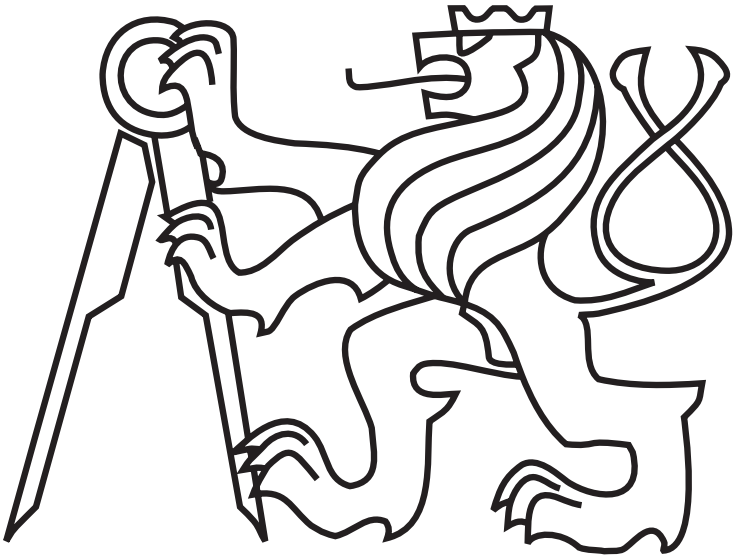
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

DIPLOMA THESIS



Viktor Kozák

Exploration of an Unknown Space with a Mobile Robot

Department of Cybernetics

Thesis supervisor: **RNDr. Miroslav Kulich, Ph.D.**

May, 2017

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Viktor K o z á k

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Exploration of an Unknown Space with a Mobile Robot

Guidelines:

1. Get acquainted with current state of the framework EAPD (Exploration Algorithm in a Polygonal Domain) for mobile robot exploration[1] and with ROS (Robot Operating System) [2].
2. Get acquainted with methods of simultaneous localization and mapping.
3. Choose an appropriate open source implementation of SLAM for laser rangefinder and experimentally verify its behavior with real data. Focus on computational complexity, precision of computed position in various environments.
4. Integrate the chosen SLAM library into EAPD.
5. Design and implement refinement of a polygonal map generated by EAPD in case loop closing is detected by the SLAM algorithm.
6. Verify experimentally the proposed solution with a real robot and describe and discuss obtained results.

Bibliography/Sources:

[1] http://imr.ciirc.cvut.cz/Research/EAPD
[2] http://http://www.ros.org
[3] Tomáš Juchelka: Exploration algorithms in a polygonal domain, Diploma Thesis, CTU in Prague, FEE, Dept. of Cybernetics, 2013
[4] Vojtěch Lhotský: An integrated approach to multi-robot exploration of an unknown space, Diploma Thesis, CTU in Prague, FEE, Dept. of Cybernetics, 2016
[5] Miroslav Kulich, Tomáš Juchelka, Libor Přeučil: Comparison of exploration strategies for multi-robot search, Acta Polytechnica , Vol. 55, No. 3 (2015), pp. 162-168

Diploma Thesis Supervisor: RNDr. Miroslav Kulich, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 17, 2017

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague on..... .....

Acknowledgements

I would like to thank my thesis supervisor RNDr. Miroslav Kulich, Ph.D for his professional leadership, advice on my work and for his patience with me. I would also like to thank all the people who supported me during my years at the university.

Abstrakt

Tato diplomová práce se zabývá prohledáváním neznámého prostředí mobilním robotem. Robot je vybaven laserovým snímačem a během explorace je tvořena 2D polygonální mapa, tato část práce navazuje na Explorační algoritmy v polygonální doméně od T. Juchelky[1]. V rámci práce byla do exploračního algoritmu implementována metoda pro uzavírání smyček v mapě a metoda pro zpětnou úpravu a rekonstrukci mapy. Implementované metody byly otestovány s reálným robotem Turtle Bot v různých prostředích.

Abstract

The subject of this diploma thesis is the exploration of an unknown space with a mobile robot. The robot is equipped with a laser rangefinder and creates a 2D polygonal map, this part of the work follows and extends the Exploration Algorithms in a Polygonal Domain thesis by T. Juchelka[1]. New functionalities were implemented and added to the exploration algorithm, a method for loop closing in the polygonal map and a method for the refinement of the polygonal map. The implemented methods were tested with a real TurtleBot robot in various environments.

Contents

Chapter 1

Introduction

Autonomous robot exploration is a widely used approach in mapping, search and rescue operations. It is utilized to survey areas which might be dangerous or inaccessible by humans. Robots equipped with a wide range of sensors are used to build maps of the environment or to search for specific objects.

In this work an autonomous robot equipped with a laser rangefinder is used for exploration and map building. The focus of this work is on exploration in 2D polygonal domain, which offers certain advantages in contrast to the more popular 2D occupancy grids. Maps in polygonal domain are much more memory efficient, offering a big advantage in larger environments. On the other hand, algorithms designed for occupancy grids may be useless when handling a polygonal representation, thus arises the need to adjust these algorithms or implement more suitable approaches.

The work follows and extends a framework implemented by T. Juchelka in his thesis *Exploration algorithms in a polygonal domain* [1]. T. Juchelka implemented a general framework based on polygonal representation of an environment in Robot Operating System (ROS)[2]. It was later extended and tested on real robots by V. Lhotský in [3]. In Lhotsky's work, robots are equipped with an RGBD camera sensors allowing them to operate using the RTAB-Map package from ROS as its main SLAM algorithm.

In the first part of this work the EAPD algorithm is adjusted for the use with a real robot equipped with a laser rangefinder instead of the RGBD camera. Gmapping package from ROS was selected as a suitable implementation of SLAM for a laser rangefinder, integrated into the EAPD package and tested on a real robot.

The second part of the work is focused on a refinement of the polygonal map, generated by EAPD in case a loop closure is detected by the SLAM algorithm. Since there are more possible methods for the detection of loop closures, these methods are discussed and the more appropriate ones were selected for this work. Gmapping's approach to the loop closure problem is closely connected with it's internal structure, therefore EAPD and Gmapping packages were adjusted to work together and appropriately react to loop closing related changes. Other possible

method more suitable for other SLAM packages is proposed and implemented separately, serving as a foundation for possible future uses with other approaches.

Chapter 2 gives the reader a short introduction to the theory of robotic exploration and covers basic description of main approaches and main differences between them. This chapter also covers the theory of simultaneous localization and mapping and describes several approaches to the problem.

Chapter 3 describes the EAPD framework and its implementation in ROS. The basic integration of Gmapping with EAPD is also covered here.

Chapter 4 describes the implementation of the loop closure detection in the EAPD framework. The chapter covers introduction to the loop closure problem first and than is divided in two parts. The first part is dedicated to the integration of extended Gmapping functionality, such as loop closure, into the EAPD package. The second part covers the loop closure independently from the SLAM algorithm and is focused on the refinement of a polygonal map generated by EAPD.

Chapter 5 covers the description of the used hardware and the experiments on a TurtleBot mobile robot. The chapter presents:

1. Results of the use of Gmapping as a SLAM algorithm with the EAPD package.
2. Results gathered after modifying Gmapping and using extended Gmapping functionality in EAPD. This part also covers loop closure by Gmapping.
3. The results for a general refinement of the polygonal map executed during the exploration process.

Chapter 2

Robot exploration

In the mobile robot exploration task a robot equipped with sensors is used to operate in an unknown environment in order to maximize the knowledge over a particular area. The robot autonomously navigates through it's surroundings and builds a map of the environment. The incrementally built map is then used as a base for selecting further goals of exploration. The robot operates until it creates a complete map of the environment, or until it reaches some specific goal, i.e. finds a specific target.

Although mobile robot exploration is most suited for surveying areas which might be inaccessible or dangerous to humans, with a decrease in prices of small computer units and sensor equipment, mobile robot exploration became widely spread in other areas, increasing the need for further improvement and optimization of current methods.

The problem of robot exploration can be split in two main parts. In the first part, the robot has to challenge the problem of creating a map of the environment and maintaining knowledge about its own position in accordance to the map. The second problem is the exploration itself, mainly the selection of next goals of the exploration and navigation through the environment.

The EAPD framework focuses on map building and exploration goal selection but requires a suitable position estimation. Our intent is to use the EAPD framework with a TurtleBot mobile robot which provides basic movement odometry and is equipped with a laser rangefinder. An appropriate method has to be selected to provide a suitable position estimate.

Even though there are algorithms that can provide position estimation based only on the odometry data and laser scans without creating a map of the environment, due to the noise and error in these measurements, the estimated positions are often inaccurate and the accumulated error after few iterations is unacceptable. Therefore we decided to select one of the algorithms for simultaneous localization and mapping (SLAM), which are more reliable. A comparison of position estimates based solely on the odometry data and estimates based on odometry and laser scan is depicted in Fig. 2.1.

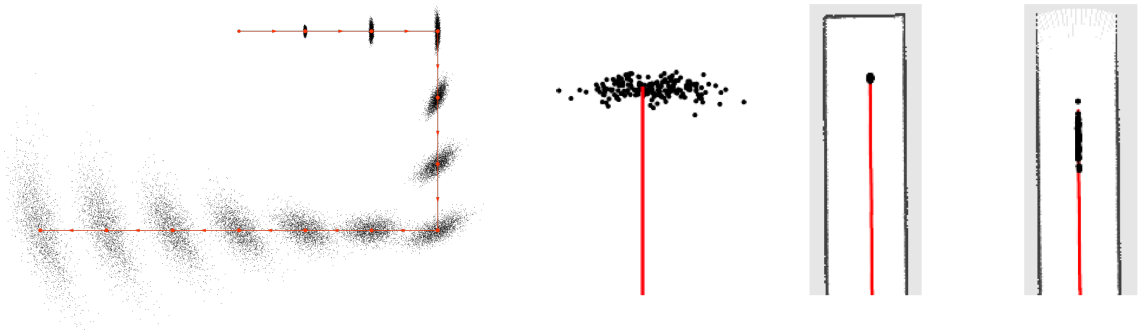


Figure 2.1: The left picture presents a visualization of particle distribution after a few iterations based on odometry. The right picture describes particle distribution based on odometry and laser measurements in several specific environments. Pictures were taken from [6] and [5].

2.1 Simultaneous localization and mapping

The problem of simultaneous localization and mapping is often compared to the "chicken and egg" problem. We need accurate position estimates to create a map using our observations, but we also need a good map of the environment to properly estimate the current position of the robot.

Since this work will be using the ROS framework, the focus on SLAM algorithms will be on those, that can be used with ROS and with a laser rangefinder. There are two suitable algorithms already implemented in ROS, the Gmapping package and the Hector Slam package. Due to better overall response of users to the package and previous personal experience with it, the Gmapping package has been selected for the use with the EAPD framework.

A noticeable selection of SLAM algorithms can be found on the OpenSlam website[4], which serves as a support for SLAM researchers and offers the possibility to publish their work. Some of these algorithms can also be used with ROS and provide extended functionalities. These algorithms were often used as a reference, while working on the loop closure problem in this work.

2.1.1 Gmapping

Gmapping package, presented on the OpenSlam website and described in [5] and [7], was later wrapped for the use with ROS and is maintained as a standard ROS package till date. Gmapping presents a Rao-Blackwellized particle filter as an effective mean to solve the simultaneous localization and mapping problem.

This approach is based on a particle filter where each particle carries an individual map of the environment. A particle creates a grid map containing occupancy probabilities for each grid cell, representing a possible obstacle at its location. As such the computation and memory requirements are strongly dependent on the number of particles used in the algorithm. Due

to advanced filtering and resampling techniques used in Gmapping, the number of particles required is relatively low and the default setting with 30 particles is usually sufficient for most environments.

Gmapping computes an accurate distribution of particles taking into account the movement of the robot and the most recent observation. The key idea is to estimate a posterior $p(x_{1:t}|z_{1:t}, u_{0:t})$ about probable trajectories $x_{1:t}$ of the robot given its observations $z_{1:t}$ and its odometry measurements $u_{0:t}$. This posterior is further used to compute a posterior over maps and trajectories:

$$p(x_{1:t}, m|z_{1:t}, u_{0:t}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{0:t}) \tag{2.1}$$

The process of the SLAM algorithm iteratively creates a map of the environment and updates positions of the particles. Every iteration contents of 5 main steps[5]:

1. *Sampling*: New generation of particles $x_t^{(i)}$ is obtained from the current generation $x_{t-1}^{(i)}$ by sampling from a proposal distribution $\pi(x_t|z_{1:t}, u_{0:t})$.

2. *Importance weighting*: Each particle is assigned an individual importance weight $w^{(i)}$ according to:

$$w^{(i)} = \frac{p(x_t^{(i)}|z_{1:t})}{\pi(x_t^{(i)}|z_{1:t})} \tag{2.2}$$

   These weights account for the fact that the proposal distribution π is in general not equal to the target distribution of successor states.

3. *Resampling*: Particles with low weights are replaced by samples with higher weights. Due to a limited number of particles it is important to approximate a continuous distribution. Resampling also allows to apply a particle filter in situations in which the true distribution differs from the proposal.

4. *Map Estimation*: For each particle position $x_t^{(i)}$ is computed a corresponding map estimate $m_t^{(i)}$, based on the previous trajectory and the history of observations according to $p(m_{1:t}^{(i)}|x_{1:t}^{(i)}, z_{1:t})$.

5. *Output selection*: The particle with the highest importance weight is determined as the *best* particle and it's position is published as the current position of the robot, together with its corresponding map estimate.

2.2 Mobile robot exploration

With the problem of localization and mapping solved by the Gmapping package we can now focus on the exploration itself. In the exploration problem the robot navigates through the

environment and incrementally updates a map which is used as a model for further exploration steps. The other tasks of exploration algorithms are selection of future goals, planning and navigation through the environment. A diagram describing the exploration process can be seen in Fig. 2.2. Individual parts of the diagram will be described in the next part of this chapter.

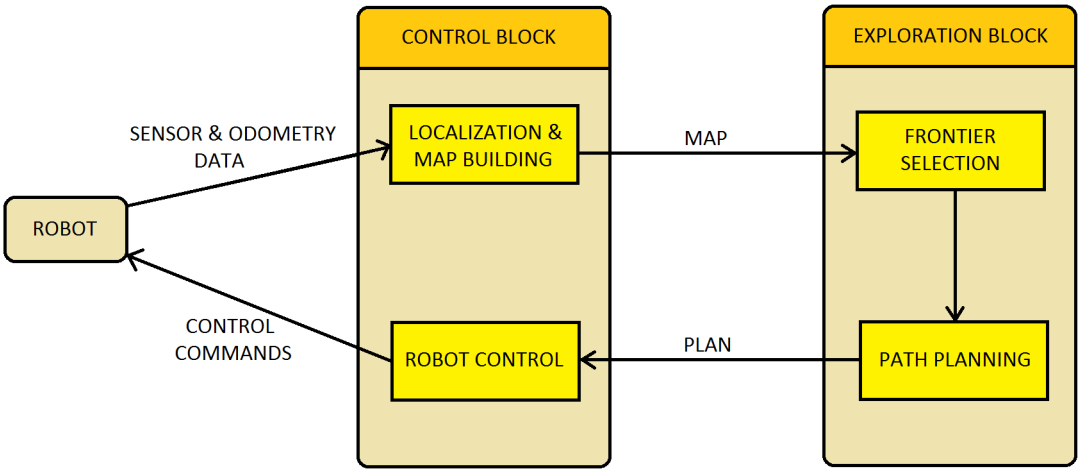


Figure 2.2: A diagram describing the exploration process.

While having some common basis, all of these operations are vastly dependent on the representation of the environment. This chapter will first describe the common base for all representations. Features specific for other representations will be described later with a focus on the most popular *2D occupancy grid* and the *polygonal domain* which will be used in this thesis.

2.2.1 Frontier selection and path planning

After completing the initial part of the map from robot's starting position, the first task for exploration algorithms is to select the next desired position towards which the robot can move to. Such task is usually connected with *frontier selection*.

Frontiers can be described as borders between unoccupied and unexplored areas, these are the parts which the robot has yet to explore. A visualization of frontiers over an occupancy grid is shown in Fig. 2.3. However not all frontiers are suitable for goal candidates, frontiers inaccessible by the robot or insignificantly small ones are generally excluded from the selection. Therefore when selecting a suitable frontier for a future exploration goal the frontiers have to be filtered and the goal has to be selected from the remaining ones. In a situation where there are no more exploration candidates, the robot usually reports the end of the exploration and either terminates the exploration process or stays on stand by.

There are many possible approaches to frontier selection, one of the easiest is the selection of the frontier closest to the robot. Other techniques are more closely connected with path

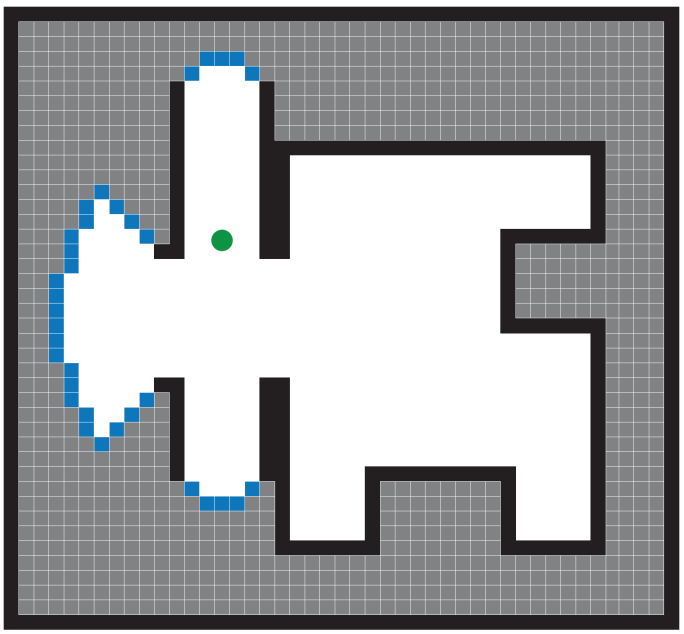


Figure 2.3: A visualization of an occupancy grid map. Frontiers are marked in blue color, obstacles are black and the unexplored area is gray. The robot is represented by a green circle.

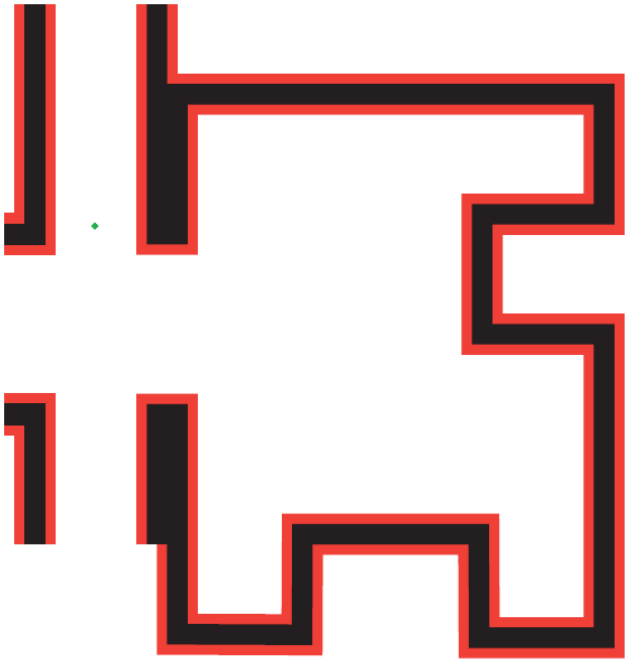


Figure 2.4: A visualization of an inflated map from Fig. 2.3. The original obstacles were inflated by the size of the robot radius (marked with a red color) and the robot representation has been replaced with a single point in the C-space.

planning, selecting the frontier accessible by the most suitable path, advanced techniques based on information gain or approaches suited for multiple robots exploring the same area.

When the next goal is selected the exploration algorithm has to compute a suitable path for the robot based on its current representation of the environment. A common approach for path planning algorithms is to first create a configuration space (C-space), in which the robot can freely move. C-space can be obtained i.e. by inflating the obstacle representations in the environment by the size of the robot radius as shown in fig. 2.4, this approach is also used in the *EAPD* algorithm and will be described in detail in section 3.2.

After the creation of the C-space, a graph representation of the C-space is created and searched for an optimal path. The found path is often further processed, taking into account physical constraints of the robot, i.e. it's motion model and maximal turn rate. Due to a high number of various approaches, we will focus only on the description of path planning in *polygonal domain* used in *EAPD*.

2.2.2 Exploration using a 2D occupancy grid

An occupancy grid represents a map of the environment as an evenly spaced field of variables (cells) each representing a probability of a presence of an obstacle at the specific cell. The probability is incrementally computed from all previous observations and each cell is marked either as occupied if the probability exceeds a certain threshold, unoccupied if it doesn't or unexplored if there was no observation for the cell so far. In a 2D occupancy grid with a specific resolution (i.e. Gmapping uses 0.05m as default) the layout of grid cells usually corresponds with x and y coordinates.

Grid maps are currently one of the most popular representations in the mobile exploration problem. As such, they offer a large variety of algorithms which are already implemented and proved functional, offering the users a large variety of techniques for use. Being based on probabilistic estimates of it's state, occupancy grids also offer high reliability while working with inaccurate position estimates or laser measurements with high noise.

The downside of this representation is its large demand on the memory, making it disadvantageous for the use in larger environments.

2.2.3 Topological graphs

While exploring an indoor environment it is often advantageous to divide structures to a set of rooms and corridors. Instead of frontiers the robot can then be assigned to explore certain rooms in the structure. Hybrid exploration algorithms often use topological maps together with other mapping approaches. Topological maps provide advanced information on the structure composition, while a different map (i.e. grid map) is composed to provide more exact model of the environment.

A topological map is a set of places represented as graph nodes and linked by edges. Each node and edge can have additional characteristics used as a base for exploration algorithms or data processing. For example, *gate* or *doorway* nodes are often used as nodes in between separate rooms, providing the width of the gate and helping further exploration decisions.

A hybrid approach connecting the topological representation with an occupancy grid map was presented in [8]. The method divides the metrical grid map in equally sized segments and uses these segments as a base for the topological representation. Each segment is then represented by an *area* node and the connections between neighboring cells are represented by *gateway* nodes. An example of the environment representation can be seen in Fig. 2.5.

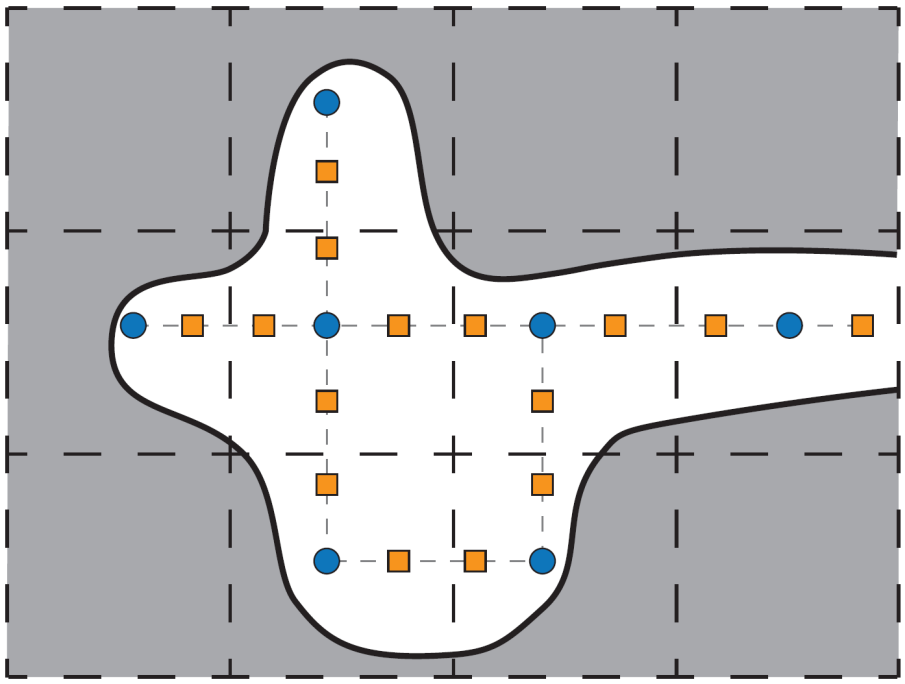


Figure 2.5: A visualization of a topological map overlay over a metric map. The area nodes are represented by blue circles and the gateway nodes are represented by yellow rectangles.

The advantage of this approach is the connection of global planning over the topological map and local planning in the individual segments of the grid map. The global plan is calculated using the topological representation and the local planning is computed whenever the robot enters a new segment in the metric map. Using this approach, there is no need to compute the full plan over the grid map at once reducing the peaks in the computation complexity. This approach also reduces the complexity when the desired path is changed midway, as there is no need to compute the path over the remaining grid map segments.

2.2.4 Exploration in a polygonal domain

T. Juchelka introduced polygonal domain representation and provided an usable framework for exploration in [1]. A map representation in a polygonal domain has significantly lower memory requirements than the occupancy grid while preserving a higher detail. The polygonal representation also allows easier processing in various ways, i.e. rotation or scaling.

This representation is based on polygons, defined by a finite number of line segments. These segments are called edges, and a point where two edges meet is called a vertex. Vertexes are represented by their x, y coordinates and the type of edges leading in and out. A map is composed of edges representing either obstacles or frontiers as can bee seen in Fig. 2.6. The map is focused only on relevant parts of the environment, thus demanding less memory. Within the map the inside of the polygon containing the robot is the unoccupied space.

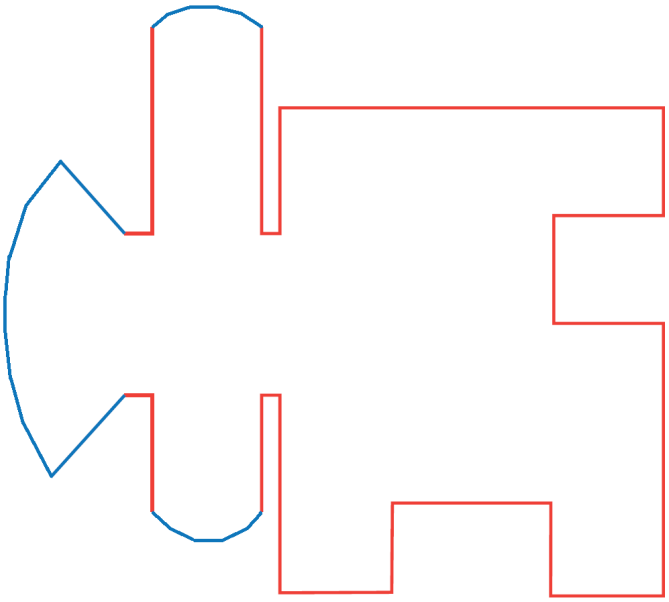


Figure 2.6: A polygonal representation of the map from Fig. 2.3. The obstacles are marked in red color and the frontiers are marked blue.

The occupancy grid approach is advantageous over the polygonal in two main aspects. The first advantage of the occupancy grid is easier calculation of posterior probability of past measurements in relation to the map. Secondly, as it has been more popular, a large variety of support tools has already been implemented for the occupancy grid and these tools are still missing for the polygonal domain. This work addresses the second issue, proposing a tool for the processing of a loop closure occurrence.

Chapter 3

Introduction to EAPD

The Exploration Algorithms in a Polygonal Domain(EAPD) framework was created by T. Juchelka[1], and was later extended by V. Lhotský[3]. The framework was implemented in C++ as a ROS package and allows exploration in a polygonal domain using various exploration strategies. This chapter contains a brief introduction to ROS followed by a description of the *EAPD* framework and later the integration of *Gmapping* SLAM algorithm.

3.1 ROS - Robotic framework

The Robot Operating System(ROS) is an open-source set of libraries and tools for development of robot applications. More information on ROS can be found on its official website [2]. ROS offers a large set of drivers, state-of-art algorithms and developer tools.

ROS employs a system based on *nodes*, separate executables which can be individually designed and run. Nodes perform individual tasks and ROS provides communication between them in the form of standardized messages published on specific topics. These topics work as broadcasts and any other node is able to receive the message. The separation of the process in individual nodes allows each subprocess (/node) to be written in different language, run on different frequency and to be modified without affecting the rest of the system. It is also beneficial as two nodes with similar functionalities can be easily exchanged.

Most important ROS tools used for work in this thesis are:

Rviz - a framework used for the visualization of broadcasted topic messages such as robot position, map and laser scans.

RQT - a framework fro GUI development for ROS.

Catkin - a low-level build system macros and infrastructure for ROS.

TF package - a highly practical tool for handling transformations between coordinate frames over time. This is useful while handling positioning of sensors on the robot and robot positions

in relation to the map. An example of robotic transformations in the system can be seen in Fig. 3.1 created with the help of the *RQT* development tool.

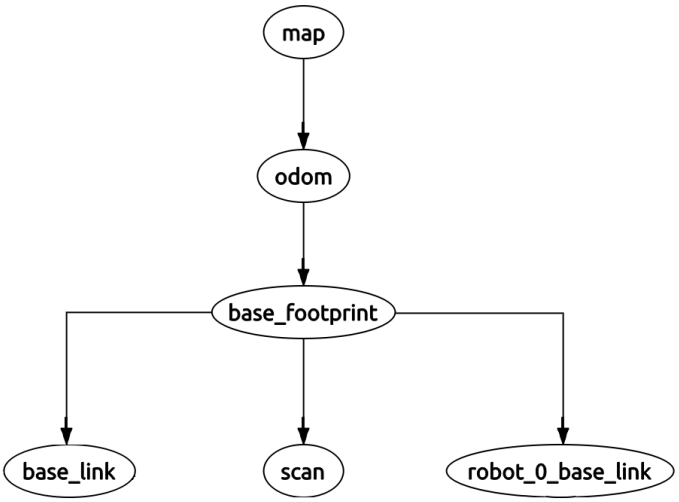


Figure 3.1: A visualization of transformation frames in ROS. The image presents connections between individual coordinate frames.

ROS also provides a launch file system allowing an user to run more nodes at the same time. Launch files also provide other useful options, such as setting parameters for the nodes at the startup, remapping specific topics or directly changing specific ROS settings. All lunch files used in this work are available on the CD, provided with the thesis.

3.2 EAPD

The EAPD package implemented in C++ allows exploration in a polygonal domain, providing three main applications executable as individual ROS nodes and supported by other ROS libraries. Those nodes are:

Robot node

The *robot* node receives data from laser scans and listens to transformations between coordinate systems. The information is processed and published after a transformation to a global map. While using several robots, each of them runs its own *robot* node which is dedicated to its own specific topics.

The *robot* node also listens to the path planning topic from the *planner* node and sets navigation goals to the Smooth Nearness Diagram (SND) algorithm [9], integrated into the

EAPD package. The SND algorithm is responsible for local navigation and obstacle avoidance, commanding the robot velocities with the help of laser-scan and odometry data.

The node subscribes to topics:

- `/robot_i/base_scan` - scan messages from the laser rangefinder, with relation to the *i*-th robot
- `/robot_i/path` - path messages from the *planner* node

The node publishes messages on topics:

- `/laser_scan` - laser scans published to the global map. This topic concentrates laser messages from all robots.
- `/robot_i/cmd_vel` - velocity command

Map node

The *map* node subscribes to laser-scan data and listens to transformations from the robots. This node creates a polygonal map of the environment with a help from the clipping library[10]. The map is then published for the *planning* node and optionally visualized in Rviz.

The node maintains a map in a form of two polygonal structures separated to represent the free-space and obstacles independently. Every measurement is split into obstacles and frontiers and merged with the map. The map representation is later extended by offsetting the map accounting for the size of the robot during the planning process. The offset representation is kept separately. An example of the offset map can be seen in Fig. 3.2.

The node subscribes to topics:

- `/laser_scan` - laser scan messages from the *robot* node

The node publishes messages on topics:

- `/map_global` - global map for all robots
- `/visualization_marker` - visualization related to the map

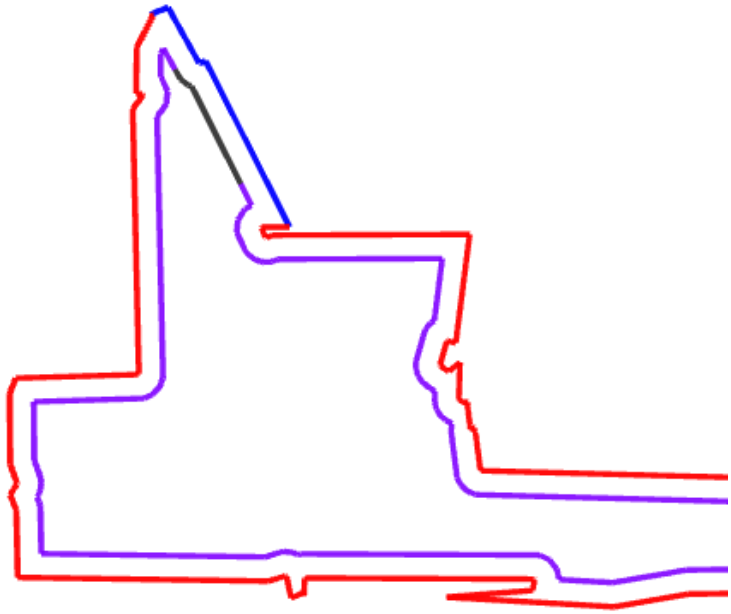


Figure 3.2: An example an offset polygonal map.

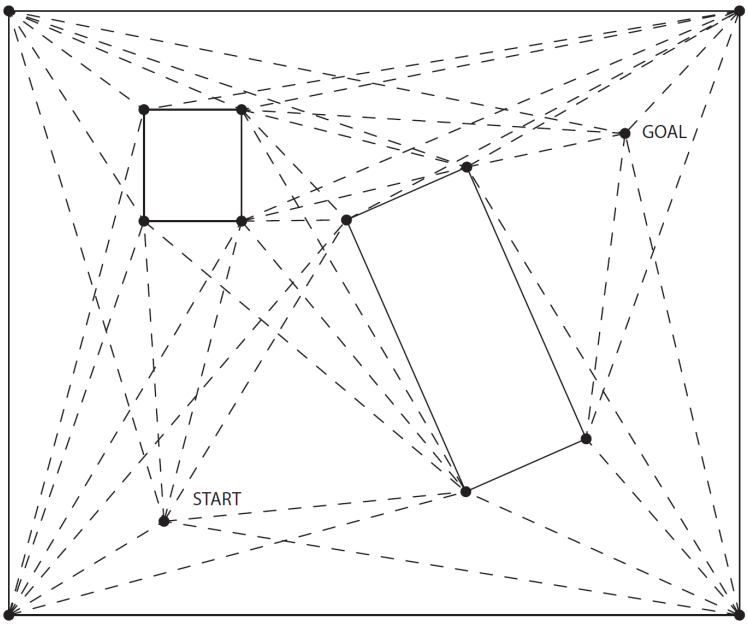


Figure 3.3: An example of a visibility graph.

Planner node

The *planner* node is responsible for frontier selection and path planning. The node receives the map and selects a frontier as an exploration goal based on the exploration strategy. Then a suitable path is computed and published for the *robot* node.

The path planing itself uses Dijkstra algorithm with the use of a visibility graph, created from the polygonal map. The visibility graph is created from vertices in the map polygons. These vertices are taken as nodes in the graph and the nodes are adjacent (connected with an edge) if they can see each other. An example of a visibility graph is shown in Fig. 3.3.

The node subscribes to topics:

- `/map_global` - global map for all robots

The node publishes messages on topics:

- `/robot_i/path` - path goals published to a specific robot
- `/planning` - visualization related to the planning

3.3 Integration of Gmapping with EAPD

As a part of this thesis the Gmapping SLAM algorithm described in 2.1.1 was integrated into the EAPD package to provide an accurate position estimate. ROS is based on a system using independent nodes, therefore adding the Gmapping node is relatively easy. After launching the node, it tries to subscribe to the *tf* topic providing coordinate system transformations for the laser, base and odometry, and to the *scan* topic for the laser measurements. An entry adding the Gmapping node to the launch file can be seen in Listing 3.1.

```
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" args="scan
   :=robot_0/base_scan _base_frame:=robot_0/base_link" >
    <param name="linearUpdate" value="0.3" />
    <param name="angularUpdate" value="0.05" />
    <param name="maxUrange" value="5" />
  </node>
```

Listing 3.1: Gmapping SLAM section in the launch file.

The entry starts the `slam_gmapping` node from the *Gmapping* package, under the same name. The parameters set are the *linearUpdate* and *angularUpdate* Gmapping parameters, defining the change in robot position after which the grid map in Gmapping should be updated, while the *maxUrange* parameter limits the maximal range of a laser scan, that should be taken into account, these parameters affect the inner process of Gmapping.

A special focus should be on the *args* section. In this section topics `scan` and `_base_frame` are remapped to topics `robot_0/base_scan` and `robot_0/base_link` respectively. Since the transformation and laser scan topics are bound to a specific robot and can be used under various names, the gmapping node has to be informed which topics it should listen to. The EAPD framework was developed to work with multiple robots, therefore its topics are numbered and dedicated to individual robots.

Chapter 4

Implementation of a loop closing algorithm

As a part of this work, the refinement of a polygonal map in case of a loop closure detection was implemented. Introduction to the loop closure problem as well as to current approaches in the area are given in the first part of the chapter, followed by the integration of Gmapping loop closure detection into the EPAD package. Gmapping has a specific approach to the topic, therefore another method on refining of the map was proposed and implemented. This method is described in the end of the chapter.

4.1 Introduction to the loop closing problem

A loop closure refers to a situation where a robot circles some part of the environment and detects that it has already been at the same place before. A visualization of such situation can be seen in figure 4.1, where we can see the real path of the robot, the estimated path and the required *loop closure correction*. At that point the loop closing algorithm tries to correct robot's position, trajectory and map accordingly. Such correction can vastly improve the robustness of localization and mapping algorithms.

Current approaches regarding the problem differ both in the way the robot detects the loop closure and the way the robot deals with the position and map correction. These differences are vastly dependent on equipped sensors, map representation and algorithms used for localization and mapping.

4.1.1 Loop closure detection

The robot deals with several difficulties when determining if his current position corresponds with some of the previous positions.

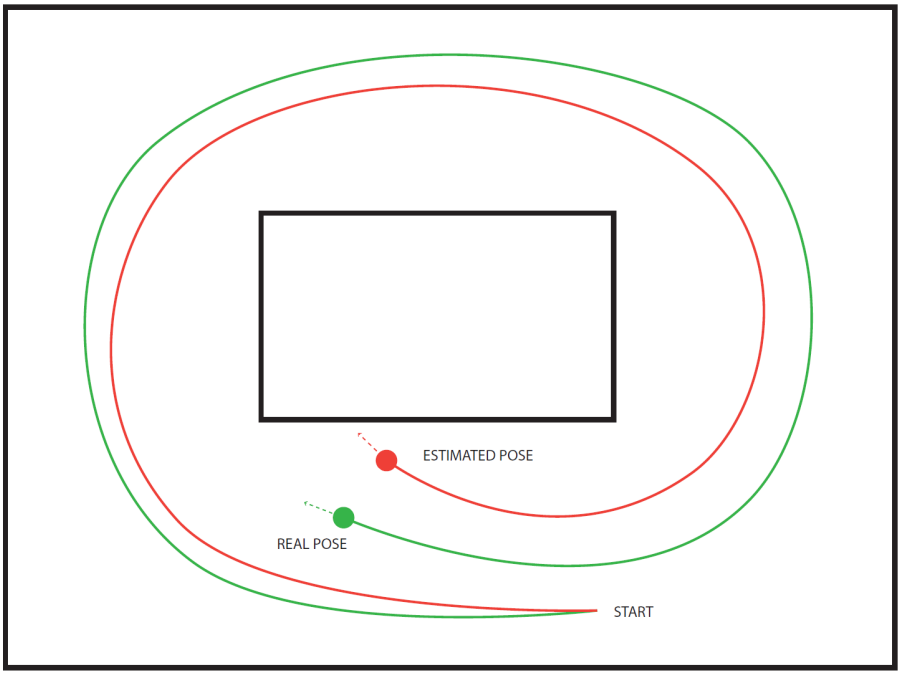


Figure 4.1: A visualization of a loop closure.

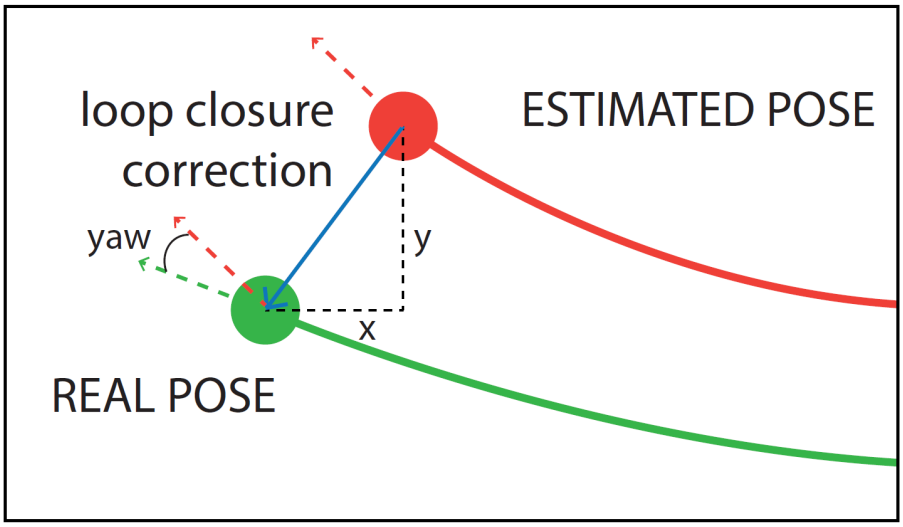


Figure 4.2: A close-up of figure 4.1. The *loop closure correction* is marked with a blue arrow. The *correction* values *x*, *y* and *yaw* are depicted in the visualization.

The main problem is the certainty with which the loop closure is detected. Although i.e. image comparison or scan-matching use different approaches, after the processing the situation is usually represented with a probability value which has to exceed a certain threshold. The setting of this threshold is crucial. If the threshold is too high the algorithm can miss multiple loop closure points and if it is too low false loop closures are detected.

Another issue is the computational complexity of the detection algorithm. If the robot tries to match its surroundings to all measurements taken in the past, the computation time increases exponentially with the number of measurements. Such problem can be solved by comparing only the measurements taken at positions in a certain area around the currently estimated position of the robot, where the compared area increases with the uncertainty of the position. Other common solution to decrease the computation requirements is to run the loop closure detection algorithm only once every few iterations.

Robots equipped with different sensors offer various approaches to the detection problem and can be obstructed with different challenges. For example uniform environments with similar layouts or repeated visual patterns can often increase the difficulty of the detection.

1. **Laser-Scan**: Although range-finders can get a fairly accurate 2D layout of the environment, it is difficult to find a proper match for the detection of loop closures. In the indoor environments, corridors and other rooms have often similar structures, that can not be differentiated with the data from range-finders.

   Scan matching is a wildly used approach in these situations. Scan matching is a process where a range scan obtained from the measuring device is rotated and translated to match an a priori map. The main problem with scan matching are false positive detections.

   The problem of false positive detections can be solved using advanced probabilistic methods, but cannot be fully eliminated. An interesting approach was presented in [11], where it is proposed that a map patch integrating several scans is created and used for the scan matching. The patch represents a piece of the map gathered over the last several meters, offering more distinctive characteristics for the scan matching.

2. **Camera**: Approaches using a camera image offer a large amount of information available for the detection algorithm. Specially while using a RGB camera with a high resolution, the certainty of a loop closure can be very high. The disadvantages are that the processing of the images takes a lot of computation power and the image comparison is prone to failure due to unstable light conditions. A comparison of several approaches, including camera recognition is presented in [14].

3. **Combination of multiple sensors**: A combination of sensors can lead to higher reliability and precision in the detection. For example GPS navigation can successfully prevent false positive closure point detections, while laser-scan can assure precision on local scale. Another approach was presented in [12], the approach connected loop closure detection using an RGBD camera, with a laser-scan based SLAM algorithm. However such hybrid approaches can increase computational requirements and the cost of the system solution.

4.1.2 Position and map correction

Once the loop closure is successfully detected, the algorithm has to integrate this new knowledge into its beliefs. Simple update of the current position could be made but depending on the application, different parts of the program can be affected.

The algorithm might try to update information on previous paths and refine the generated map, based on the *loop closure correction*. This requires information on the previous states of the robot and knowledge of the starting point of the loop. The implementation of polygonal map refinement is described in detail in section 4.3.

The loop closure detection can be a crucial information for topological representations of the environment, connecting the current node in the graph with a previously visited one. In such situations the higher logic can trigger the refinement of the map.

In a case of SLAM algorithms based on particle filters (such as Gmapping), loop closure can lead to a preference of particles which are closer to the pose estimate from the new data. This will be described in more detail in the next section.

4.1.3 Loop closure in Gmapping

Gmapping SLAM uses a particle filter, where each particle carries its individual map of the environment. It would be overly demanding to refine all individual maps, therefore the loop closure implementation in Gmapping is based on a different approach.

Particles are assigned an individual importance weight every iteration, these weights are later used when producing a new generation of particles. The evaluation can be affected by the correlation of the current particle scan and the posterior map, which can result in a preferential treatment of some particles.

Gmapping uses scan matching to find a correlation between measurements projected from the pose of the corresponding particle and the posterior map in its surroundings. The better matching particles are favored during the creation of the new generation, reducing the uncertainty about robot's position. Figure 4.3 visualizes trajectories of particles shortly before and after it came in the vicinity of previously taken data.

As specified in section 2.1.1, the importance weighing also results in a selection of the *best* particle. It's position is then published as the new position of the robot, together with it's map.

4.2 Integration of the extended Gmapping functionality into EAPD

While Gmapping has the advantage of a path correction after a loop closure, the correction itself is a part of a standard particle weighting and selection process. The loop closure usually leads to a switch of the *best* particle or eventually to the resampling of particles.

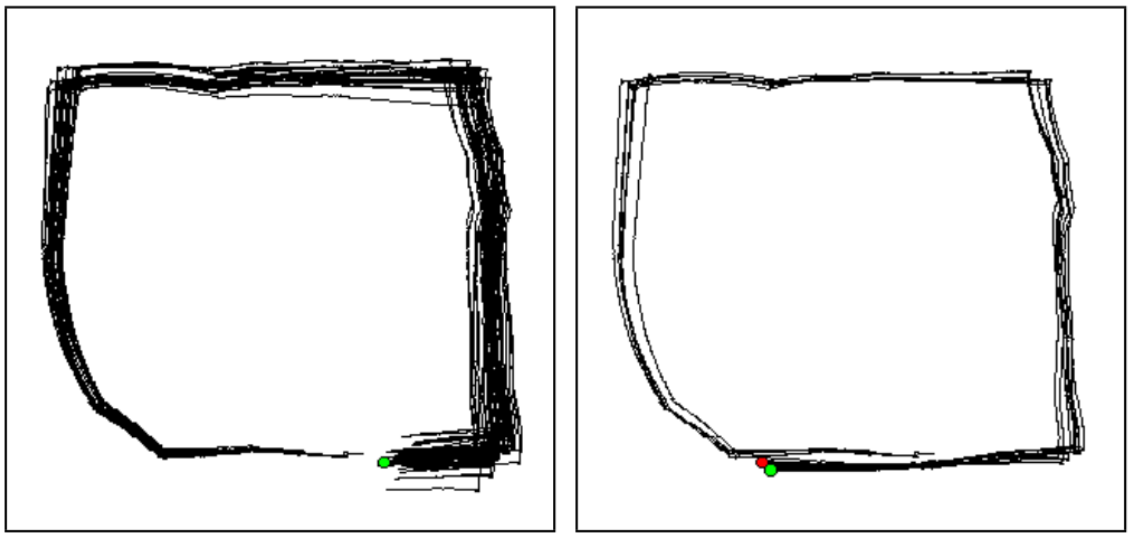


Figure 4.3: Trajectories of 100 particles shortly before (left) and after (right) closing the loop. The figure was taken from [13].

As it is a result of the standard selection process, there is no distinct call informing about the loop closure. When switching particles we also have to switch the maps, as each particle carries its own. If we consider these issues, it would be unreasonable to perform the refinement of a polygonal map from EAPD. The refinement would not only require information on all previous particles and their positions, but more importantly it would be triggered whenever a switch in the *best* particle occurs, resulting in the refinement on frequent occasions even without closing the loop. Since each change could lead to a refinement of the map a few dozens or even hundreds iterations back we decided to use a different approach.

The approach proposed for the integration of loop closure correction from the Gmapping package to the EAPD package is to transmit relevant information about all particles from the Gmapping package to the EAPD *map* node, responsible for creation of the polygonal map. Individual maps are continuously created for each particle and the map belonging to the *best* particle is published as the result and used for the exploration. Since polygonal maps require a small amount of memory, keeping several dozens of them at once has little effect on the memory requirements. During the experiments the control unit also proved to have no issue with the increase in computational complexity caused by the processing of multiple polygonal maps every iteration.

The `slam_gmapping` node from the Gmapping package was modified to publish information on all particles used in the algorithm to a `gmapping_particles` topic. The particle information from Gmapping is transmitted to the *map* node as a custom message containing several standard information structures. A separate message file had to be created to define the structure of the message. The content of the created *prtcl.msg* file is shown in Listing 4.1.

```
Header header
geometry_msgs/Twist[] poses
int32[] p_index
sensor_msgs/LaserScan l_scan
int32 best_index
```

Listing 4.1: File *prtcl.msg*, defining a customized message

The necessary particle information contains a standard header used by ROS messages, an array with particle positions in a form of a standard ROS *Twist* message, an array of indexes corresponding to particles from the previous generation, information on the *best* particle and a laser scan message corresponding to this specific batch of particles.

The *map* node in the EAPD package was modified to subscribe to the `/gmapping_particles` topic and process the new information in the following manner:

1. When a new message in the `/gmapping_particles` topic is detected a *ParticleSet* function is triggered to process the new batch of particles.
2. The node keeps a memory of maps, created by the previous batch of particles. When a new particle is processed, the algorithm takes corresponding map, based on the previous index number from the message, and updates the map using the particle position and laser scan information.
3. When all the new maps are created, the algorithm selects the map belonging to the *best* particle, based on the `best_index` from the message file, and publishes it on the `/map_global` topic.

A `gmapping_particles` parameter was integrated into the EAPD package. This parameter is used to specify the number of particles expected by the *map* node and also works as a switch between the functionality following all the Gmapping particles and the original functionality of the EAPD package using the transformation topic for position information triggered by a newly published laser scan message. The parameter is optional and set to 0 as default, forcing the original functionality of the EAPD package.

4.3 Implementation of polygonal map refinement in case of loop closure detection

The integration of the loop closure functionality with the use of Gmapping is specific to particle based SLAM algorithms. If the EAPD package is used with different SLAM or loop closure detection algorithms a more general solution is desired.

A new "loop" node was integrated in the EAPD algorithm as a part of this thesis. The *loop* node is based on the original *map* node and provides the same functionality, extended by an

option for a map refinement. The *loop* node was written as a proposed solution to the map refinement problem. A visualization of the desired map refinement functionality can be seen in Fig. 4.4. The experiments proving the functionality of the node are described in section 5.

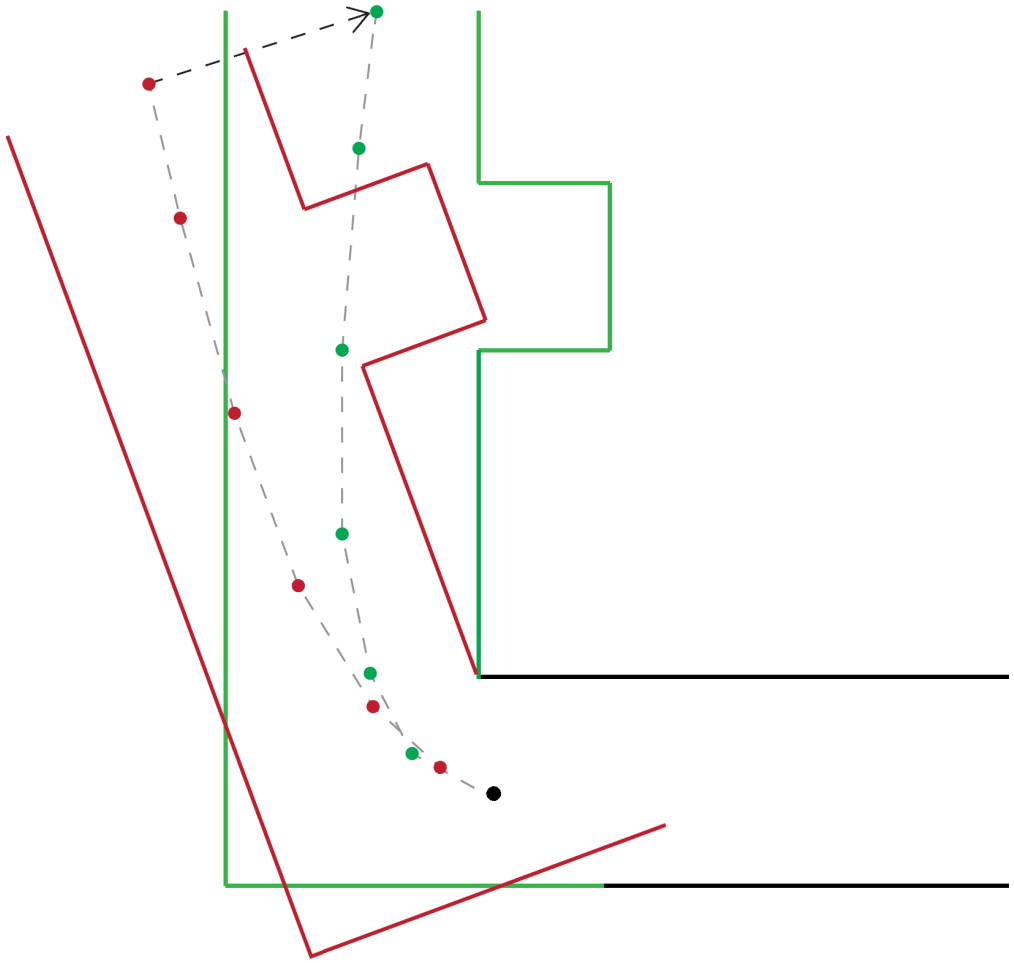


Figure 4.4: A visualization of a map refinement. The initially incorrect path is visualized in red color, as well as the section of the map generated along the incorrect path. The green color represents the corrected path and the newly refined map section. The dashed arrow represents the path correction.

The *loop* node saves previous positions of the robot and all measurements taken by the laser rangefinder, and uses it for the map refinement. The refinement is triggered by an outside message on the `/map_correction` topic, where a customized message contains information on the *loop closure correction*.

As described in figure 4.2, the robot travels through the environment, circles back to a previously visited position and has to adjust its pose information using a loop closure *correction*

value. The loop closer *correction* value indicates the difference between the *current* position and the *previous* position. Our initial assumption is that the `/map_correction` topic published from a "loop detection" node provides information on the *previous* and *current* positions and the *correction* value.

The initial functionality of the *loop* node is the same as the *map* node. When the node receives a message from the `/laser_scan` topic it processes the scan into the polygonal map, and saves the laser scan message and the current position during the process. For convenience the node also saves every X-th map. The number can be changed using a newly added `map_frequency` parametr, set to 50 as default. Keeping "checkpoints", the program doesn't have to rebuild the whole map, but can more or less accurately refine only the part affected by the loop closure.

The refinement process can be described in 3 steps.

1. **Trigger**: The *loop* node receives a message from the `/map_correction` topic containing the *previous* position, *current* position and the loop closure *correction*. This triggers a *PathChange* function.

2. **Path change**: The function iterates through the path of the robot, starting with the *previous* position. The path *correction* contains a difference between the current position and the desired position of the robot in *x*, *y* coordinates and the angle correction *yaw* value.

   The *correction* values (x_c, y_c and yaw_c) are iteratively distributed between individual position changes. Each position P_i can be represented by its coordinates x_i, y_i and yaw_i. For each position touple $< P_i, P_{i+1} >$, the additional change is proportional to the ratio between the difference in positions and the total movement over the loop.

   These vales are computed individually for each coordinate. An example of a change in the x coordinate of position P_i can be computed using the total movement over the loop in the x coordinate $diff_{total} = \sum_{j=1}^{current} |x_j - x_{j-1}|$ described by a sum of changes in the x coordinate from the beginning of the loop till the current position. The change of the movement can be computed using the following equation:

   $$diff_i = \frac{x_c(x_i - x_{i-1})}{diff_{total}} \tag{4.1}$$

   The new x coordinate for the position is then computed as:

   $$x_i = x_i + \sum_{j=1}^{i} diff_j \tag{4.2}$$

3. **Map refinement**: The last map saved before the starting position of the loop is selected for the refinement. The map is updated iteratively using the newly modified path of the robot and saved laser scan measurements.

Chapter 5

Experimental results

The task of the EAPD framework is to explore its surroundings and create a polygonal map of the environment. A series of experiments was performed to confirm the functionality of implemented algorithms. A common setup and hardware is described in the first part of the chapter. The rest of the chapter is dedicated to specific setup, experiments and results of individual implementations.

During the process of the implementation and debugging of the algorithms the Stage simulator [15] was used extensively. The final experiments were executed with a real robot, to confirm the functionality in a real environment.

The Stage simulator[15] was used extensively during the process of the implementation and debugging of the algorithms.

5.1 Hardware

The experiments were performed using a TurtleBot mobile robot equipped with the SICK LMS 111-10100 laser rangefinder. All the algorithms and computations were run on an INTEL NUC5i5RYK control unit.

5.1.1 TurtleBot

TurtleBot with the Kobuki robot base is a popular platform for researchers and students. The Turtlebot meta package in ROS provides basic drivers for running and using the robot. The robot provides odometry information and data from tactile sensors placed at the front. Robot's movement is supported by two differential motors that can be controlled from the ROS interface. The robot is equipped with its own power source, making it a self sufficient unit, and can be connected to the control unit with a single USB cable.

A hardware mounting kit that can be attached to the base is provided with the robot, supporting the attachment of additional equipment(i.e. sensors or a control unit). Although the mounting kit can be freely customized a few modifications were made to support the laser rangefinder. The rangefinder was mounted to the construction and placed approximately 15 centimeters above ground. The low placement was chosen to prevent the sensor from the overlooking of obstacles.

An external 12V battery providing power supply for the laser and the control unit is fixed to the mounting kit by customized 3D printed holders. The battery was attached to a simple On/Off switch and connectors for the laser and the control unit.

Both the laser and the battery had to be properly fixed. Their weights are considerably heavy in comparison to the robot and an improper placement could affect the overall movement of the robot or even result in a system failure. The TurtleBot mounted with the rest of the equipment can be seen in Fig. 5.1.

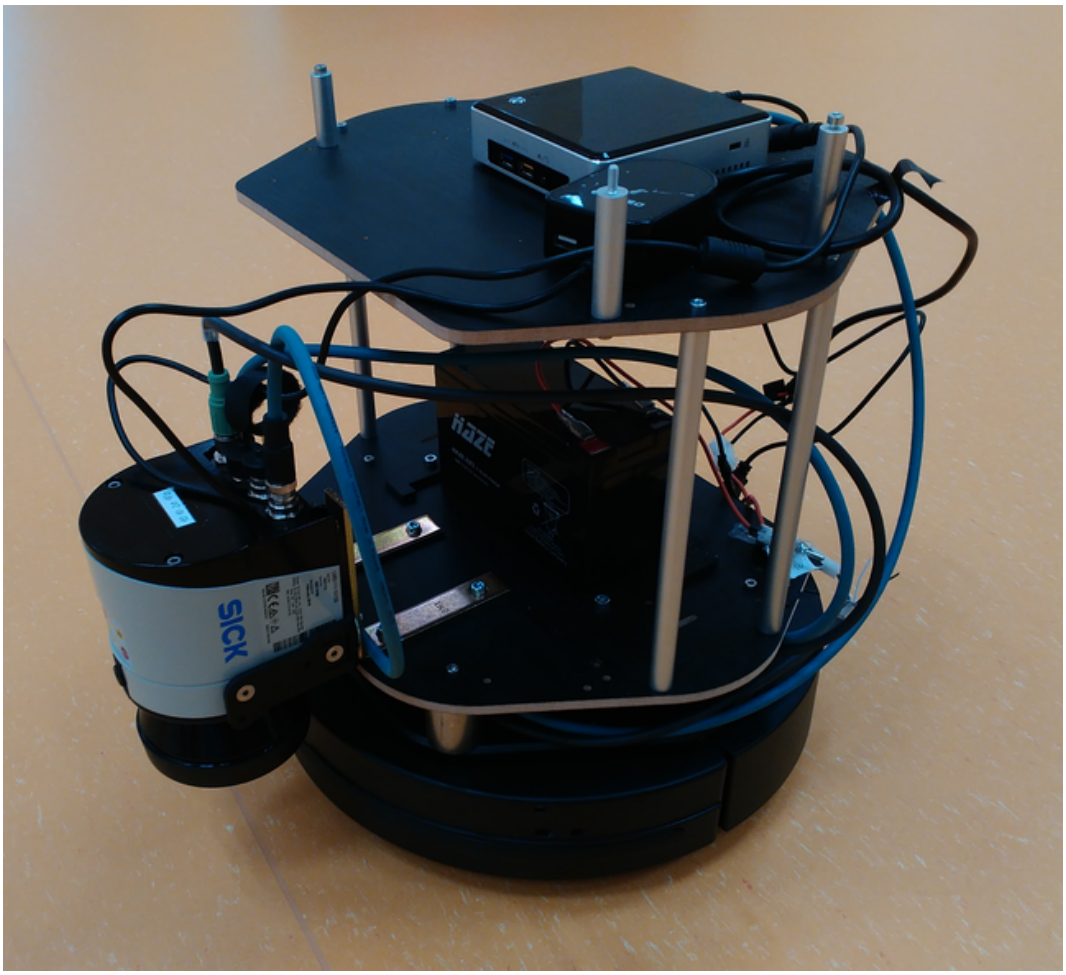


Figure 5.1: The TurtleBot mobile robot equipped with the laser rangefinder and the control unit, connected to a external battery.

5.1.2 LaserRangefinder

The SICK LMS 111-10100 laser rangefinder[16] chosen for this work has a maximal range of 20 meters and an aperture angle of $270°$. An operating range diagram for the laser can be seen in Fig. 5.2 , the ranges are more than sufficient for the purposes of the exploration algorithm. The wide range and robust construction of the laser is also advantageous for a possible future use in outdoor environments.

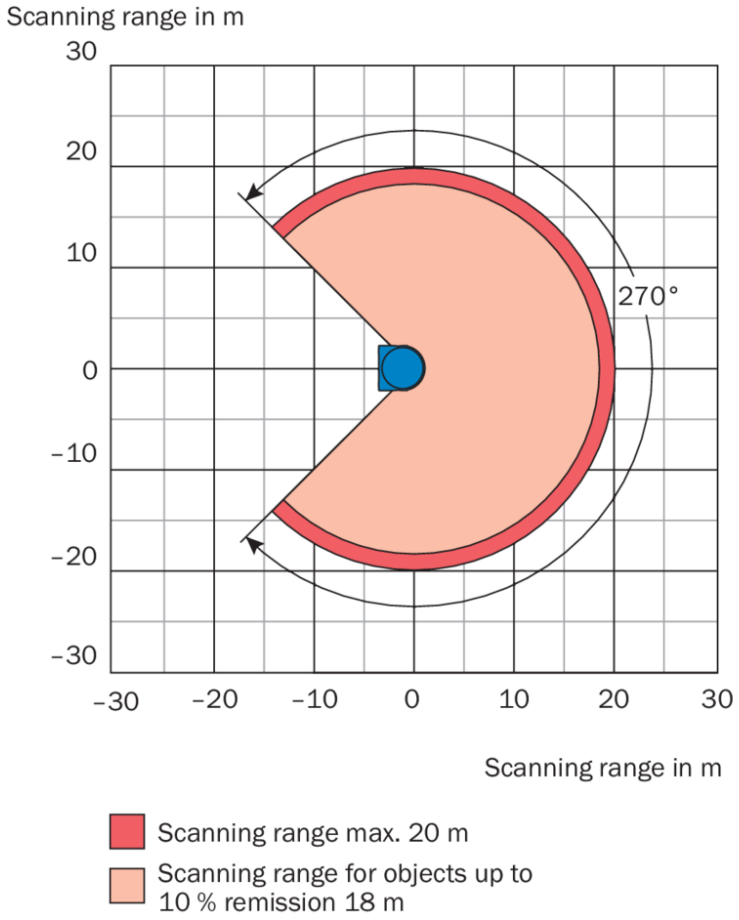


Figure 5.2: The operating range diagram for SICK LMS 111-10100 laser rangefinder. The image was provided in [16].

With an angular resolution of $0.5°/0.25°$ and a scanning frequency of $25Hz/50Hz$, the laser is able to provide accurate measurements in high detail even in higher velocities.

The laser is connected to the control unit using an ethernet connection. A driver for the LMS models is already implemented in ROS but due to the specific version of the sensor the IP address of both the laser rangefinder and the Intel NUC control unit had to be changed for a correct functionality. An entry in the launch file starting the *lms1xx* node and setting the necessary parameters, can be seen in Listing 5.1. The IP address of the control unit had to be synchronized with the laser and set to 10.10.20.181.

```
<node pkg="lms1xx" name="lms1xx" type="LMS1xx_node" output="screen">
  <param name="host" value="10.10.20.180" />
  <param name="frame_id" type="string" value="/scan" />
</node>
```

Listing 5.1: lms1xx driver section in the launch file.

The sensor is mounted to the robot in an upside-down position, this enables the laser to get measurements from a position, which is relatively close to the ground (15 cm), allowing a proper response for most of the obstacles faced during the exploration. There is a possibility to set the position of the laser upside down using the transformation library in ROS, however not all packages are compatible with these settings, therefore a separate node that would rewrite the measurements had to be written. The `laser_switch` node was written to revers measurements taken by the sensor. The node could also be used to trim the measurements in case the robot sees itself or to remove invalid values, coming from the measurements. The source code of the `laser_switch` node is available in the CD, provided with this thesis.

5.1.3 Intel NUC mini computer

The INTEL NUC5i5RYK mini computer uses a dual core 1.6 GHz processor, 16 GB RAM and is running the Ubuntu 14.04 operation system. The unit offers wi-fi and bluetooth connection, together with and ethernet connection and 4 USB 3.0 ports. The version of ROS used is the Indigo version.

5.2 Environment

The experiments with a real robot should reveal possible issues with noise in sensory data, low precision of the localization algorithm or the ability of the exploration algorithm to deal with an external movement in the environment, i.e. passers-by or opened doors.

Three sets of functionalities were tested, the integration of Gmapping as a SLAM algorithm with the EAPD framework, the integration of extended Gmapping functionality and the refinement of the polygonal map. The experiments were based on a common setup in both the software parameters and hardware settings, apart for the parameters specific for individual functionalities.

The EAPD parameters include parameters for map dexterity and visualization, snd parameters for robot's navigation, and several others. The main parameters are the `robot_radius` influencing the navigation, the `poly_epsilon` parameter setting the detail in which the map will be kept and the `map_laser_range` which marks any measurements further than its distance as frontiers. These parameters were set to values 0.2, 0.05 and 5.0 respectively (the values are in meters). The `params_set.yaml` file used for the setting of parameters is included on the CD provided with this thesis.

Indoor offices

The BLOX building in Prague contains a typical office environment with long corridors and several resting areas. The university laboratories were moved during the course of the experiments, therefore the BLOX building was used only in the initial part of the experiments, using the office environment to verify the support of the Gmapping SLAM algorithm for the exploration with the EAPD framework.

Figure 5.3: The indoor environment in the BLOX building.

The uniform corridors in the building offer limited diversity for laser measurements, presenting a challenge for the SLAM algorithm. The algorithm also has to face the difficulty of a dynamic environment with a lot of people moving around. The environment can be seen in Fig. 5.3

CIIRC Laboratiories

Laboratories in the CIIRC building in Prague offer a diverse environment. There are multiple smaller rooms adjacent to narrow corridors and even a large open space laboratory available for the experiments.The open space laboratory can be seen in Fig. 5.4. The CIIRC laboratories were used in the experiments verifying all the implemented functionalities.

Figure 5.4: An open space laboratory in the CIIRC building.

5.3 Results

5.3.1 The EAPD framework supported by the Gmapping SLAM algorithm

Section 3.3 covered the integration of the Gmapping SLAM algorithm in the EAPD framework, where Gmapping is used to provide pose estimates. The following experiments were made to confirm the correct functionality of the system.

The first part of the experiments was executed in indoor offices of the BLOX building. The robot starts its movement in one of the office rooms and explores through narrow corridors and adjacent resting areas. Fig. 5.5 shows a grid map created by the Gmapping SLAM algorithm and the final polygonal map created by the EAPD package, a clear correspondence with the map from Gmapping can be seen.

The robot had to deal with a lot of passers-by in the BLOX building. The exploration algorithm had no problem with such environment and both the map and exploration algorithm were able to recover within a short time after encountering a temporary obstacle in the environment.

The second part of the experiments was made in a large open space in the CIIRC laboratories, which should provide a different challenge than the narrow corridors from the BLOX building. The robot had to face the difficulty of navigation through an open environment. Situations, where most obstacles are out of the range of the laser sensor, are challenging for the laser-based

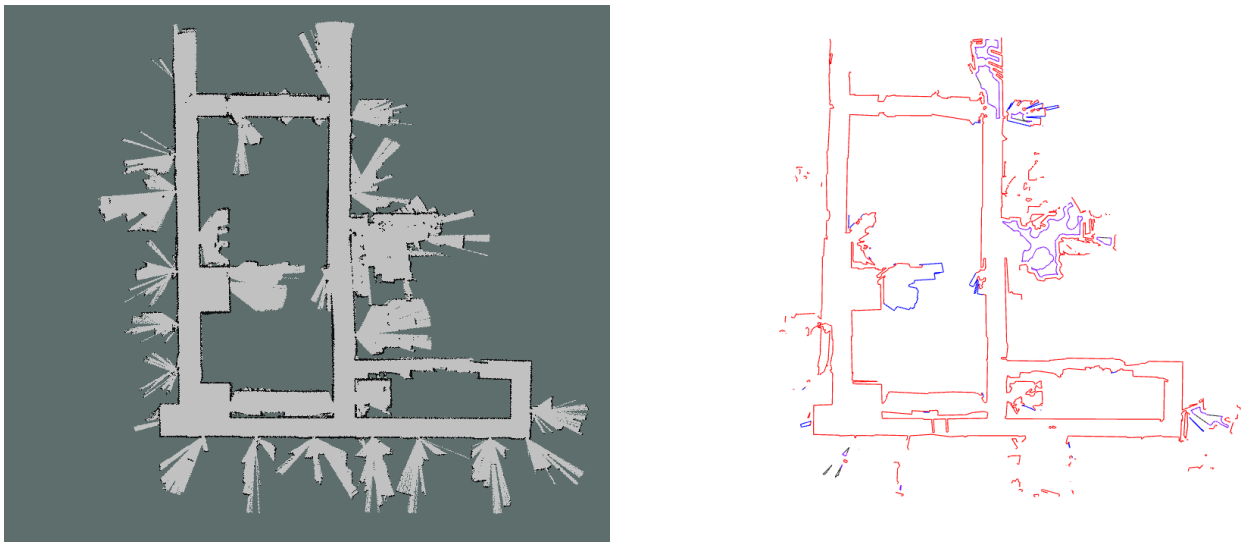


Figure 5.5: A map of the BLOX building created by the exploration algorithm. The Gmapping occupancy grid map is on the left and the polygonal map created by EAPD is on the right.

SLAM algorithm and the robot has an increased need for a precise odometry information. Fig. 5.6 presents the created maps from Gmapping and EAPD.

The functionality of the system is apparent, as the exploration was fully autonomous and the robot was able to create a model of the environment in both buildings.

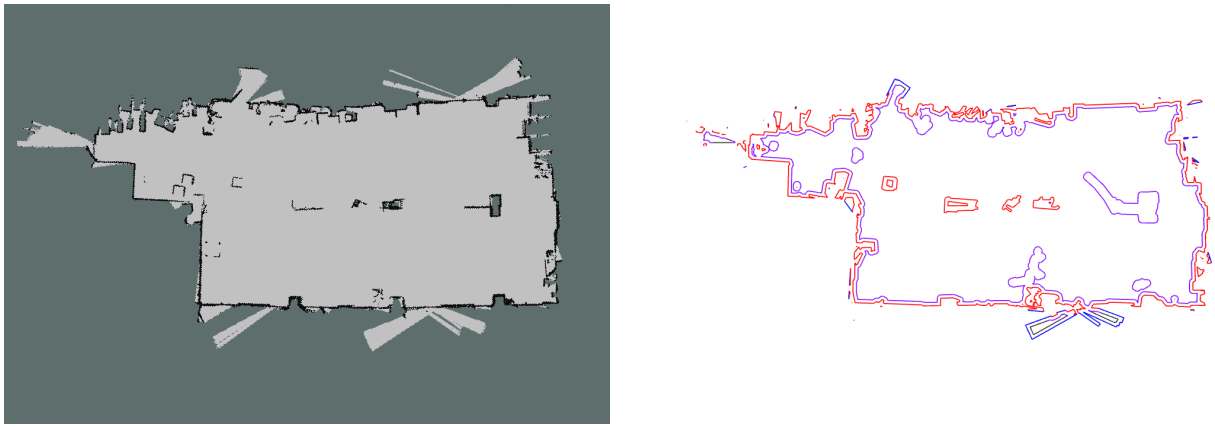


Figure 5.6: A map of the CIIRC laboratory created by the exploration algorithm. The Gmapping occupancy grid map is on the left and the polygonal map created by EAPD is on the right.

5.3.2 The EAPD framework with the use of the extended Gmapping functionality

Section 4.2 presents the integration of the extended Gmapping functionality into the EAPD package. The Gmapping package was modified to publish specific information on all particles allowing the EAPD package to synchronize with the SLAM algorithm. This synchronization allows the adaptation to the changes in the *best* particle described at page 5. Experiments testing this functionality are presented here.

Although the SLAM algorithm proved to be functional during the previous experiments, its connection with the exploration package was limited only to the estimates of the current position. This resulted in the inability of the exploration algorithm to properly react to the changes made by the Gmapping SLAM during a switch between particles. Fig. 5.7 presents a situation shortly before and after the switch in the *best* particle. The image on the left presents the state prior to the particle switch. The image on the right shows an incoherency in maps produced by the algorithms, caused by a change in the *best* particle.

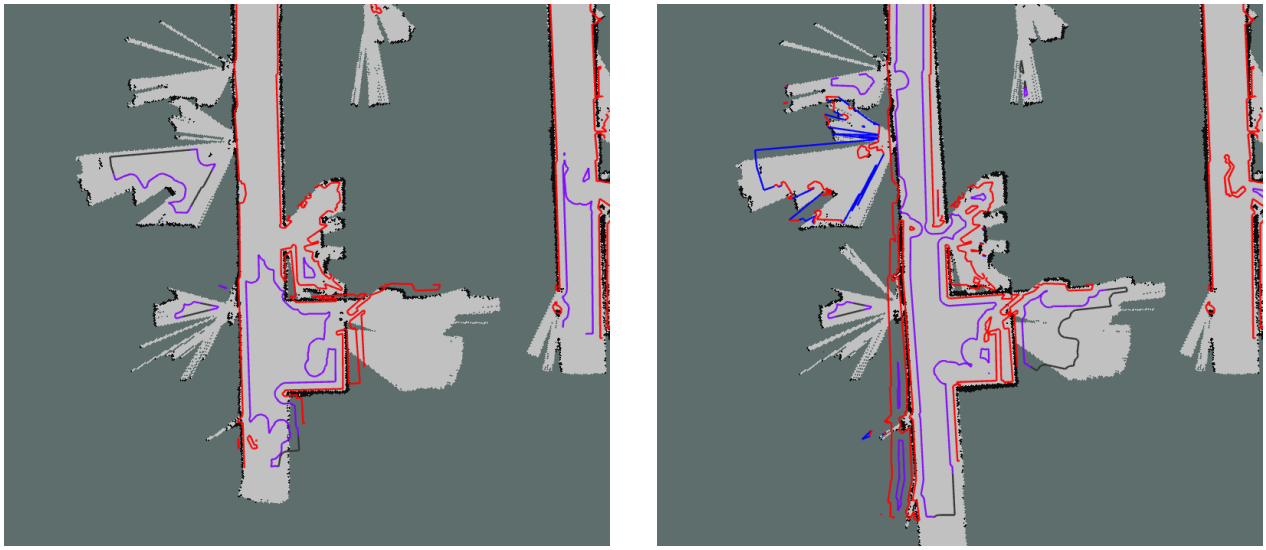


Figure 5.7: An overlay of the polygonal map and the grid map, before(left) and after(right) a switch in the *best* particle in the Gmapping SLAM algorithm. The situation occurred during the experiments testing the slam functionality, connected with the creation of the map in Fig. 5.5.

The Gmapping algorithm processes all particles during the process, it has already assimilated a full path of the particle leading to the current position and only switched between the particles and their map representations. However the EAPD algorithm has been following a different particle up to the point where the switch occured and received only the resulting change in position at the time of the switch, therefore the map created prior to the switch occurrence follows a different path than the path of the current *best* particle.

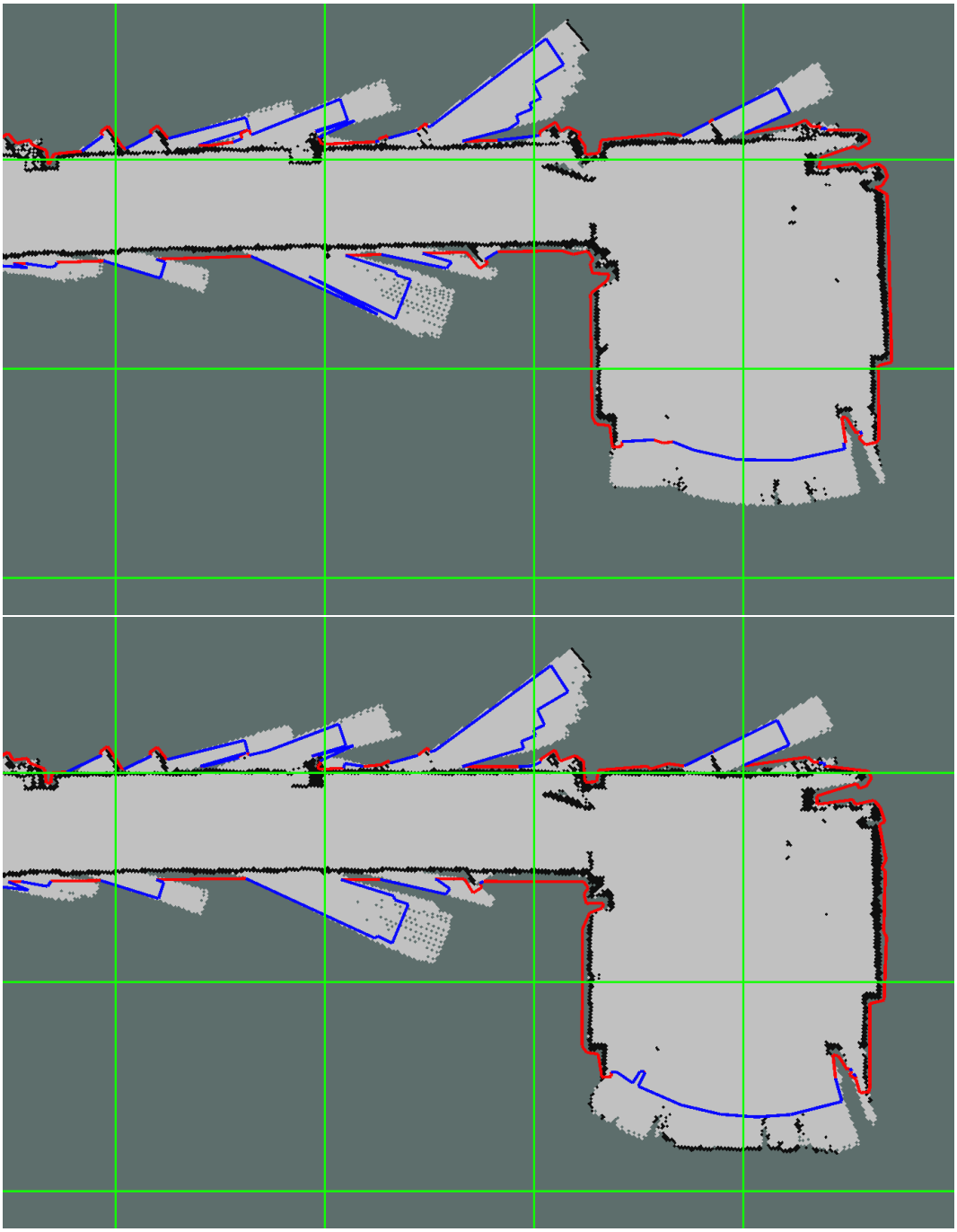


Figure 5.8: An overlay of the polygonal map and the grid map, before(top) and after(bottom) a switch in the *best* particle in the Gmapping SLAM algorithm. A green grid has been added to emphasize the shift in the structure of the maps.

The modifications made in Section 4.2 were initially proposed to integrate Gmapping loop closure functionality, but since the functionality in Gmapping projects itself in the selection of the *best* particle, the approach had to be changed and the EAPD framework had to be extended to support switches between the particles. Experiments testing the new functionality were performed in CIIRC laboratories and are focused on the proper functionality during a particle switch.

Fig. 5.8 presents the situation shortly before and after the switch in the *best* particle. As can be seen in the figure, the structure of both maps shifted in reaction to the switch. The maps are fully aligned after the shift, which proves the correct functionality of the implementation.

Although this new approach prevents discontinuities in the polygonal map and supports the loop closure functionality from Gmapping, the new implementation has two downsides to it:

- The first one is the limitation on the visualization functionality for the map. Until now the map was visualized incrementally, however with the new changes the visualization is switching between complete map representations. For a correct visualization of the map under the new functionality, the `clean_map` map parameter in EAPD had to be set as *true*. The setting doesn't have any influence on the exploration algorithm itself but it influences the visualization of the map, forcing it to disregard more recent measurements of a previously integrated area.
- The second downside is the increased demand on the CPU usage. The *map* node has to process dozens of maps during one iteration instead of the one map in the original functionality. The number of maps depends on the number of particles, our approach used 30 particles - the default setting of the Gmapping algorithm. Although the exploration process was still fluent, the CPU usage increased to 5 times of the original and the memory requirements tripled. The *Process monitor* plugin in the RQT development tool was used to monitor the CPU and memory usage.

5.3.3 Map refinement with a separate loop node

The integration of the map refinement into the EAPD algorithm is covered in Section 4.3. A separate `loop` node derived from the `map` node was written to provide the functionality to refine a section of a map according to specifications received from an outside source. The node was initially intended for situations regarding a loop closure, but due to the character of the final implementation the functionality is suitable for a general map refinement and the name was therefore changed to the *refinement* node in the final version of the framework.

The node as it is doesn't affect the overall functionality of the EAPD package and is provided with the package solely for the purpose of possible future uses in situations where a refinement of the polygonal map is desired (i.e. the aforementioned loop closure). To confirm the correct functionality of the refinement, the original `map` node was substituted with the `loop` node and a series of experiments was executed with a real robot in this setup.

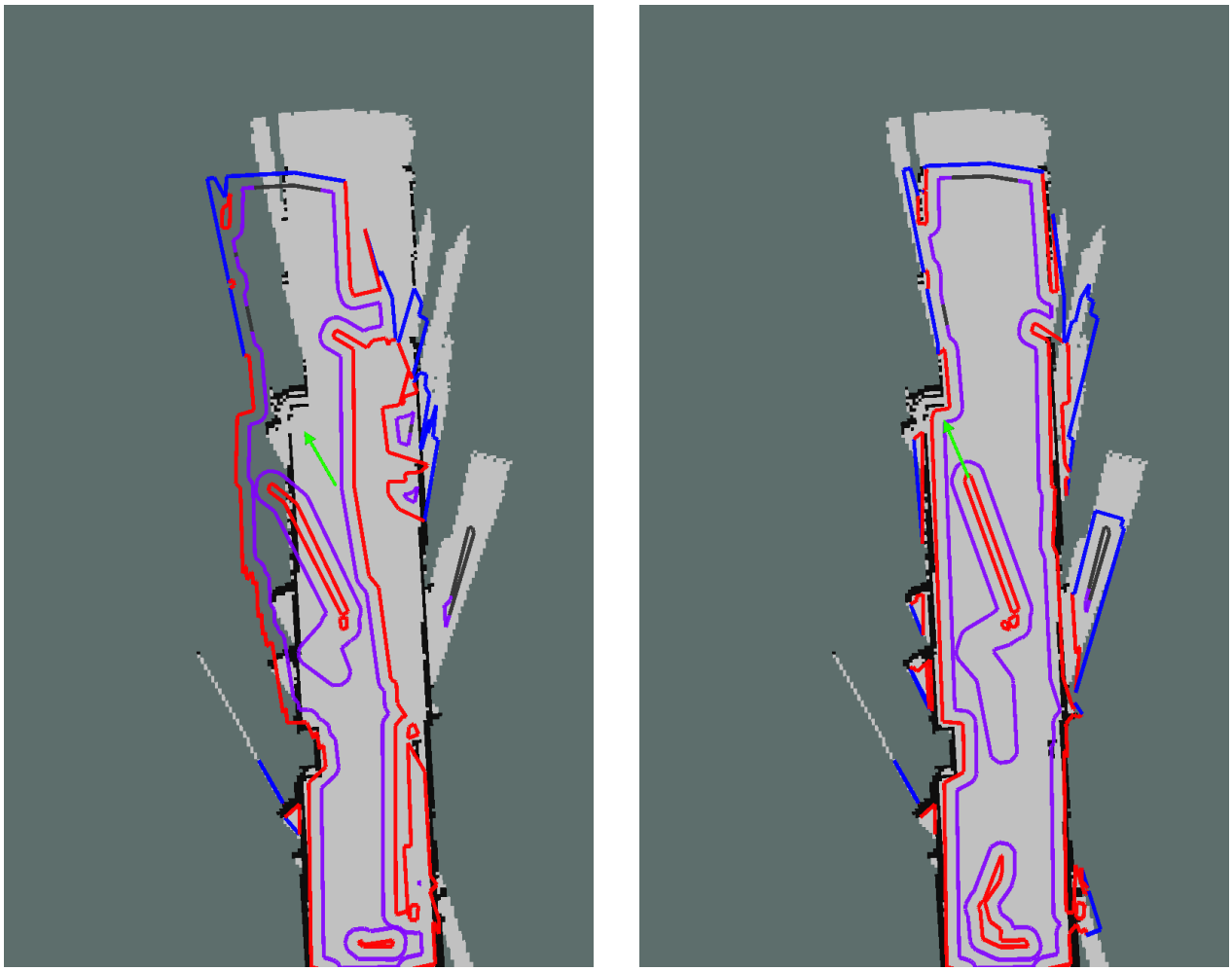


Figure 5.9: An overlay of the polygonal map and the grid map, before(left) and after(right) the refinement of the polygonal map.

As we currently don't have any working slam algorithm that would be capable of producing a sufficient refinement information, a separate `closure_detection` node was written to simulate both the deviation in robot's position and the *map refinement* message containing the information needed for the `loop` node.

The synchronization of the `closure_detection` node and the `loop` node through the *map refinement* message is one of the critical parts in the solution and has to be customized when used. In our example we use a simple synchronization through a *counter* variable determining the number of the laser messages received and their matching positions.

We use the `closure_detection` node to provide *fake* position messages to the `loop` node. The *fake* positions are created by shifting the correct positions received from Gmapping in one of the coordinates, in our case it is the x coordinate. At a specific time, the position starts to shift and after a hundred iterations a *refinement* message is send with the accumulated error in the position. The message carries the start and the end point of the deviation in the form of the appropriate *counter* values, and the *correction* value.

The results of the map refinement can be seen in Fig. 5.9 presenting the polygonal map created with the use of deviated positions and the same map after the refinement. After the refinement, the polygonal map aligned with the original map created by the Gmapping package.

The refinement processed only the last 100 positions. The refinement was processed in real-time and the additional CPU usage didn't have a specific effect on the exploration process. The refinement of 100 positions and laser scans was processed in approximately 1 second, there was a slight variation in the time with every executed experiment. The peak in the CPU usage lasted less then a second, and regretfully the current monitoring packages available in ROS were unable to provide more specific statistics for such a short time. Even though the refinement was processed in real-time, we can assume that the usage with a significantly larger segment of the map could have a higher demand on the computational complexity. In such situations the node can easily be modified to work offline with recorded data.

5.4 Discussion of the results

A mobile robot suitable for exploration was set up with a laser rangefinder and a control unit. The functionality of implemented algorithms was successfully tested during the experiments, individual comments to each functionality are listed bellow.

1. **The EAPD framework supported by the Gmapping SLAM algorithm**: The integration of the SLAM algorithm with EAPD was successful. The robot properly explored its surroundings and created a polygonal map representing the environment.

   However due to the nature of the particle filter based SLAM algorithm the final map was affected by the SLAM algorithm switching between individual particles. As can be seen in Fig. 5.7 the situation caused a discontinuity in the polygonal map.

2. **The EAPD framework with the use of extended Gmapping functionality**: The integration of information on particles from Gmapping into the EAPD framework successfully solved both the problem caused by a particle switch, and the integration of Gmapping's loop closure detection and map refinement functionality.

   However the computational requirements of the new version of the *map* node were several times higher than that of the original solution, especially during a particle switch occurrence. The rise is caused by the increase in the number of processed maps.

3. **Map refinement with a separate loop node**: The refinement of the polygonal map was implemented in a separate node, but it can be freely used in a place of the original *map* node. A crucial part for the refinement process is the synchronization of the refinement algorithm and the algorithm responsible for providing the request for the refinement.

   The refinement can processed reasonably large segments of map in real-time and can be modified to provide offline refinement if needed.

Chapter 6

Conclusion

The goal of this thesis was to set up a hardware solution with a real mobile robot equipped with a laser rangefinder for the use with the EAPD exploration framework, integrate a SLAM library into EAPD and design and implement loop closure and map refinement methods for a polygonal map generated by the EAPD framework. The implemented functionalities were tested in various environments with a mobile robot.

A hardware setup was implemented using the Kobuki robot base with the TurtleBot robotic package. The robot is controlled by the Intel NUC mini computer and uses the SICK LMS 111-10100 laser rangefinder to navigate through the environment. The control unit and laser were mounted to the robot base and connected to an external battery allowing for over 60 minutes of usage.

The Gmapping SLAM library was selected as a reliable provider of position estimates and integrated into the EAPD framework, allowing the exploration with the use of a laser rangefinder. The functionality was tested with the Kobuki robot in several different environments and the exploration framework supported by the SLAM library proved to achieve decent results.

Two approaches were chosen for the implementation of the loop closure and map refinement functionality. The first method is connected to the particle filter based SLAM algorithm in the Gmapping library. The second method was designed as a general map refinement, which could be used in a wider scope of applications.

A method to synchronize the particle filter functionality with the EAPD framework was proposed and implemented. The newly proposed method should be suitable for the use with most particle filter based SLAM algorithms. The method can remove negative impact of jumps in the estimated position caused by a switch in the *best* particle from the SLAM algorithm, improving the precision of the created polygonal map. The method also supports the innate loop closure functionality of the Gmapping library. The downside is that the new implementation has higher computational requirements than the original mapping process.

A method for a general map refinement was proposed and implemented. Since the approach to map refinement is different than for the first method, a separate ROS node was implemented

for the second approach. The new node is derived from the original *map* node responsible for the creation of a polygonal map in the EAPD framework, and can be freely used in place of the original *map* node.

The functionality of the refinement was successfully tested with a real robot. The refinement can be executed in real-time during the exploration on moderately large segments of the map. The refinement can be synchronized with an outside loop closure detection algorithm or used in other situations where a map refinement is required, however the synchronization in such cases is a crucial part of the implementation and has to be handled with extra care.

So far, the EAPD framework was only tested with external SLAM libraries. This work uses the Gmapping SLAM library and T.Juchelka[1] used the EAPD framework together with the RTAB-Map library. This part of the solution offers an opportunity for improvement, since the EAPD framework creates its own polygonal map of the environment. An implementation of a SLAM algorithm using the polygonal map as its base could simplify the solution, as there would be no need to create an additional map solely for the SLAM algorithm.

Bibliography

[1] T. Juchelka, *Exploration algorithms in a polygonal domain*. CTU in Prague, FEL, Dept. of Cybernetics, 2012.

[2] The official website of the Robot Operating System(ROS). http://www.ros.org

[3] V. Lhotský, *An Integrated Approach to Multi-Robot Exploration of an Unknown Space*. CTU in Prague, FEL, Dept. of Cybernetics, 2016.

[4] The official website of the OpenSLAM project. https://www.openslam.org/

[5] Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard, *Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling*. In Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2005.

[6] Wikimedia Commons, the free media repository. https://commons.wikimedia.org/wiki/File:Particle2dmotion.svg

[7] Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard, *Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters*. IEEE Transactions on Robotics, Volume 23, pages 34-46, 2007.

[8] Matías Nitsche, Pablo de Cristóforis, Miroslav Kulich and Karel Košnar, *Hybrid Mapping for Autonomous Mobile Robot Exploration*. Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011.

[9] J. W. Durham and F. Bullo, *Smooth Nearness-Diagram navigation*. Intelligent Robots and Systems, 2008.

[10] A. Johnson, *Clipper - an open source freeware library for clipping and offsetting lines and polygons*. Available from http://www.angusj.com/delphi/clipper.php.

[11] Jens-Steffen Gutmann and Kurt Konolige, *Incremental Mapping of Large Cyclic Environments*. Computational Intelligence in Robotics and Automation, 1999

[12] Paul Newman and Kin Ho, *SLAM- Loop Closing with Visually Salient Features*. ICRA, 2005

[13] Dirk Hähnel, Wolfram Burgard , DIeter Fox and Sebastian Thrun, *An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements*. IEEE International Conference on Intelligent Robots and Systems, (Vol. 1, pp. 206-211), 2003.

[14] Brian Williams, Mark Cummins, José Neira, Paul Newman, Ian Reid and Juan Tardós, *A comparison of loop closing techniques in monocular SLAM*. Robotics and Autonomous Systems, 2009.

[15] Richard Vaughan, *Massively Multiple Robot Simulations in Stage*. Swarm Intelligence 2(2-4):189-208, 2008.

[16] The official website of the Sick company.
https://www.sick.com

List of Figures

List of Tables

Appendix

CD Content

In table 6.1 are listed names of all root directories on CD

Directory name	Description
eapd2	The newest version of the EAPD framework. The versions in other folders are used during the activities connected with their folder location.
Experiment setup	Code sources of the framework, configuration files and launch files used to test individual functionalities.
thesis.pdf	The pdf file containing this thesis.
Thesis	Source files for this thesis.

Table 6.1: CD Content