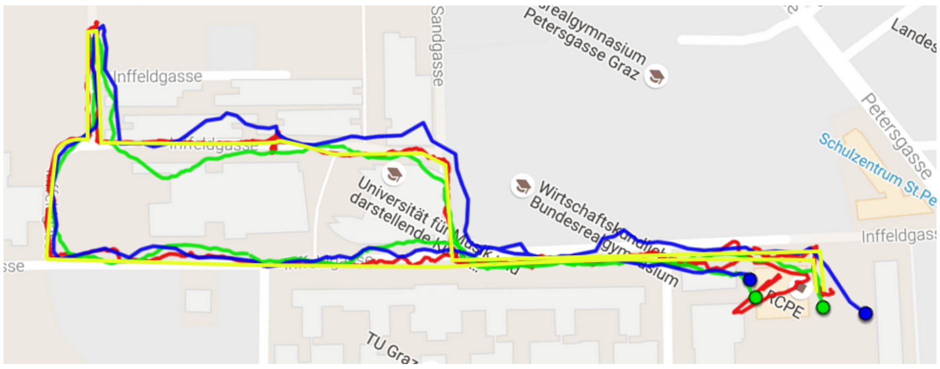

Inffeldgasse
Sandgasse
Realgymnasium Petersgasse Graz
Petersgasse
Schulzentrum St.Pe
Universität für Musik und darstellende Kunst
Wirtschaftskundliches Bundesrealgymnasium
Inffeldgass
RCPE
TU Graz

Figure 7.3: Sample route taken through the campus, to compare GPS from three different manufacturers. The track colored in red is captured from the IMU-GPS. The blue one is measured with the Samsung Galaxy S5 and the green one by the Garmin Edge 705. The ground truth is highlighted in yellow (map image © Google).

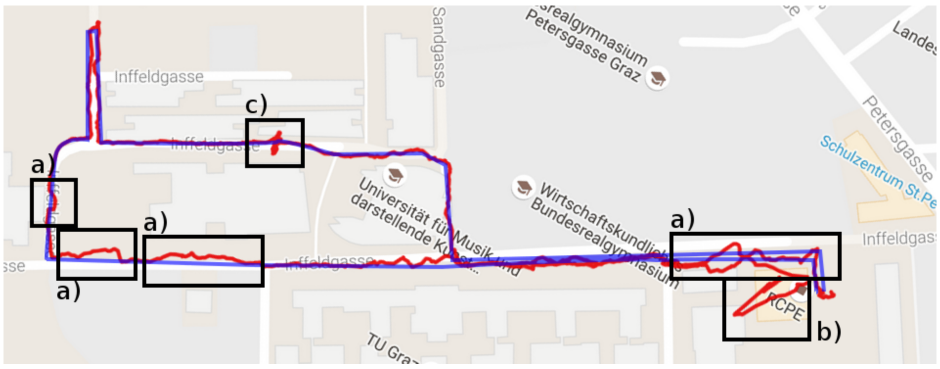
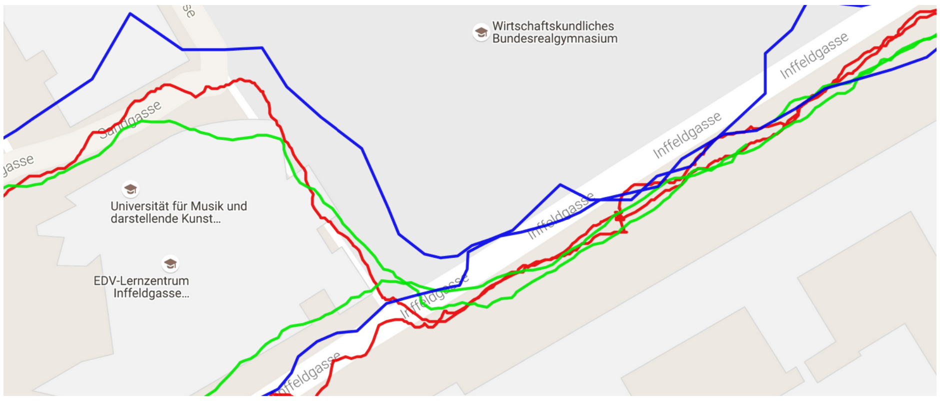
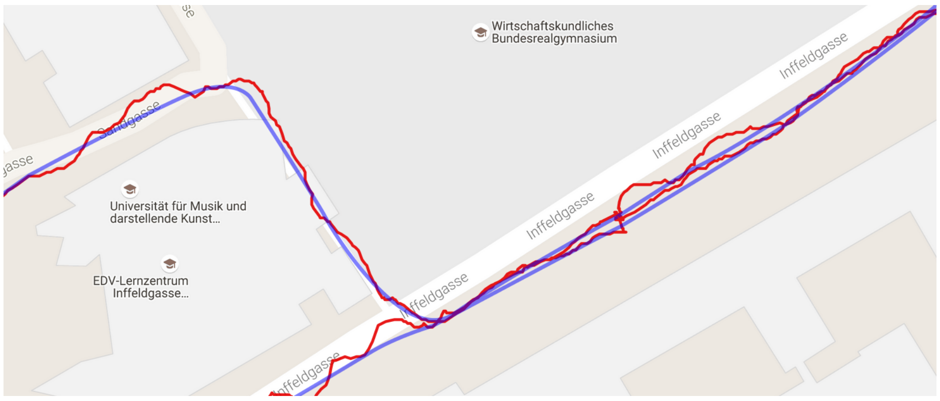

a)
a)
a)
a)
c)
b)
Inffeldgasse
Sandgasse
Realgymnasium Petersgasse Graz
Petersgasse
Schulzentrum St.Pe
Universität für Musik und darstellende Kunst
Wirtschaftskundliches Bundesrealgymnasium
Inffeldgass
RCPE
TU Graz

Figure 7.4: Same sample route as in Figure 7.3 to highlight the IMU-GPS together with the ground truth (colored in blue) separately. Framed places with label **a** demonstrate influences from large buildings. Label **b** shows a failure due to indoor driving and in label **c** the measurement unit stood still for several minutes and this caused scattering (map image © Google).



(a) IMU-GPS is colored in red. The green track is measured by the Garmin Edge 705 and the blue shows the readings from the mobile device.



(b) Close-up view of the IMU-GPS, again in red and the ground truth in blue color.

Figure 7.5: Close-up shot of the GPS comparison in Figure 7.5a as well as the single IMU-GPS track compared to the ground truth (map images © Google).

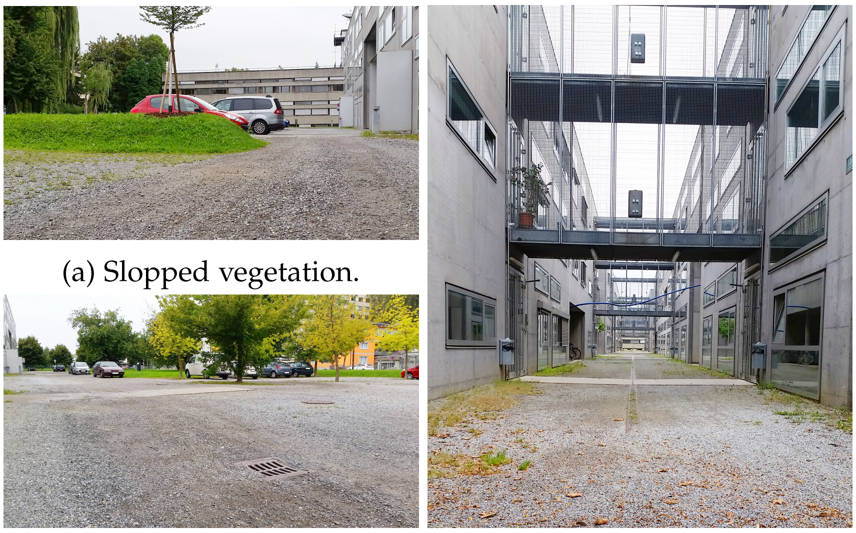
7.2 Mapping

In this section the proposed mapping approach comprising the graph-based and the topological hybrid mapping is evaluated.

7.2.1 Graph-Based Slam

For testing the graph-based slam we captured several individual measurements of the complete campus. Therefore the full sensor suite was required. Then we picked the most promising measurement (depending on the GPS quality, the number of dynamic objects) and finally created the graph for evaluation. In Figure 7.7a the constructed graph is overlapping a provided map of the TU Graz. The created pose-graph is completely consistent because every real place consists only once within the graph. Consistency was mainly possible due to loop closing, which successfully connected recurring places. In our specific case, the most critical spot for loop closing was below the bottom right corner of the building Inffeld 10. This building is framed with a red rectangle and labeled with **c** in Figure 7.7a. The overall map quality fits the underlying map. Some areas look very similar to the map, whereby others show wide disparities and do not capture the reality quite well. This is especially the case if smaller structures are not registered in the official map.

Figure 7.7b shows a close up of the building Inffeld 16 together with the overlaying laser scans. The second close up (Figure 7.7c captures the difference between the building Inffeld 10 and the created pose-graph. The underlying images in gray correspond to the specific sections of the campus building plan offered by the TU Graz. The green colored rectangles within both figures highlight well fitted places. Whereas blue areas depict aberrations. The detail view of Inffeld 16 shows many deviations, mainly because the laser scanners where not able to reach far enough (see Figure 7.6b) or measured the slopped vegetation (see Figure 7.6a). We further had to cope with many GPS failures in that area because of many close


(a) Slopped vegetation.

(b) Wide open area

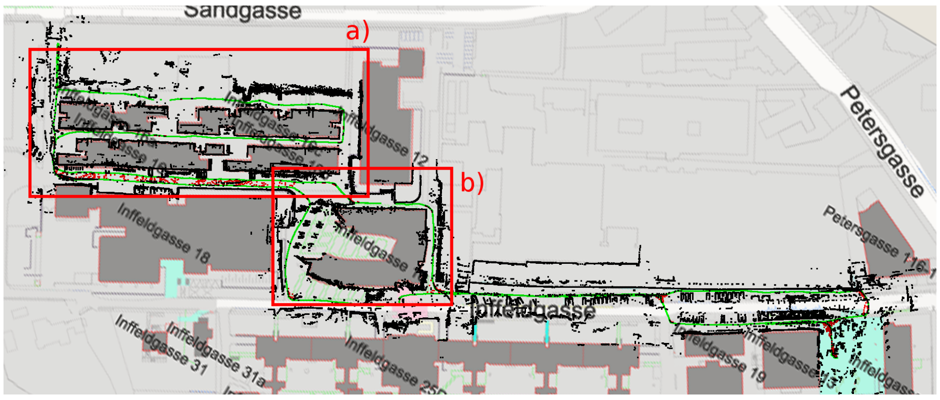
(c) Close nearby buildings.

Figure 7.6: Difficult regions for building maps.

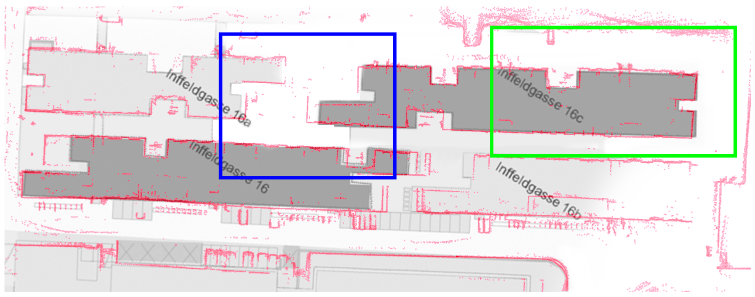
by buildings (see Figure 7.6c), nevertheless the green framed part of the map still fits very well. Some difficult regions next to Inffeld 16 are presented in Figure 7.6. When the robot traverses the inner part of these buildings the translational error increased (area inside the blue) mainly because no error correction with the GPS was possible.

The inner place of the Inffeld 10 was not mapped, due to not far enough reaching laser range measurements. In Figure 7.7c the faulty part is caused by the large slightly curved part of the building. This is because scan matching tries to fit a straight wall which is simply wrong in this case. One might think the wall on the lower left is completely shifted to the original but here the visualized laser scans relate to the bicycle stands behind. This happened because the ground is never perfectly flat and thus the robot stood slightly lopsided and measured over the actual wall.

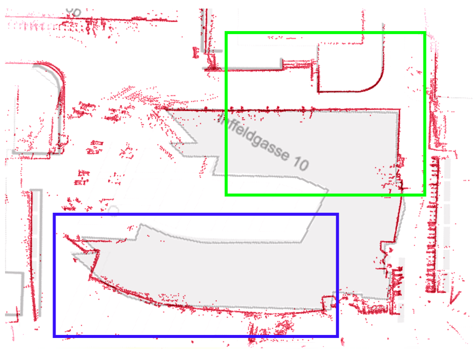
The time for building the most accurate pose-graph took almost twice as long as the capturing time on a standard Lenovo T450s with an i7-5600U CPU and 2.60 GHz. The measurement time took



(a) Overview of the official map with the overlaying pose-graph. The laser scans are colored in black. The red rectangle **a)** shows the extracted close up related to Figure 7.7b and **b)** relates to Figure 7.7c.



(b) Close up of the pose-graph near the Inffeld 16 buildings.



(c) Inffeld 10 building in more detail.

Figure 7.7: Close ups of the constructed pose-graph. Green rectangles denote a good fit with the underlying map and the blue rectangles show faulty parts (map images © TU Graz).

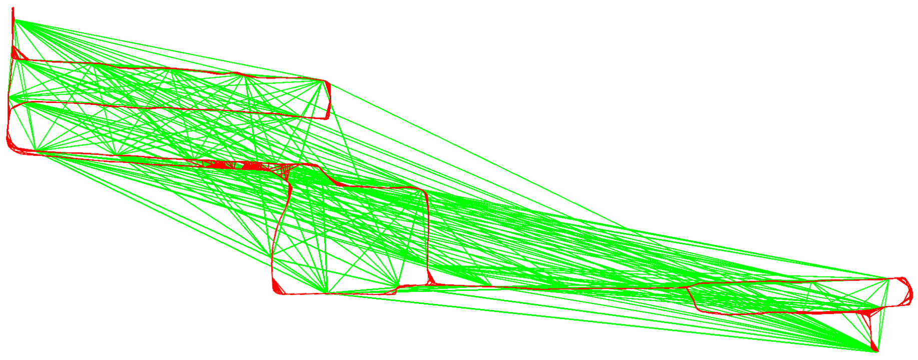


Figure 7.8: Generated pose-graph of the TU Graz campus consisting of all vertices, edges as well as GPS related edges. Red color indicates ordinary edges and green denote GPS edges.

45 minutes and the process required nearly two hours for building the graph. The scan matching, especially the fine grained matching for loop closing was very time consuming. The optimization of the graph never took longer than several microseconds. The visualized graph of the campus in Figure 7.7a has overall 6.371 vertices and 18.382 edges including 385 GPS edges. In Figure 7.8 the complete pose-graph is shown as standalone together with all vertices, edges as well as GPS related edges.

7.2.2 Topological Hybrid Mapping

The overall generated topological hybrid map is visualized in Figure 7.9 with an activated local grid example in the center. Figure 7.10 shows the same scene as Figure 7.9 but zoomed in to the activated map. This graph consists of 52 local maps, 64 edges. It took one and a half minute to generate by using the previous mentioned pose-graph. Therefore, an online approach would be imaginable by changing the implementation in a way to take only the near neighborhood into account. This would break down the computation to few milliseconds.

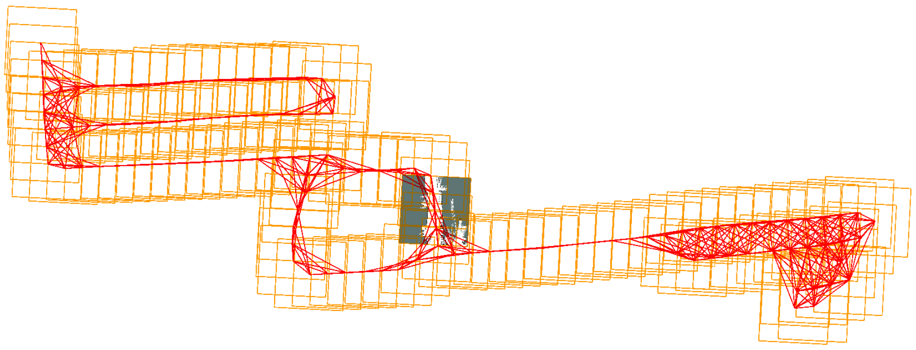


Figure 7.9: Generated topological hybrid graph of the TU Graz campus together with an activated local grid sample in the center.

The local occupancy grid maps have a size of 60 by 60 meters and some selected examples are depicted in Figure 7.11. The overview map in Figure 7.11a highlights the selected local grid map locations. The first three samples present very accurate generated maps with very clean walls, free areas in white, and only few outliers. The lower three figures demonstrate bad results, mainly caused by free open areas where the majority of the laser scans did not reach any obstacle or only uneven terrain.

The mapping has definitely problems in wide open areas. Laser range measurements which do not hit any obstacles in such areas have no influence on the resulting map. Therefore, the mapped area is labeled as unknown and planning algorithms sometimes avoid unknown spaces, depending on the configuration. Nevertheless, these maps can still be used for localization but one can expect state estimation problems here.

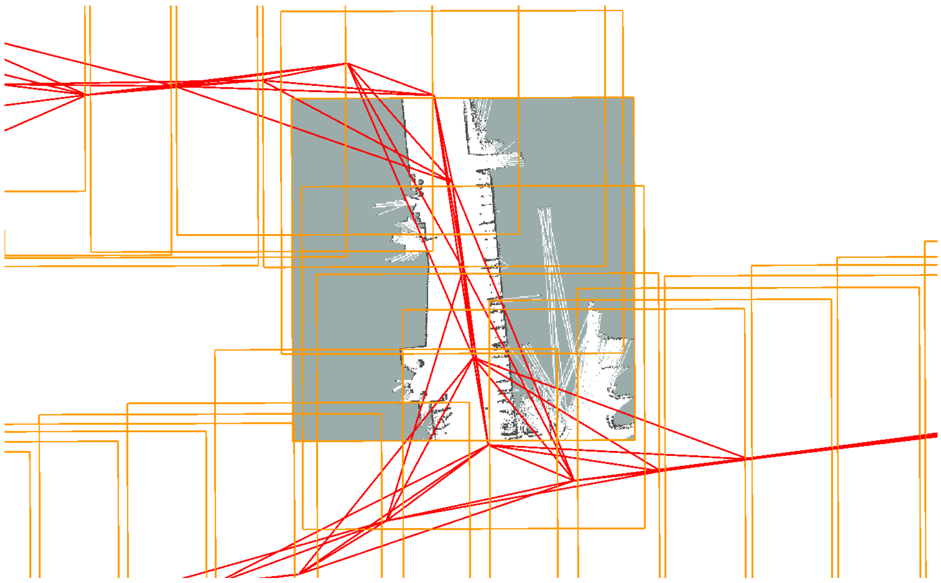
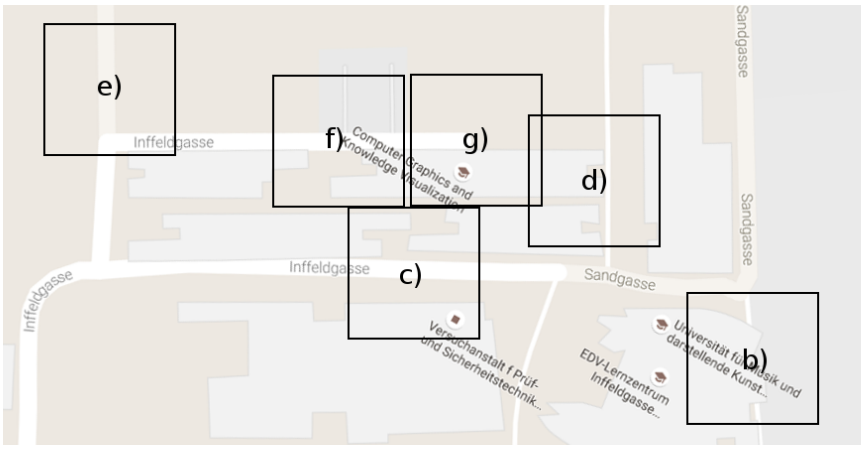


Figure 7.10: Cutout of the overall topological hybrid graph of the TU Graz campus. Edges are colored in red and orange denote local grids.

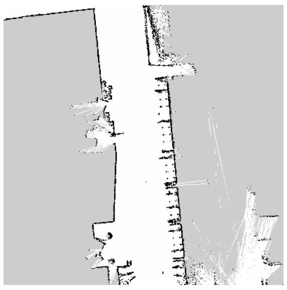
7.3 Localization

We utilize *AMCL* for localization within the local grid maps, as described in Section 4.4. This particle-filter based implementation provides the functionality to switch the map at any time. As we are using different local maps, the particle filter has to deal with a changing map at any time.

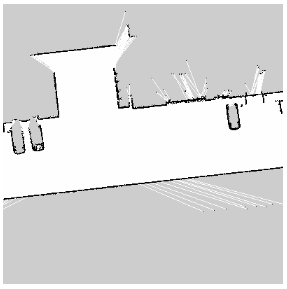
While testing, we recognized that the filter re-distributes (sampling a Gaussian distribution) its particle believe on map change. Figure 7.12 shows the re-distribution effect of the cloud. The first Figure 7.12a presents the localization state shortly before the map change happens. As one can see the first particles cloud features more accurate information about the distribution as the second one. Due to accumulating translational errors caused by long corridors the smaller cluster appeared. This can be explained by this specific implementation, which is only referring to the estimated


e)
f)
g)
d)
c)
b)
Inffeldgasse
Inffeldgasse
Inffeldgasse
Sandgasse
Sandgasse
Sandgasse
Sandgasse
Computer Graphics and Knowledge Visualization
Versuchsanstalt f Prüf- und Sicherheitstechnik...
Universität für Musik und darstellende Kunst...
EDV-Lernzentrum Inffeldgasse...

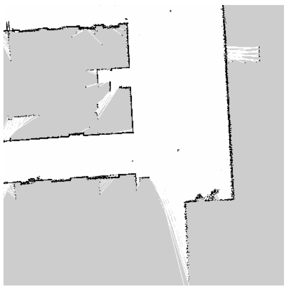
(a) Map showing the regions of each selected local occupancy grid map (map image © Google).



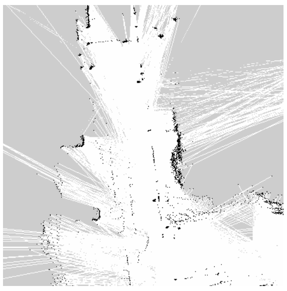
(b) Good result with many details.



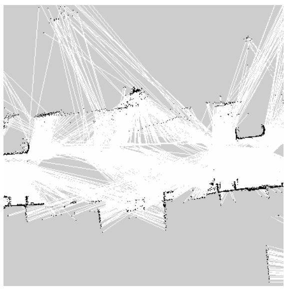
(c) Clean map with straight walls and cars.



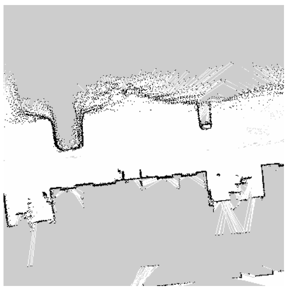
(d) Corners and walls are good captured.



(e) Bad result, due to open area.

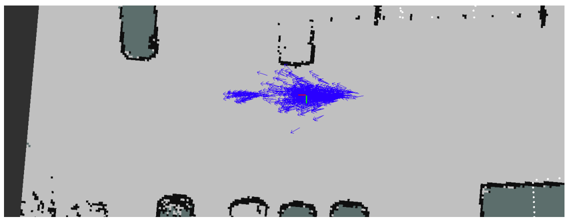


(f) Sparse features and almost no objects.



(g) Noise due to slopped terrain.

Figure 7.11: Selected occupancy grid maps of three good (b to d) and bad (e to g) results. The upper maps show good outputs of the map making process, whereas the lower ones demonstrate critical regions with very few measurements or uneven terrain.


(a) Two particle clusters emerged.

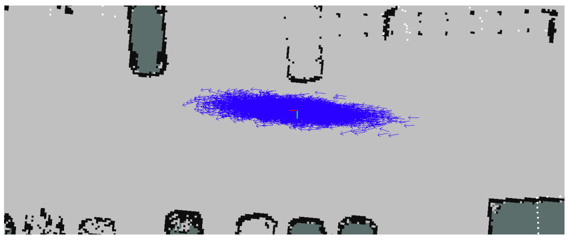
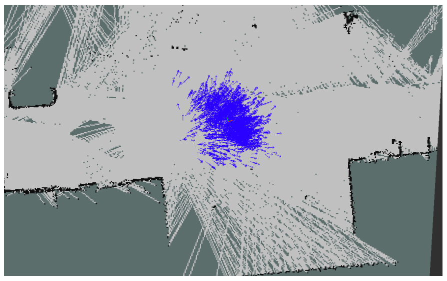

(b) One large cluster after re-sampling.

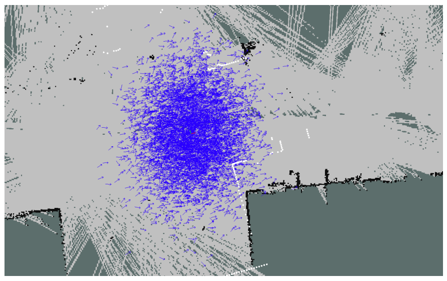
Figure 7.12: Particles distribution changes due to map switching. First two clusters appeared because of translational errors. After switching the map this information was lost.

covariance matrix on map change. The effect has advantages as well as disadvantages. We lose the previous obtained knowledge in some cases. One advantage might be that in other cases this filter refreshing causes better localization results because the internal believe is re-distributed. The particle filter was configured to introduce more translational noise by tuning the motion model accordingly. Two factors strongly influence the localization success. Without any landmarks at the front or back the robot is not able correct the translational error. For that reason we increased the particle spreading in its driving direction.

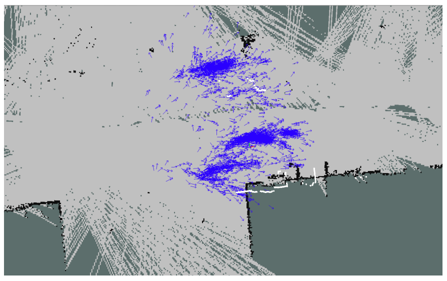
Figure 7.13 shows a very difficult localization scenario with very few landmarks, except slopes on a gravel road. The IMU usually supports the Kalman filter by estimating a better odometry but when driving on gravel the sensor is shaken and the odometry in general is fairly poor. Nevertheless the position estimate recovered after the adaptive particle filter successively increased the particle number and the size of the particle cloud.



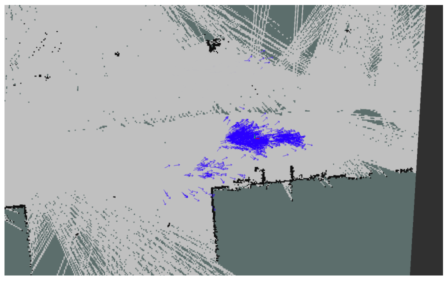
(a) Particle cloud increases.



(b) Unclear about its position



(c) New clusters emerged.



(d) Almost re-localized.

Figure 7.13: The laser range measurements do not capture enough features and therefore causes a spreading of the cloud **a)**. The size of the cloud increases, until new parts of the map appear **b)**. The cloud is then re-sampled due to the earlier mentioned map change effect and is therefore able to find better solutions **c)**. The particle filter then quickly creates new clusters and re-localizes **d)**.

A proper setup for evaluating the localization is rather difficult, especially for outdoor robots because no real ground truth exists. As we had seen before, GPS is far not precise enough to serve as the ground truth and other systems are not feasible for outdoor usage.

We therefore decided to mark various spots at the campus site (see Figure 7.14) and only measure the position deviation on these predefined places. For evaluation the robot was remotely driven through every marked spot in the same order. We repeated this process three times with different level of difficulty. Late at night with favorable empty streets, at midday with some ordinary disturbances, and a relatively crowded and dynamic environment. This methodology enables us to obtain qualitative results and to draw some conclusions about the localization performance.

Figure 7.15 shows the resulting accuracy distribution, measured with increasing difficulty. In the test scenario at night the best localization results are achieved, as expected. Among some few outliers due to translational shifts it accomplished an average accuracy of approximately 0.2 meter. Whereas the crowded scenario has overall more problems but nevertheless the mean deviation is still lower than half a meter, which is good enough for outdoor transportation tasks.

In Figure 7.16 the distribution for every spot is visualized. The plot gives a little insight on how well the robot is localized around that specific area. For example spot 15 (labeled within Figure 7.14) is one of the most difficult places at the TU campus. This area has very few localization features and earlier visited spots (spot 13 and 14) already cause a relatively poor position estimate. Furthermore the robot has to cope with a gravel road and vibrations, which results in very bad odometry and faulty laser scan measurements. The overall localization accuracy is surprisingly good. Further developments by for example introducing GPS measurements into the particle-filter could still improve the accuracy, especially in free open areas.

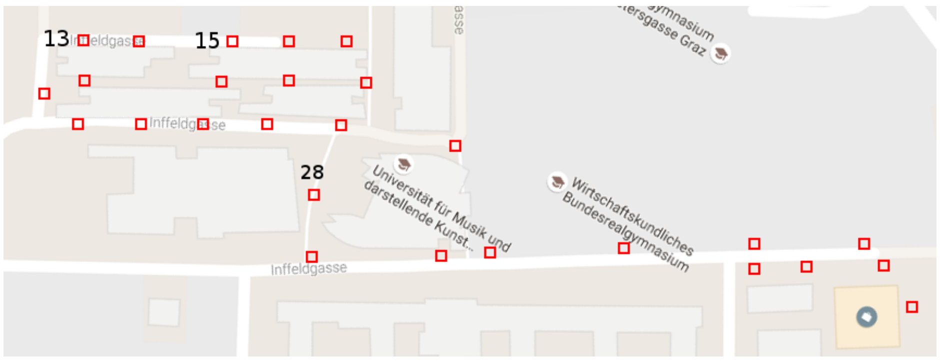

13
15
28
Inffeldgasse
Inffeldgasse
Universität für Musik und darstellende Kunst...
Wirtschaftskundliches Bundesrealgymnasium
...ersgasse Graz
Gymnasium

Figure 7.14: Marked placed on the TU Graz campus to evaluate localization accuracy and performance. Spot number 13, 15 and 28 probably cause the lowest localization accuracy.

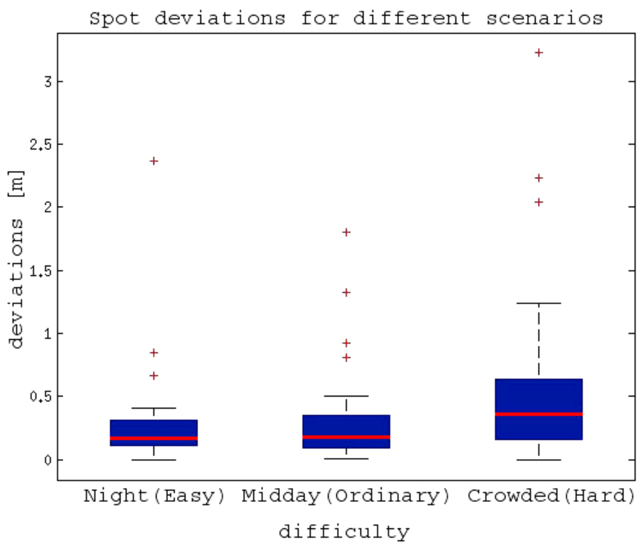


Figure 7.15: Resulting spot deviations for three different scenarios with varying difficulties. Measured at the earlier defined places.

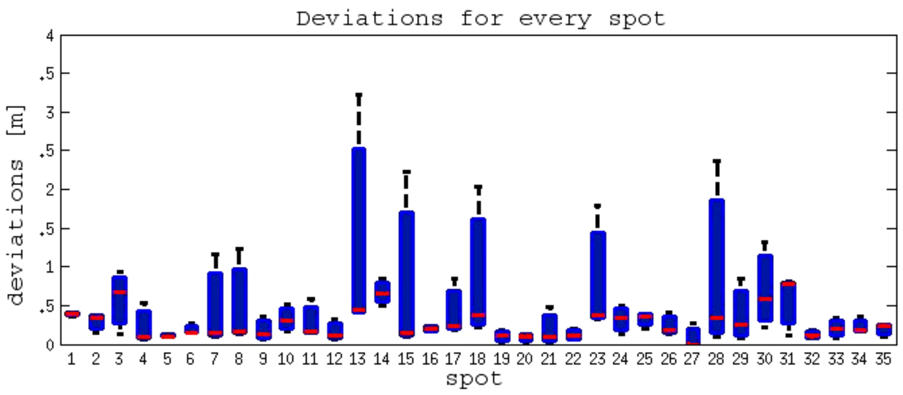


Figure 7.16: Localization deviations for every spot. Individual spots show very low localization accuracy (e.g. 13 and 15).

7.4 Planning

An extensive comparison of the runtime and the path optimality between graph and grid-based search (or metrical planning) has been done in former literature by Konolige et al. [30] as well as in newer work by Imlauer et al. [33]. As a conclusion Konolige pointed out that the graph planner is significantly faster although the traveling distance is negligible longer. To validate their findings for larger environments we directly compared the metric planner (`GlobalPlanner`) with our topological planner. We rendered one large occupancy grid map because the metric planner requires it for planning. Then we performed 50 test by randomly chose the start and end point within the map. Thus the resulting path distances varied from 10 to 700 meter. In Figure 7.17 the averages of the resulting distances are presented. It can be seen that the metric planner created overall shorter paths with an average length of 182 m, than the graph-based method with an average of 273 m. Whereas the expected planning duration of the graph planner (presented in Figure 7.18) is several orders of magnitude faster than the metrical one. The graph planner outperforms the metrical planner with an average of 0.006 seconds compared to 0.79 seconds.

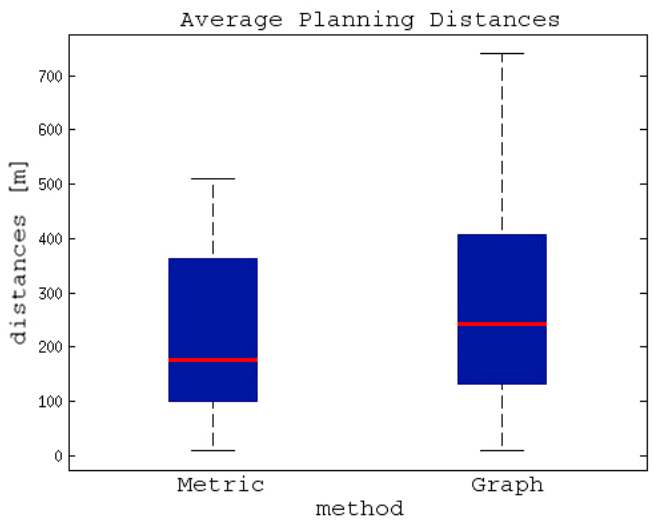


Figure 7.17: Evaluation of the resulting path distances computed by metrical and graph planner. The metrical planner created shorter paths on average to the goal.

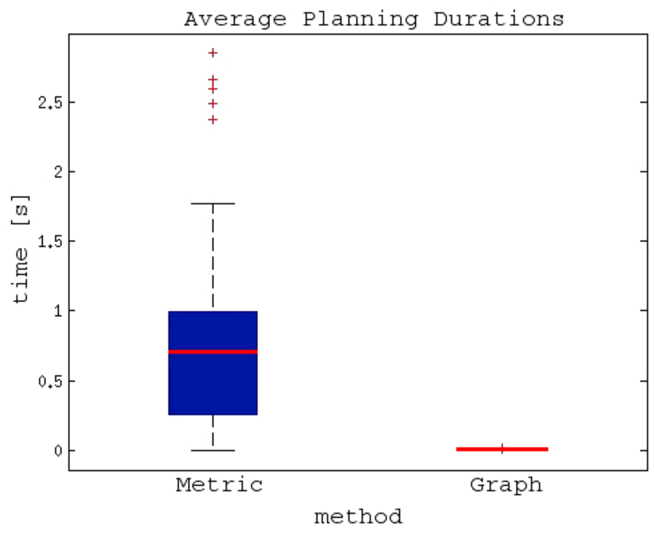


Figure 7.18: The resulting planning durations of both methods. Graph-based planning outperforms the metrical planner by several orders of magnitude.

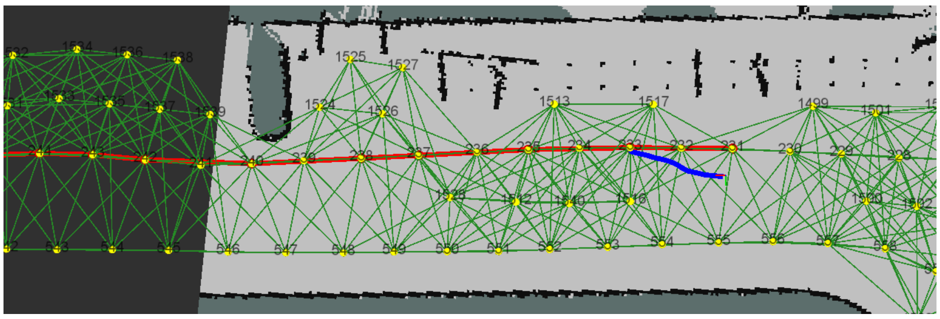


Figure 7.19: Topological planner provides a way-point sequence for the low-level planner (path is colored in red). The metrical planner's path is colored in blue. The first two way-points have already been visited and the next valid goal is fed to the metrical planner.

The implemented planner outputs a sequence of way-points as described in Section 5.6.2. A planned path, starting next to the building Inffeld 13 and finishes at the upper left entrance for example, consists of 157 way-points.

Figure 7.19 shows the individual planning hierarchies. The way-point sequence to the goal is provided by the topological planner and colored in red color. This path is rather straight forward. The low-level planner is receiving the way-points one by one. The high-level executor frequently updates these goals in a way, so that the low-level planner never really reaches a goal. This is the reason why the metrical planner is already planning to the next way-point in the image. As shown in the picture the occupancy grid is limited and the low-level planner cannot reach beyond the map, whereas the high-level planner is able to, by utilizing the roadmap.

The other Figure 7.20 shows a corner of a building, where the planner created a smooth path because of the roadmap graph topology. The graph not only connects close nearby points but points which are slightly further away, to allow shortcuts. This behavior is especially essential for slow robots and prevents abrupt motions.

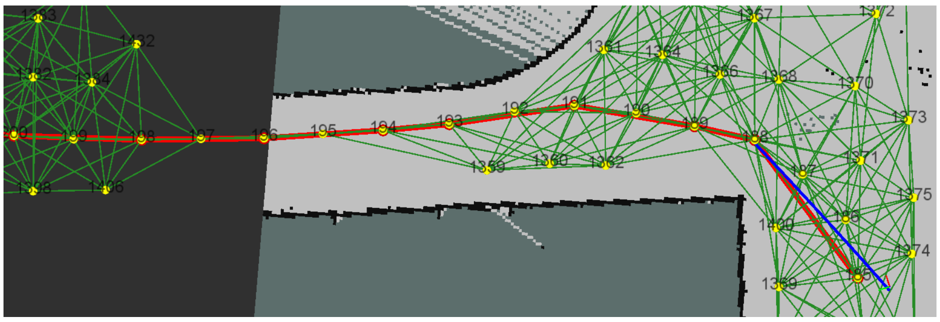


Figure 7.20: The roadmap topology allows the planner to select points, so that corners are translated to curves, which is useful for easier driving.

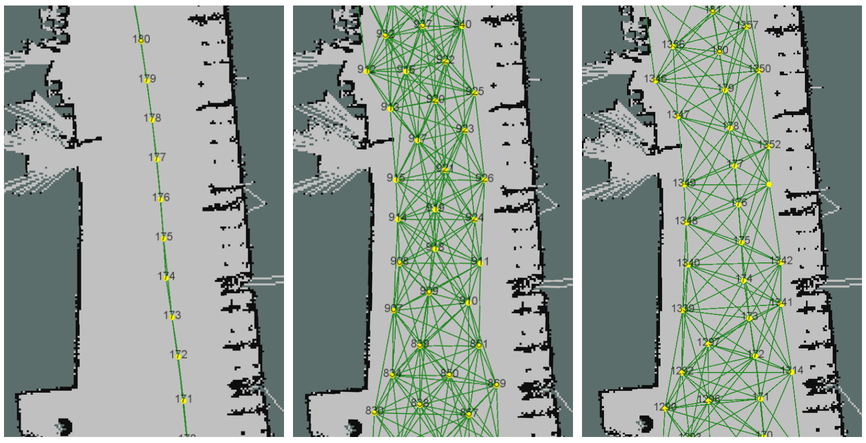
7.4.1 Roadmap

As we pointed out in the previous chapter the roadmap generation can be made from various inputs. In Table 7.1 the different output sizes of each generation method can be compared. The smallest solution is with no doubt the pose-graph only variant, as it consists of three times fewer way-points compared to the other roadmaps. The coupled solution features approximately one percent fewer way-points as well as edges than the building method from the maps only.

	Pose-graph	Maps	Both
Vertices	711	2.598	2.505
Edges	1.836	16.816	15.346

Table 7.1: Size comparison of roadmaps generated with different methods.

The difference in graph topology is compared in Figure 7.21. It can be seen that the pose-graph created roadmap consists of a very simple topology (Figure 7.21a). The combined method constructs a similar complex topology with even fewer way-points and edges compared to the map layout approach. The main advantage is indeed the controlled influence on the graph structure by utilizing the pose-graph as additional input.



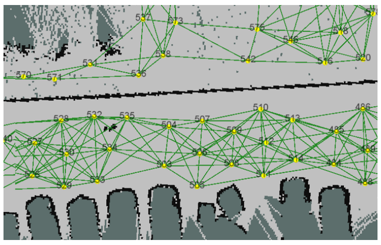
(a) Pose graph only. (b) Evenly distributed points. (c) Both informations combined.

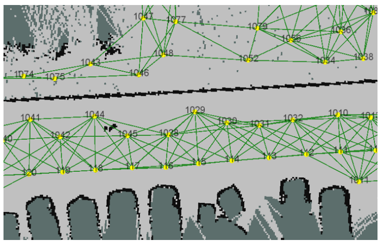
Figure 7.21: Comparison of different roadmap generation methods. The figure on the far left utilizes only the pose-graph for creating the roadmap. Whereas the center figure presents a result after spreading out points evenly on the map. The combined version can be seen on the right.

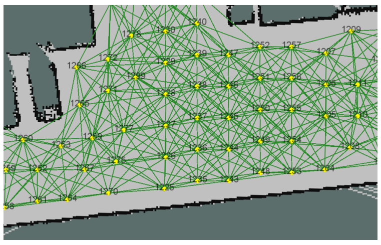
To underpin this advantage we picked several example sections, which are collected in Figure 7.22. The first two figures present a narrow walkway near to parking cars. Due to the pose-graph, more edges parallel to the walkway exist. The robot can easier follow this simpler and straighter way-point topology.

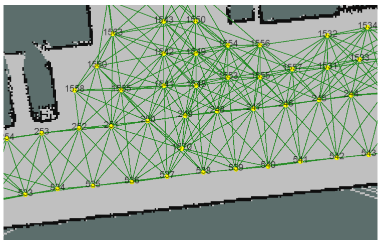
In Figure 7.22d the roadmap is directly located on a street without any walkways. One of our main goals was to incorporate reasonable routes into planning and in this particular case the robot should avoid the center of the street. The normal way-point spreading technique would evenly spread out points including the street center, which causes routes along the center. With the combined version it is more likely that the robot will stay at the street boarders, except it is necessary to traverse the street.

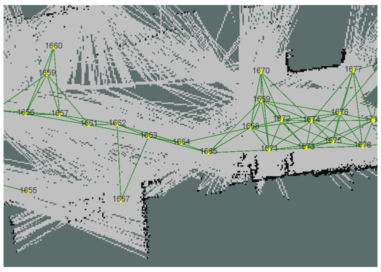
The last sample section shows a barely useful map (Figure 7.22f) but together with the information of the pose-graph even an additional possible path emerged.


(a) Irregular path structures.


(b) Edges parallel to the path.


(c) Everywhere evenly distributed.


(d) Preferable for avoiding streets.


(e) Only one possible path.

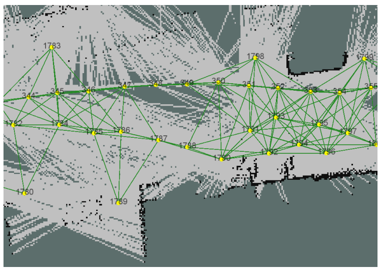
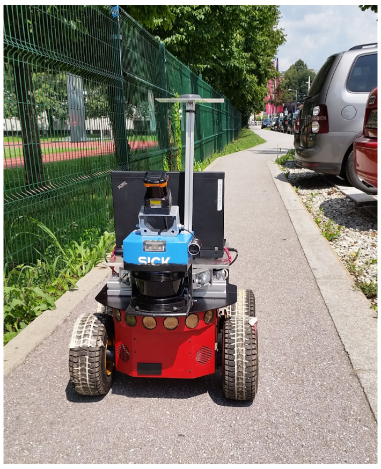

(f) More paths available.

Figure 7.22: Example sections of the generated roadmap. Figures on the left (Figures 7.22a, 7.22c and 7.22e) show the roadmap topology without using any pose-graph information. The effect of building the roadmap together with the pose-graph is presented on the right-hand side.

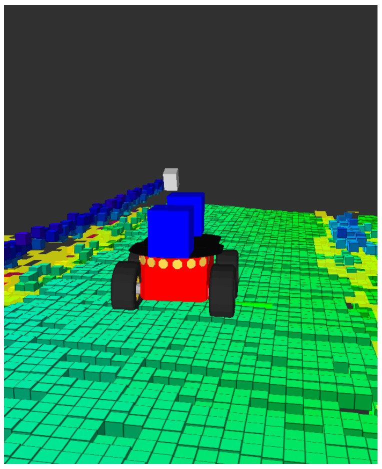
7.5 Terrain Analysis

The ground obstacle elevation is visualized in Figure 7.23. The robot is traveling along a narrow passage between cars, gravel and a slope with a fence. This scenario has a great demand for environmental assessment. The elevation map is fed by the down facing laser and therefore only obstacles nearby the robot are visible in the representation. The model includes still some unknown cells although the laser's field of view should have covered them. One reason might be the impact angle and another cause is shadowing effects (terrain behind objects cannot be measured). Too steep beam angles cannot always be recognized correctly by the sensor. In this figure the elevation map visualizes every grid cell with boxes. The color as well as the vertical position represent the measured height at that position. For example blue boxes are higher located than green ones. In addition the costmap layer is rendered simultaneously, to highlight non-drivable cells (colored in yellow). To provide a better glimpse of the costmap accuracy we added Figure 7.24 of the same location with a top down viewpoint.

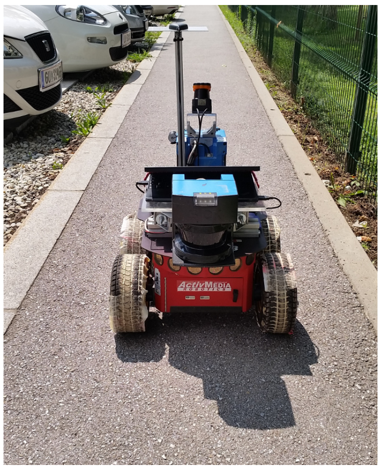
The terrain classification is evaluated in Figure 7.25. Two positions next to vegetation were picked and the corresponding raw classification outputted. As mentioned in Section 5.7 the classification quality strongly depends on the measured range, especially for the Hokuyo laser (see Wurm et al. [31] for more). Thus it was calibrated to classify in a conservatively fashion by means of preferring the road more often. This behavior can be identified in Figure 7.25d because many cells are still classified as road. From this point forward, some post-processing steps like smoothing and applying morphological operators can improve the result drastically. Nevertheless this classification already suffices after inflation and feeding this information to the costmap layer.



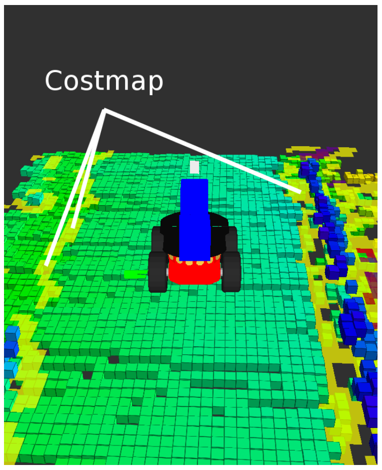
(a) Robot traveling through a narrow passage.



(b) Related elevation map from the same viewing angle.



(c) Same passage taken from the back view.



(d) Corresponding elevation map. Costmap is shown in yellow.

Figure 7.23: Elevation map taken from various angles, while traveling on the campus. Visualization shows the point cloud as boxes and different colors denote different vertical values (z-axis). The resulting non-inflated costs are colored in yellow.

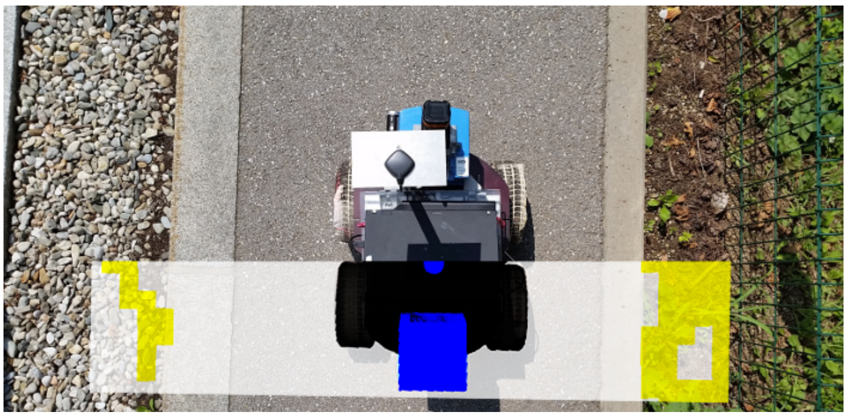
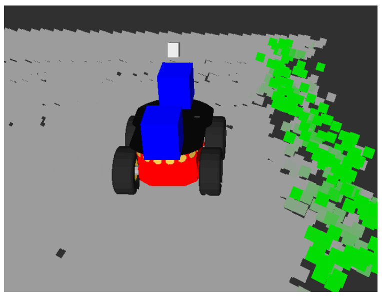
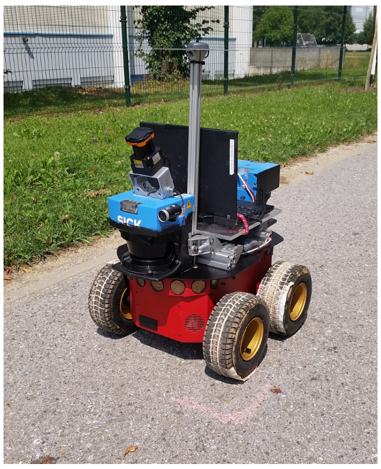


Figure 7.24: Costmap comparison with underlying ground truth image. Yellow color denotes non-drivable terrain and will be inflated later in the process.

(a) Robot standing next to a plant.


(b) Elevation map and color is consistent with classification.



(c) Driving along a walkway with grass nearby.

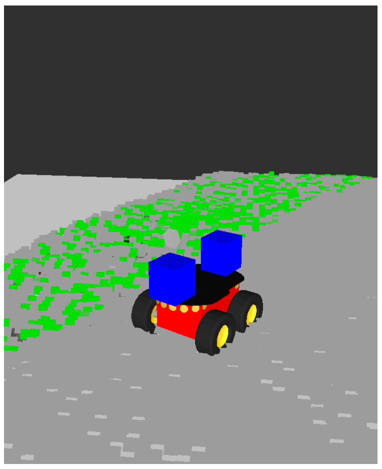

(d) Classified grass, without any post-processing.

Figure 7.25: Sample vegetation classifications at to different spots. Green color indicates classified raw vegetation, without any post-processing.

8 Conclusion

This chapter summarizes this thesis and provides additional guidance for improvements and lessons learned. Furthermore some future developments related to this work are suggested.

8.1 Discussion

Within this thesis an autonomous transport system for indoor and outdoor use was presented. The robot is able to travel dynamically within a campus like environment, by avoiding obstacles and choosing adequate routes to the goal.

In order to navigate on the campus a global map was created by utilizing and further developing former graph-based techniques. This environment representation was built by fusing several sensor sources and made a global consistent map possible. Furthermore, an extended topological hybrid graph was proposed which is able to deal with large-scale maps. This allows the usage of existing and well known technologies, such as A* search for the metrical planning or the adaptive particle filter for estimating the localization. It has computational benefits which is particular important when coping with insufficient processing power, a common problem for mobile applications.

The planning architecture is split up in three abstraction level, so that planning on large-scale outdoor environments remain feasible. Even though the planner still comprises reasonable routing. The proposed high-level planner uses an overlaying roadmap topology which can be modeled to fulfill several needs. We presented different roadmap structures, necessary to meet higher design decisions like avoiding street centers. Moreover we showed how analyzing the surrounding terrain to avoid non-traversable, rough, or just unwanted terrain like vegetation. Hence the planner incorporates the traversability and type of terrain.

In addition the evaluation at the end, presented individual performance results of the map creation, planning or localization module. The overall evaluation pointed out the necessary of every single module and how they collaborate. Finally a fully functional prototype verified the implementation and the proposed tool set. We showed the technical feasibility of developing an autonomous navigation system for campus wide transport tasks.

8.2 Lessons Learned

The map generation of large territories is rather difficult and time consuming. The constantly changing environment, due to construction sites or parking cars is a challenging problem for mapping as well as localization. Furthermore, all kind of moving objects disturb the mapping process and create artifacts within the rendered maps. Moving objects should be treated in a more specific way because not only the mapping result would improve drastically but robots operating in urban environments could then react in a more appropriate way.

The right amount of filtering is one key aspect of such a system. For example the ground filter worked quite well for the mapping but has an adverse effect on localization. For the graph-based mapping, filtering of outliers was the most important part.

Never trust a single sensor. No matter how reliable it appears to be, problems will occur and only multiple hypothesis resolve unexpected behaviors.

The hardest part of such a broad project seemed the distinction between what is important from what is unimportant. Always worried about failing at the key aspects of this thesis because of minor details at the start. Even at the end we cannot rule out the possibility, that small changes at the setup would have made large impacts on the overall system.

8.3 Future Work

The presented system only processes the environment in two dimensions, except for the elevation map. Due to the small height of the robot several problems occurred in conjunction with laser measurements hitting the ground (see Section 5.2 for more on this topic). For systems which work only in two dimensions these are relatively hard problems to solve, whereas in 3D they are considerable easy. Furthermore, every local map, the topological hybrid graph as well as the roadmap could be translated into 3D.

As already mention in Section 8.2 moving objects and constantly changing environments are difficult. Detecting objects, like for example cars, definitely would help the map making process and further the localization. We had some localization problems because nearby cars seem to have very popular scan matching features.

Bibliography

[1] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[2] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. Intelligent robotics and autonomous agents. the MIT Press, Cambridge (Mass.) (London), 2005.

[3] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007.

[4] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 273–278, May 2010.

[5] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613, May 2011.

[6] Henry Carrillo, Philip Dames, Vijay Kumar, and José Castellanos. Autonomous robotic exploration using occupancy grid maps and graph slam based on shannon and rényi entropy. In *2015 IEEE International Conference on Robotics and Automation*. Seattle, Washington, May 2015.

[7] Paul Timothy Furgale and Timothy D. Barfoot. Visual teach and repeat for long-range rover autonomy. *J. Field Robotics*, 27(5):534–560, 2010.

[8] S. Thrun, M. Beetz, W. Bennewitz, M. andBurgard, A.B. Cremers, F. Dellaert, D. Fox, C. Haehnel, D. andRosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot Minerva. volume 19, 2000.

[9] Lars B. Cremean, Tully B. Foote, Jeremy H. Gillula, George H. Hines, Dmitriy Kogan, Kristopher L. Kriechbaum, Jeffrey C. Lamb, Jeremy Leibs, Laura Lindzey, Christopher E. Rasmussen, Alexander D. Stewart, Joel W. Burdick, and Richard M. Murray. *Alice: An Information-Rich Autonomous Vehicle for High-Speed Desert Navigation*, pages 437–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[10] Andrea Bauer, Klaas Klasing, Georgios Lidoris, Quirin Mühlbauer, Florian Rohrmüller, Stefan Sosnowski, Tingting Xu, Kolja Kühnlenz, Dirk Wollherr, and Martin Buss. The Autonomous City Explorer: Towards natural human-robot interaction in urban environments. *Social Robotics*, 1(2):127–140, 2009.

[11] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. Autonomous robot navigation in highly populated pedestrian zones. *J. Field Robotics*, 32(4):565–589, 2015.

[12] Karel Košnar, Tomáš Krajnık, Vojtech Vonásek, and Libor Preucil. Lama-large maps framework. In *Proceedings of Workshop on Field Robotics, Civilian-European Robot Trial*, pages 9–16, 2009.

[13] Pablo De Cristóforis, Matias Nitsche, Tomáš Krajník, Taihú Pire, Marta Mejail, P. De Cristoforis, M. Nitsche, T. Krajnik, T. Pire, and M. Mejail. Hybrid vision-based navigation for mobile robots in mixed indoor/outdoor environments. *Pattern Recognition Letters*, 53(Complete):118–128, 2015.

[14] M. Rufus Blas, M. Agrawal, A. Sundaresan, and K. Konolige. Fast color/texture segmentation for outdoor robots. In

2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4078–4085, Sept 2008.

[15] Triebel R. Pfaff, P. and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing.

[16] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R.Y. Siegwart. *Robot-centric Elevation Mapping with Uncertainty Estimates*. ETH-Zürich, 2014.

[17] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2276–2282, Oct 2006.

[18] Sebastian Thrun, Michael Montemerlo, and Andrei Aron. Probabilistic terrain analysis for high-speed desert driving. In *Probabilistic Terrain Analysis For High-Speed Desert Driving.*, 2006.

[19] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software in Robotics*, 2009.

[20] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 300–307, May 2010.

[21] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *National Conference on Artificial Intelligence (AAAI)*, pages 343–349, 1999.

[22] N. J. Nilsson P. E. Hart and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[23] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[24] Y. Borenstein, H. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, 1996.

[25] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. pages 182–193, 1997.

[26] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[27] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, June 1989.

[28] Michael de Villiers. Heavenly mathematics: The forgotten art of spherical trigonometry. *The European Legacy*, 20(5):560–561, 2015.

[29] G. Grisetti, R. Kuemmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based SLAM. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43, 2010.

[30] K. Konolige, E. Marder-Eppstein, and B. Marthi. Navigation in hybrid metric-topological maps. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3041–3047, May 2011.

[31] Kai M. Wurm, Henrik Kretzschmar, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. Identifying vegetation from laser data in structured outdoor environments. *Robotics and Autonomous Systems*, 62(5):675–684, 2014.

[32] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[33] Stefan Imlauer. A Hierarchical Navigation System for Groups of Autonomous Logistics Robots in Industrial Environments. Master's thesis, TU Graz, Austria, 2016.