TECHNISCHE UNIVERSITÄT WIEN

Institut für Computertechnik
Institute of Computer Technology

Master's Thesis

submitted by

Milanovic Sinisa

Registration Number 01429143

Evaluation of SLAM methods and Adaptive Monte Carlo Localization

In partial fulfillment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Vienna, Austria, 2.April 2022

Study code:	066 504
Field of study:	Embedded Systems
Supervisor:	Univ.Prof. Dipl.-Ing. Dr.techn. Hermann Kaindl
Co-Supervisor:	Univ.Ass. Dipl.-Ing. Dr.techn. Hoch Ralph

Acknowledgements

First of all, I want to thank my entire family, who supported me throughout my studies in all matters and gave me the motivation as well as the opportunity to achieve my goal. Especially in very stressful times when I certainly did not make life easy for them, they always stood by me and showed understanding of my situation.
Furthermore, I would like to thank my supervisor Prof. Hermann Kaindl, and co-supervisor Dr. Ralph Hoch, who advised me well during the execution, but especially during the preparation and documentation of this thesis.
Last but not least, I would like to thank Prof. Markus Vincze, who supported me with his expertise and advice during the finalization of this work.

Kurzfassug

Mobile Roboter, die autonom Aufgaben erfüllen, müssen in der Lage sein, zwei grundlegende Probleme selbständig zu lösen, um eine Autonomie zu gewährleisten. Einerseits muss eine Karte einer unbekannten Umgebung erstellt und gleichzeitig die Position des Roboters bestimmt werden, wofür der Lösungsansatz Simultaneous Localization and Mapping (SLAM) verwendet wird. Andererseits muss ein autonomer Lokalisierungsalgorithmus, wie die Adaptive Monte Carlo Localization (AMCL), verwendet werden, um das Kidnapped-Robot Problem zu lösen.

Jeder Sensor hat eine gewisse Messungenauigkeit, die im Falle eines Laser-Distanz-Sensors (LDS) sehr gering ausfällt und keine signifikanten Auswirkungen auf die Genauigkeit der erzeugten Karte hat, aber diese Ungenauigkeit addiert sich mit jeder Messung, bis ein Versatz in der Karte entsteht und sie unbrauchbar macht. Daher müssen SLAM-Algorithmen in der Lage sein, sowohl eine detaillierte Karte zu erstellen, als auch ein breites Spektrum von LDS mit unterschiedlichen Messungenauigkeiten abzudecken.

Der Lokalisierungsalgorithmus AMCL hingegen zeigt starke Einschränkungen bei der Lokalisierung nach dem Auftreten des Kidnapped-Robot-Problems, wenn zum einen die Position des Roboters stark verändert wurde, oder wenn mehrere Partikel bereits einen Schwarm gebildet haben und eine mögliche Position des Roboters darstellen. Aus diesem Grund wird AMCL oft nur als Basis für weiterentwickelte Lokalisierungsalgorithmen verwendet.

In dieser Arbeit wurde evaluiert, wie robust und zuverlässig die drei SLAM-Methoden Gmapping, Hector und Karto bei Verwendung von LDS mit unterschiedlichen technischen Eigenschaften arbeiten. Darüber hinaus wurde eine optimierte Parametrisierung von AMCL durchgeführt und an die Umgebung angepasst, um eine autonome Lokalisierung nach dem Auftreten des Kidnapped-Robot-Problems zu gewährleisten, ohne dass ein externes Eingreifen erforderlich ist.

Diese Arbeit wurde in einer Simulation im Robot Operation System (ROS) durchgeführt, in der eine Wohnung als simulierte Welt erstellt wurde. Von dieser virtuellen Umgebung wurden Karten mit verschiedenen technischen Charakteristiken des LDS erstellt, die mit Hilfe eines für diese Arbeit entwickelten Programms mit dem Grundriss der Wohnung verglichen wurden. Auf diese Weise konnte festgestellt werden, wie sich die unterschiedlichen Ungenauigkeiten des LDS auf die Genauigkeit der Karte auswirkten.

Für die Bewertung von AMCL mit den angepassten Parametern wurden verschiedene Szenarien durchgeführt, welche das Kidnapped-Robot Problem mit verschiedenen Schwierigkeitsstufen simulieren. Um die Effizienz bei kleinen Positionsänderungen zu erhalten, wurde der Referenzierungsprozess in drei Phasen unterteilt.

Die Resultate dieser Arbeit zeigen, dass sowohl Gmapping als auch Karto sehr robuste und zuverlässige SLAM-Methoden sind, die selbst bei hohen Ungenauigkeiten des LDS brauchbare Karten erzeugen, während Hector schon bei kleinen Abweichungen Probleme hat, die Position des Roboters zu bestimmen, was zu einer Überlappung in der Karte führt. Darüber hinaus wurde anhand verschiedener Kidnapped-Robot Szenarien gezeigt, dass AMCL immer in der Lage ist, eine Lokalisierung durchzuführen, sofern die Parameter an die Umgebung angepasst werden.

Abstract

Mobile robots that perform tasks autonomously must be able to solve two fundamental problems on their own to ensure autonomy. On the one hand, a map of an unknown environment must be created and the robot's position determined simultaneously, for which the Simultaneous Localization and Mapping (SLAM) solution is used. On the other hand, an autonomous localization algorithm, such as Adaptive Monte Carlo Localization (AMCL), must be used to solve the kidnapped-robot problem.

Every sensor has some measurement inaccuracy, which in the case of a laser distance sensor (LDS) is very small and has no significant effect on the accuracy of the generated map, but this inaccuracy adds up with each measurement until there is an offset in the map and makes it unusable. Therefore, SLAM algorithms must be able to produce a detailed map and cover a wide range of LDS with various measurement inaccuracies as well.

The localization algorithm AMCL, in contrast, shows major limitations in localization after the occurrence of the kidnapped-robot problem, if the position of the robot has been changed significantly, or several particles have already formed a swarm and represent a possible position of the robot. For this reason, AMCL is often used only as a base for improved localization algorithms.

In this work, we evaluated how robustly and reliably the three SLAM methods Gmapping, Hector, and Karto perform when using LDS with different technical characteristics. In addition, an optimized parameterization of AMCL was performed and adapted to the environment to ensure autonomous localization after the occurrence of the kidnapped-robot problem, without the need for external intervention.

This work was carried out in a simulation in the Robot Operation System (ROS), where an apartment was created as a simulated world. Maps of this virtual environment were created with different technical characteristics of the LDS, which were compared to the floor plan of the apartment using a program developed for this work. This allowed us to determine how the different inaccuracies of the LDS affected the accuracy of the map.

For the evaluation of AMCL with the adjusted parameters, different scenarios simulating the Kidnapped-Robot problem with different difficulty levels were performed. In order not to reduce the efficiency of this localization in the presence of small position shifts, the referencing process was divided into three phases, which allowed for fast localization even in the presence of small position changes.

The results of this work show that both Gmapping and Karto are very robust and reliable SLAM methods that produce usable maps even in the presence of high inaccuracies in the LDS, while Hector has problems determining the position of the robot even in the presence of small deviations, resulting in an overlap in the map. Furthermore, it has been shown using various kidnapped-robot scenarios that AMCL is always able to perform localization, provided that the parameters are adapted to the environment.

Contents

List of Figures

Chapter 1

Introduction

In recent years, robotics has made great progress in the research field of autonomous mobile robots. Initially, they were mainly used for outdoor applications, which primarily have access to external sensor technology, such as localization via GPS. This makes it possible, for example, for robotic lawnmowers to locate themselves in the garden at any time.
In the meantime, the range of tasks of commercial robots has also expanded to indoor tasks, which means that even complex applications, such as vacuum cleaning, can be executed autonomously. One problem with indoor applications is that localization via GPS is not possible. As a result, these mobile robots need to be extended with additional sensors to perform localization autonomously.
Therefore, the simplest mobile robots were equipped with gyroscope navigation, which can detect any change in orientation. Together with the traveled path, which is calculated by the rotation of the wheels, and an ultrasonic sensor, which detects the obstacles in front, a map could be created. This technology is comparatively cheap, but it provides very inaccurate maps of the environment and has little ability to react to errors due to the low level of sensors. If the robot is moving on a carpet, the wheels may slip, which results in incorrect measurements of the traveled distance.
Currently, the most used technology is based on a laser distance sensor (LDS), which is mounted on the top of the robot as a kind of tower and rotates continuously 360°. This enables up to 720 measurement points per revolution and together with a standard deviation of up to 0.06% is a significantly more precise measurement around the entire robot possible. Another advantage is also that the traveled distance no longer has to be calculated only based on the rotation of the wheels since the distance to an object can also be taken as an indicator.
Another approach is to use cameras with mobile robots. The advantage is that an image provides much more information than an LDS since this is mounted at a fixed height and a captured image, on the other hand, contains information from all obstacles and structures from the ground up to the ceiling. An LDS is also more expensive than a simple camera, but the reason why this technology is not yet widely used in commercial robots, especially in indoor applications, is the complexity of processing these images. For example, to extract the distance to an obstacle from an image, several images have to be taken from different perspectives.
A commonly asked question, why such a detailed map of the environment is even necessary for applications, can be answered with a good example of the first generation of vacuum cleaning robots, which cleaned the room by a random principle. They always moved forward until they collided with an obstacle then they rotated around a specific angle and started again to move straight ahead. In most cases, the robot was able to cover the entire space, but in a very inefficient

way. However, if the robot has an accurate map of the space before starting its task, an algorithm can calculate the shortest path, which not only saves a lot of time but can also guarantee a better result.

1.1 Problem Description

The development of autonomous robots for indoor applications led to several challenges that had to be solved by a combination of sensor technology and specific algorithms. This work tackles two fundamental problems that are essential for most mobile robots for indoor and outdoor applications.

Every sensor has some measurement inaccuracy, even if it is very small, as, in the case of the LDS, this creates serious mapping problems. If a robot scans an area of the environment with an LDS, all objects, like a wall, in the range of the sensor will be added to the map with minimal deviations, which are so small that they can be neglected. If the robot moves further through the room and performs additional measurements within short distances, further sections with measurement inaccuracies will be added to the map. These measurement uncertainties will be added up until the map has such a large shift that it is no longer useful.

To solve this problem, research has been done for many years on the Simultaneous Localization and Mapping (SLAM) solution, which enables mobile robots to create simultaneously a map while localizing their position. There are already many different SLAM approaches, which use different algorithms to handle the measurement inaccuracies. In addition, there are a large number of manufacturers that have developed LDS with different technical characteristics. Thus, SLAM algorithms must not only be able to generate a detailed map for the applications but also cover a wide range of LDS with different measurement accuracies.

Another approach is mobile robots that use Visual SLAM (VSLAM) methods to create a map and localize themselves by using a camera instead of the LDS. Through the captured images of the environment, it is possible to determine the depth of objects and recognize structures. Besides the used algorithms, the resolution of the camera is essential to solve these tasks efficiently and to create a precise map. A low resolution of the images, therefore, leads to inaccuracies in the calculated distances and in the recognition of objects, which is also required for an optimization of the created map.

Another well-known problem with autonomous mobile robots is the kidnapped-robot problem, where the robot is moved to an arbitrary location while they are in operation. This causes the robot to lose its position and therefore the task cannot be continued. To solve this problem autonomously, a self-localization must be performed, trying to find the current position in the existing map with measurements of the environment. A popular algorithm for solving the kidnapped-robot problem is Adaptive Monte Carlo Localization (AMCL), which dynamically distributes particles based on measurements of the environment, where each particle represents a possible position and orientation of the robot. As described in [5], autonomous localization with AMCL fails when the robot is abducted while the convergence process has already started. The algorithm uses rather the center of the approximated particle swarm as the best pose estimation instead of comparing the results of the measurements with the entire map. Additionally, a large distance between the true position of the robot and the particle swarm often leads to localization bias. Because of this problem, AMCL is more often used only as a base for an autonomous localization algorithm, such as in [4], or only in combination with position and orientation estimates by an operator, which results in a loss of autonomy.

1.2 Proposed Solution

The goal of this work was to evaluate how robustly and reliably different SLAM and VSLAM methods work when using LDS and cameras with different technical characteristics, and to adapt AMCL to the environment in such a way that autonomous localization of the robot after the kidnapped-robot problem occurs is always successfully performed without operator intervention. Additionally, the referencing should be divided into stages, so that a simple referencing is attempted first before a more complex process is started. This should ensure the efficiency of AMCL on the one hand and guarantee successful localization on the other.

Since neither a mobile robot nor LDS and cameras with different characteristics were available, this thesis was realized within a simulation. To determine which method can best handle the effects of measurement inaccuracies, algorithms were chosen that use different approaches. For LDS based SLAM, the probabilistic approach Gmapping, which uses a particle filter, and the two optimization-based approaches Hector and Karto, were chosen, while for VSLAM, the methods ORB-SLAM and Mono-SLAM were selected because they require only a single camera for mapping and localization. Many manufacturers offer LDS for standard applications up to high-performance measurements, which differ significantly in standard deviation, update rate, and samples/revolution. To be able to simulate the full spectrum of different LDS, a standard deviation of 0.06% - 5%, an update rate of 1Hz - 500Hz, and samples/revolution of 180 - 720 were used for the evaluation.

For the VSLAM approaches, a classic webcam, as well as a high-resolution camera, can be used, as long as they provide the required technology for the implemented method. In the scope of this work, resolutions from commercial webcams up to the newest models of Raspberry Pi cameras were used as a reference in the simulation. To detect a significant difference in mapping with the VSLAM approaches, resolutions from 0.3 MP up to 12MP were selected for the evaluation. With these parameters, a separate 2D map should be created with each of the SLAM and VSLAM methods. To be able to use a typical environment for indoor applications for the simulation, my apartment was created as a virtual world with the 3D robot simulator software Gazebo. For the evaluation, it was necessary to develop a program, which can determine how the different parameter sets of the LDS and the camera affected the created map. This program requires an ideal floor plan of the virtual apartment, which is used as a reference map. Both the SLAM-generated maps and the reference map are visualized in an Occupancy Grid, which divides the environment into free, occupied, and unexplored areas. Portable Gray Map (PGM) was chosen as the image format for the maps, as it represents each pixel of the Occupancy Grid Map line by line as a grayscale value between 0 - 255. This offers the great advantage that the three different areas of the environment are represented as 0 for occupied, 254 for free, and 200 for unexplored. This allows the program to compare the free pixels between the reference map and all SLAM-generated maps line by line and to calculate the deviation of the scanned area in percentage.

For the evaluation of the reliability of AMCL, different kidnapping robot scenarios had to be specified, which are then executed in the simulation. The kidnapping problem refers not only to cases where the robot is lifted and repositioned but also to slight shifts in position and orientation due to collisions with moving objects. Therefore, scenarios were specified that include small to medium position shifts of 1m, 3m, and 5m, as well as orientation shifts of 45° and 90°. Since this problem involves autonomous localization in a known environment, a detailed map of the virtual world had to be created, which was done with the Gmapping algorithm. The framework Rviz is a 3D visualizer for the sensor data and state information from the mobile robot and was used to visualize the process and the results of the localization. AMCL offers many parameters for fine-tuning, which were adjusted through various tests so that localization within the virtual world could always be performed successfully, even if several referencing procedures had to be done. The

referencing process was divided into 3 stages which were executed one after the other until a correct localization was achieved. The first stage was a 360° reference rotation, which already led to the resampling of the particles. If this did not lead to correct positioning, 3 more rotations were performed in the second stage. The last stage, which always leads the simulation to success, was to navigate the robot through the virtual world along the wall until a complete resampling was performed, which repositioned all particles within the apartment. This referencing process was evaluated along with the specified scenarios to determine the reliability of AMCL in primarily indoor applications. At the end of the evaluation, the kidnapped-robot problem was analyzed on a real robot vacuum cleaner to get an idea of the respective performance of an autonomous localization method developed by a manufacturer.

1.3 Structure of the Work

The remainder of this thesis is organized in the following manner. Sections 2 and 3 cover the basic understanding of different SLAM and VSLAM approaches, as well as the current state-of-the-art of these technologies. Section 4 describes the setup of the simulation environment and the used hardware components. The setup and the specification of the scenarios for SLAM, VSLAM, and AMCL are defined in Section 5. Section 6 evaluates the results of this thesis and Section 7, while Section 7 analyzes the results of the localization from a robot vacuum cleaner. Section 8 describes the opportunities, applications, and constraints of Visual SLAM.

Chapter 2

Background

In this chapter is all the necessary background for this thesis described. First, the setup of SLAM is covered before introducing the different types of SLAM and the most common approaches. For VSLAM are also the required techniques and the popular methods explained. In the end, the MCL and the further developed AMCL localization methods are presented.

2.1 SLAM Setup

Simultaneous localization and mapping are used by many applications to build a map and locate the position of a mobile robot in an unknown environment.
This technology offers great advantages, especially in environments where the system cannot use any information from outside, like GPS data, to get the current position.
Nether the map or the current localization is at the beginning known and therefore it is necessary to execute both tasks simultaneously.
Each SLAM system consists of a front-end and a back-end.

- **Front-end:**
  The front-end takes raw sensor data and transforms it into a specific representation, such as constraints of nodes or probable distances to landmarks, depending on the used sensor and algorithm. It is also used for feature matching and subsequently for the loop closure.

- **Back-end:**
  The data of the front-end are used by the back-end as inputs for SLAM algorithms like an EKF, a particle filter, or a graph-based approach. To increase the performance of detecting a loop closure, it is often necessary to send the optimized poses back to the front-end, which requires them as an input.

The following components are necessary to execute the tasks of the front-end and back-end. Sensors and feature matching are based on the front-end, while the others are used for the back-end.

- **Sensors:**

  Each SLAM technology needs at least a range distance sensor to calculate the current position and to be able to create a map. Most algorithms are also extended by an odometry method to increase the accuracy of the pose estimation.

  The sensors are divided into distance measurement sensors to scan the environment, such as cameras and laser distance sensors, and those that capture information from the robot itself, like a gyroscope or an Inertial Measurement Unit (IMU).

  The odometry is used to estimate changes of the robot's position through the use of motion sensors. This pose estimation is usually done by locating over GPS, but SLAM algorithms are calculating the position by measuring the traveled distance through the wheel rotation.

- **Optimization:**

  Sensors and odometry always have a certain inaccuracy in the measurements. This leads to an additive drift in the pose estimation of the robot, which results in an unusable map. The variance is negligible for individual measurements, but the mobile robot moves on with this discrepancy and measures again, resulting in a summation failure.

  To avoid this behavior, it is necessary to use techniques such as EKF or particle filter.

- **Loop closure:**

  Loop closure detection is usually implemented in the SLAM algorithms after the optimization process and is supposed to give feedback to the front-end. The best example for explaining the loop closure is a room with one obstacle in the corner. The robot starts by measuring that obstacle and moves a specific distance clockwise through the room. Every time it stops, a new scan of the environment will be performed. When the robot finishes an entire round, he will detect the same obstacle again. The problem is that this measurement will be different from the first one because the uncertainties of sensors and odometry lead to a shift of the pose.

  The loop closure detection is then triggered and adds constraints between all adjacent positions and especially between the first and last positions. These two positions should be identical and therefore they will be pulled together until they are matched completely. The other poses will also be forced to move because the constraints between them are not flexible. This procedure will execute several times until the created map is accurate enough.

- **Feature matching:**

  Special structures such as corners and edges are searched for in the recorded images, which are then saved as key points together with the current robot position. This allows the robot to recognize the same objects, even if the picture was taken from a different view.

  A better feature matching increases the loop closure detection because the robot is not supposed to be at the same spot and he can detect much more loop closures at one position.

- **Map:**
  The map can be built as a set of sparse points representing unique or unusual regions of the environment, or also as a dense surface model. An easy and commonly used map is the Binary Occupancy Grid which is representing the environment with rectangles. If the rectangle is occupied by an object, like a wall, it is set to 1, otherwise to 0. This allows the mobile robot to navigate through the environment without collisions with any obstacle.

2.2 Types of SLAM

There exist various types of SLAM, but all of them are separated into two different types.

- Online SLAM:
  All SLAM algorithms are calculating and predicting a lot of positions, angles, and paths in the environment to find the most efficient way to create a map. This leads to a lot of unnecessary data which all have to be stored. Online SLAM is just interested in the current position and the last calculated estimations. All other data from the past are deleted and will therefore save a lot of memory space and computational effort, but this means that optimization through loop closing, for example, is no longer possible.

- Full SLAM:
  Full SLAM stores the whole trajectories and offers a lot of data for the calculations, which increases the complexity of these methods significantly, but it enables an optimization of the map with loop closing and feature matching. Therefore, Full SLAM is often required for critical applications, which need precise maps.

2.3 SLAM Approaches

The most popular Online SLAM methods are the particle filter and EKF. These algorithms are also mostly the base for many of today's current approaches. The fast evolution of computer technology enables powerful processors and larger memory, which opened up new possibilities for SLAM technology. The system was able to handle Full SLAM algorithms, such as Graph-based approaches.
The filtering algorithms were described in the work [23], while the graph-based method was documented in the Matlab libraries.

2.3.1 Particle Filter

The Particle Filter is mostly used for position and orientation estimation for nonlinear functions in a given map. This algorithm creates random particles within the entire map and each of these particles represents a probable pose, where the robot might be. Each particle is represented by position coordinates and by an angle, which indicates the estimated orientation.
The filter can be described by three steps that are performed continuously, but first, initialization of the particles has to be done based on the possible positions in the environment of the robot. After that, the prediction step must be performed, which estimates the subsequent position of the robot, depending on its current speed and orientation. This

means that the algorithm shifts all the particles from the initialization by the distance which will be reached in a specific time, based on the movement of the robot. The Particle filter adds to this new position a noise, which is called a proposal distribution and needs to be done because of the uncertainties of the sensors and the odometry. The next step is to perform a correction on the shifted and newly generated particles. This is done by assuming that the robot is at this shifted position and comparing its measurements at this point with the estimated measurements of all particles. The most probable positions get a higher weighting, while the very improbable ones get a lower weighting. The last step is the Resampling step, which is necessary because the numbers of particles are limited. After the correction step has evaluated all the particles, those with lower weights will be deleted, while new ones are generated for the more likely positions to increase the density. These three steps will be repeated until most of the particles are in the same position, which should represent the current position of the robot.
The Particle Filter can also be used in an unknown environment. The difference in the procedure is, that the mobile robot has to create at first an inaccurate map of the environment. The following localization procedure will then also optimize and extend the map.

2.3.2 Extended Kalman Filter

The Kalman filter (KF) is an estimation algorithm for linear Gaussian environments, which can not be used for robots in real environments because they are always non-linear. For enabling the usage of the KF also for SLAM applications in real environments, it was necessary to extend the approach for non-linear environments, leading to the Extended Kalman Filter.
EKF is calculating the position and orientation of the robot, the position of the landmarks, which are points of interest, and their corresponding uncertainties of them. These uncertainties are separated by the robot pose, the landmarks pose, and the correlation between them. This information is typically stored in a coherence matrix, which can lead to a lot of data in big environments. Therefore, the nearest neighbor approach is often used, to minimize, for example, the number of unknown landmarks.
EKF is separated into prediction and correction tasks. The first step is to initialize the entire coherence matrix of the robot with 0. Then the mobile robot moves to the next position and updates the current pose together with the uncertainties and the correlations to the landmarks. Next, the robot estimates the distance to the landmarks without updating the coherence matrix and secondly, it performs real measurements. EKF compares then this estimation with the real measurements of the distances and calculates the difference between them. When this is done successfully, the coherence matrix can be updated with the poses of the robot and the landmarks.
This algorithm was one of the first approaches for SLAM and is still used often in various methods, but it is not recommended for larger environments, because it has to calculate and store the information of a large number of landmarks, and their associations, which leads to the high computational effort.

2.3.3 Graph-based SLAM with pose graphs

The pose graph algorithm starts with saving the pose of the robot and the distance to landmarks in a pose graph. The range detection sensor and the odometry of the robot cannot offer full accurate precision. This leads to a small offset at every measurement. When the measurement is done, the mobile robot moves further to the next position, saves the

pose, and measures again the environment. The uncertainties of this measurement will be added to the last one. This means, that even if the tolerance of the sensor and the odometry is very small, it will be added incrementally with every new position until the shift leads to a useless map.
The Graph-based SLAM approach with pose estimation creates constraints between adjacent poses. These constraints can be imagined as a rubber band, which flexibility depends on the quality of the odometry. Inaccurate odometry will result in very flexible constraints in all directions, while a precise one will leads to a rigid connection.
If a loop closure gets detected, which indicates that the mobile robot is at a position it already was before, the algorithm adds an edge between these nodes. The uncertainties cause the shift between these nodes and this is going to be corrected by pulling them together because both of them are actually on the same spot. All other constraints and poses are shifting equally and correcting with every loop closure the map and the saved positions a little bit. This procedure has to be repeated until the map gives a correct picture of the environment, and the offset from the calculated and the real position of the robot is accurate enough.

2.3.4 Fast SLAM

Fast SLAM is an algorithm that combines the Extended Kalman Filter and the Particle Filter. The landmarks and their corresponding uncertainties are estimated with the EKF and the robot position is calculated with the particle filter. One problem of the particle filter is that if a wrong predicted position is taken as a probable correct pose, there will be an irreversible mistake generated in the map.
Fast SLAM solves this problem by creating for every particle also a different map. This means, that Fast SLAM creates incrementally several maps of the robot pose and gives, therefore, the opportunity to jump back to an older map, which was marked as not probable but was the correct solution in the end.
The procedure of Fast SLAM is:

- Initialization of the robot pose and saving the landmarks
- Mobile robot moves to the next position and calculates the estimated pose with the corresponding uncertainties.
- Spreads randomly particles within the calculated uncertainty area.
- Compares the estimated measurements of the landmarks of every particle with the saved ones and spreads more particles at higher probable particle positions.

2.3.5 Gmapping SLAM

Gmapping [11] [12] is a SLAM method, which is using the Rao-Blackwellized particle filer to create a map. This algorithm takes data from the LDS and the odometry and creates for every particle its own map. Furthermore, an EKF is used to estimate the pose of the robot, which leads to increased memory consumption and higher computational effort. However, Gmapping is providing an efficient algorithm to improve the estimated distribution of the particles, which reduces the number of particles and leads to an improvement in performance. Gmapping is performing well for localization and mapping and is one of the most popular SLAM methods.

2.3.6 Hector SLAM

Hector-SLAM [15] uses an LDS in combination with an Intertial Measurement Unit (IMU) to create a 2D map and does not require odometry data. Scan matching is necessary to compare the measurements of the LDS with already scanned objects on the map, which enables to perform a loop closure. This leads to an optimization of the map with the Gauss-Newton method. To determine the position and orientation of the robot, Hector-SLAM is also using an EKF.

2.3.7 Karto SLAM

Karto-SLAM [17] is a graph-based approach, as described in 2.3.3, which is representing every position and orientation with nodes, and the constraints between them as edges. For scan-matching and loop closure is the Sparse Pose Adjustment (SPA) [16] used, which is also required for the optimization of the nonlinear pose graphs. Graph-based SLAM approaches are highly efficient for bigger environments, and especially Karto-SLAM achieves good results because it handles only a pose graph.

2.4 Visual SLAM

Visual SLAM (VSLAM) uses one or more cameras instead of a LDS to calculate the distance and it is not dependent on odometry algorithms. This offers some great advantages and opportunities, but it also needs more complex programs to calculate and interpret the information correctly. The common problems of especially monocular VSLAM systems are:

- depth estimation error
- losing tracking
- scale drift

2.4.1 Necessary Techniques for VSLAM

To extract the required information of the captured images, some techniques are necessary.

Bundle Adjustment

Bundle Adjustment (BA) is a state estimation technique that estimates the positions of key features and cameras in the environment by using several images. The first step is to estimate the position of a key feature and the camera. These estimations are going to be projected into a camera image, which represents this key feature in pixel coordinates. The difference between the location of the projected point and the observation can be calculated by the number of pixels. BA leads to a very large amount of data and calculations because every image contains a lot of key features. Therefore it is important to use only these features, which are represented in nearly every image, and to remove all other, which occurs just a few times. With this smaller set of key features, it is now possible to use the least square method to minimize the error and increase the accuracy.

Bag of Words

BoW is used to describe images and find similarities in them. This technique allows matching the same key feature like a building, in different pictures and from different perspectives. It is also used to filter all images according to the same key features. This is very helpful if a specific structure has to be found within a huge amount of data. It is necessary to define all words, which are representing specific geometrical structures of interest, within a BOW vocabulary. Every image will be represented by these words and the amount of them in one picture. This reduces the needed memory significantly because the images themselves are not needed anymore.
BoW is used in SLAM for place recognition and to detect loop closures.

2.4.2 Methods of VSLAM

The following Visual SLAM methods are described in [26] and [22].

Feature-based methods

Feature-based methods extract and compare features from a 2D image and calculate the distance to them, by using the position of the camera. All these features are then added to a virtual 3D environment with the corresponding distance.

- **MonoSLAM:**
  MonoSLAM uses a single monocular camera and calculates with a probability model the needed 3D features. This is necessary because it is not possible to recover the depth from a single image.
  MonoSLAM estimates in every frame the location of the key features in the map and the camera pose. The 6-DoF camera position and the calculated 3D map location are updated simultaneously at every single frame.
  This method needs a lot of features from a lot of images and therefore tracking and mapping are often realized in different threads to increase the performance.

- **ORB-SLAM:**
  ORB-SLAM is also a MonoSLAM method because it can be used with just a single monocular camera, but it uses place recognition systems with ORB features and includes the BoW technology. ORB-SLAM is a rotation-invariant and scalable feature with a 256 bits descriptor. The place recognition algorithm is very efficient and offers a real-time implementation of reallocation and loop closing in SLAM systems.
  The ORB-SLAM algorithm is divided into 3 parts, which are all running simultaneously in 3 separate threads.

  - Tracking:
    Matches the features frame-by-frame and compares them with a local map to find the exact camera location in real-time. It does a motion-only BA to minimize error in placing each feature in its correct position.
    If the camera moves too quickly, the tracking might get lost. In this case, a global relocation has to be done.

  - Local mapping:
    Local mapping is responsible for the optimization of the camera pose by processing the key features and executing local BA. It is also used to iterate through the already gained information of the images and to find correspondences to unknown key features which are updated from new frames.

- Loop closure:

  Loop closure checks at first if there is a loop detected. If this is the case, a pose-graph optimization has to be performed and the map updated. The scale drift in MonoSLAM methods is quite a big problem because there are a lot of degrees of freedom where it could shift. Therefore it is necessary to use similarity transformations to correct that scale drift.

  Another solution would be to use stereo cameras if this opportunity is given.

Direct Methods

This method extract pixel of an image to calculate a 6-DoF camera position. The photometric errors are getting minimized with this calculation. Direct sparse odometry combines the photometric error with the geometric error and optimizes all model parameters together. This allows a high performance of tracking and mapping with good robustness in a feature-less environment.

2.5 Monte Carlo Localization (MCL)

The Monte Carlo Localization (MCL) [10] is a probabilistic localization algorithm based on the particle filter, which requires a map of the environment to calculate the position of a mobile robot. The particles are distributed within the entire environment and represent a possible position and orientation of the robot. By taking measurements with the LDS, a score is calculated for each particle, indicating the probability with which this position and orientation corresponding to the real robot pose. If the probability is lower than a predefined limit, these particles are going to be removed, while close to particles with higher probability, new particles will be placed to increase the density. With each new measurement, more particles are removed until eventually all of them are placed in the same position.
The big problem with MCL is the kidnapped-robot-problem, where the current position of the robot gets lost. Since this algorithm is based on a calculation with all previous positions, a loss of the current position during the abduction process would cause the localization to fail.

2.6 Adaptive Monte Carlo Localization (AMCL)

The Adaptive Monte Carlo Localization (AMCL) [28] is a combination of MCL and the KLD-Sampling [9] approach and was developed primarily to solve the kidnapped-robot problem. KLD-sampling is used to increase the efficiency of the particle filter by dynamically changing the number of particles. This is done by calculating the probability of a correct position each time the environment is sampled, and then generating the number of particles based on the score achieved. Thus, even if the kidnapped-robot problem occurs, the particles are generated based on the measurements of the new environment, and no longer based on the random initialization as in MCL. To increase efficiency, more particles are also distributed in smaller parts of the map than in larger ones, which leads to an increase in efficiency and at the same time to a reduction in computational effort.
Even though AMCL was able to solve the kidnapped-robot problem, localization issues occurred with the algorithm when multiple particles have been placed in the same position. It turned out that new measurements often calculated the particle swarm as the most probable position instead of considering the whole map by resampling.

2.6.1 Procedure of AMCL

- **Initialization:**

  The First step is always to spread the particles around the robot as shown in figure 2.1. All particles are representing the possible positions of the robot together with the possible orientation, which is indicated with the green arrows.

Figure 2.1: Initialization of AMCL

- **Prediction:**

  After the robot starts to move, the algorithm calculates the next position and orientation, which leads to a shift of these green arrows according to the traveled distance or angle. Figure 2.2 is representing the new orientation of the particles after the robot rotated for nearly 270°. This means, that the algorithm has not corrected any particles according to the measurements yet, it has just shifted the orientation.

Figure 2.2: Rotation of the robot changed the orientation of the particles

- **Correction and Resampling:**

  The next step is to move with the robot through the environment to execute more measurements. After a predefined distance or angle is reached, the algorithm starts to evaluate each particle, as described in section 2.3.1. After this is done, the unlikely particles are going to be removed, and at the most probable position will the density be increased. Picture 2.3 shows the distribution of the AMCL after the robot moved through the environment. There are several probable positions with a higher density of the particles, and the biggest one is outside the flat.

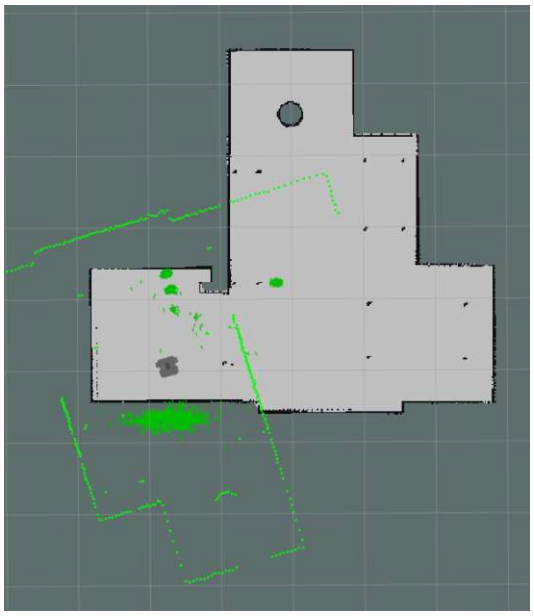


Figure 2.3: Distribution of particles after the robot has moved

This leads to the conclusion that the algorithm needs further measurements to complete the localization. If this procedure was repeated many times but did not lead to any improvement in the result, then recovery is initiated which spreads particles within the entire environment, shown in figure 2.4. This allows the robot, unlike the initialization step, to find the correct position within the entire map, and not only near the robot.

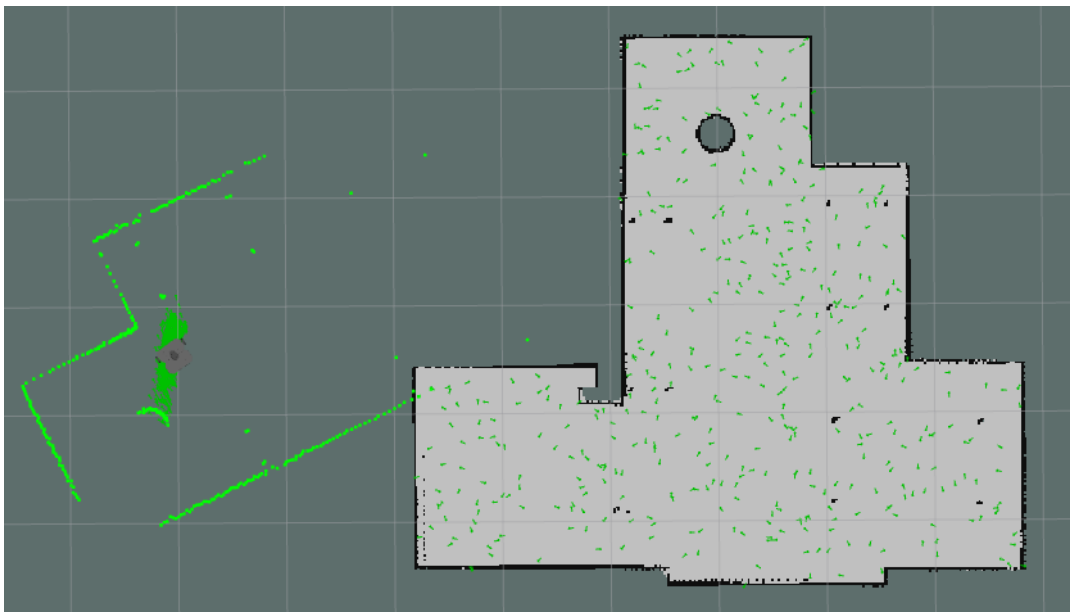


Figure 2.4: Particles are distributed within the entire environment

- **Found correct position:**

  If the particles are all in one position, it means that the localization was successful, as shown in the 2.5 figure. The green dots are representing the measurements of the Laser Distance Sensor, which are matching the object in the environment, like the walls and the furniture.

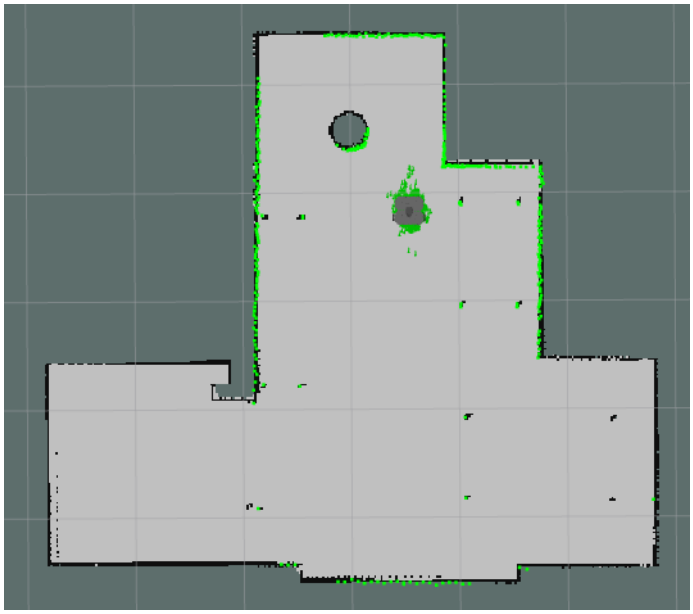


Figure 2.5: Successful localization of the robot

Chapter 3

State of the Art

In this chapter, related works and studies on SLAM and Visual SLAM approaches are analyzed and further development of AMCL is presented.

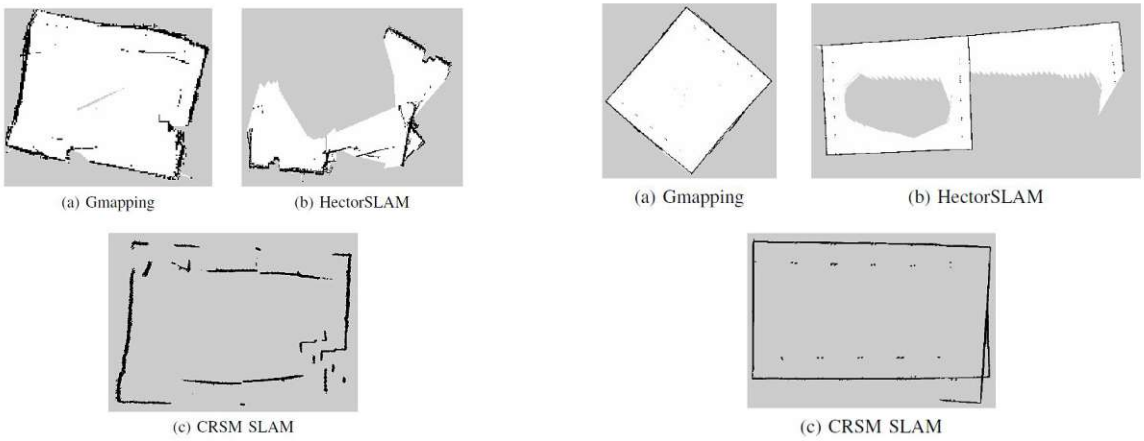
3.1 Performance Comparison of different SLAM Approaches

3.1.1 Performance Comparison of 2D SLAM Techniques available in ROS

The authors of [29] are comparing the performance of three different SLAM methods. They used a Roomba robot platform with an RGB-D Kinect sensor, which was used for emulating a laser range detector. Figure 3.1a is representing the map that was created from the environment with the Roomba robot. The results illustrate that Gmapping is the only algorithm, which is delivering useful results. The reason is that all these algorithms are expecting a laser detection sensor that is rotating around the robot. The Kinect sensor is just used to emulate such sensor values, which is leading to these uncertainties.

The same algorithms were tested with the ROS package Turtlebot1 as a robot model and the same virtual environment, which was created with Gazebo. The second figure 3.1b is showing quite good results, except Hector SLAM had problems when the robot rotated fast.

This work concludes that only Gmapping was able to achieve good results with an RGB-D Kinect Sensor.



(a) Maps created with RGB-D Kinect sensor (b) Maps created with simulation

Figure 3.1: Map creation results. Source: [29]

3.1.2 Performance Comparison of VSLAM Approaches

A different approach is investigated in [24], which compares the performance of three Visual SLAM methods: ORB-SLAM3 [3], OpenVSLAM [35] and RTAB-Map [18]. The purpose of this comparison was to clarify if one of these VSLAM methods can replace 2D SLAM in service robots. These algorithms had to perform in multiple operating domains and with several different sensors by using the three datasets EuRoC MAV, TUM RGB-D, and Kitti as benchmarks.
The results showed that OpenVSLAM was the best solution for several types of service robots. It had the best performance at re-localization and reliability, even with bad light conditions. ORB-SLAM3 was just suitable with a pair of synchronized stereo-IMU and good light conditions, but due to the periodic segmentation faults, this method is not recommendable. RTAB-Map achieved the worst overall performance with all datasets and is therefore also not able to replace 2D SLAM efficiently.

3.2 Optimizing Performance by tuning SLAM Parameters

The main focus of [1] was to evaluate the performance of the SLAM method Gmapping by tuning the number of particles, the displacement update, and the resampling threshold. Several Tests were executed with a recorded dataset and each run has generated its grid map, which was used to evaluate the accuracy of the map, the CPU load, and the memory consumption.
The tables in figure 3.2 show that especially increasing the number of particles and the resample threshold is leading to the significantly higher CPU load and memory consumption, while the linear update step is not taking much effect on them. The reason is, that a bigger number of particles has to be generated and stored, and a higher threshold increases the memory consumption because the resampling process is not triggered that often anymore. The linear update step parameter is used to configure a specific distance that the robot has to travel until the map will be updated. Increasing this parameter together with the angular update step, which sets the angle threshold for updating the map, could lead to higher utilization of the CPU.

Particles	CPU Load	Memory (MB)
5	5%	5
15	17%	54
30	17%	54
50	22%	56

(a) Effects of increasing the number of particles

Step (m)	CPU Load	Memory (MB)
0.2	20%	39
0.4	17%	31
0.7	16%	28

(b) Effects of increasing the distance of the linear update steps

Resample Threshold	Particles	CPU Load	Memory (MB)
0.1	30	15%	46
0.1	5	12%	42
1	30	23%	58
1	5	22%	54

(c) Effects of increasing the resample threshold

Figure 3.2: Effects of tuning Gmapping parameter. Source: [1]

Figure 3.3 shows, that the linear update step generates inaccuracies in the map. The marked positions in the figures represent objects and walls that were duplicated in the map due to the increase of this parameter.
The results of these experiments conclude that the number of particles should be 5, the resampling threshold should be about 0.5, and the linear update step should be as small as possible to achieve a good mapping accuracy and low computational effort for this specific environment.

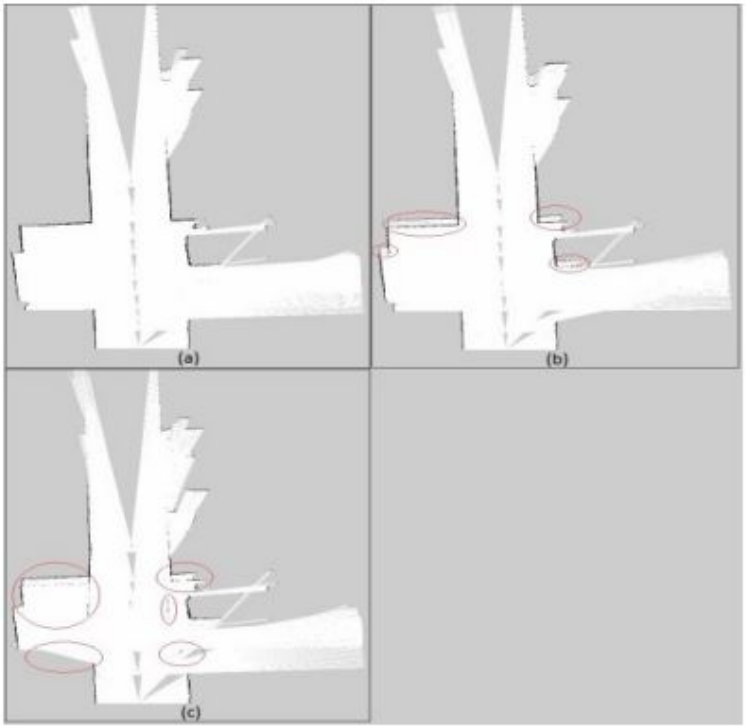


Figure 3.3: Inaccuracies in maps when increasing linear update step. Source: [1]
a) 0.1m b) 0.4m c) 0.7m

3.3 SLAM in various Applications

3.3.1 Autonomous Underwater Vehicle

The paper [8] is describing how the authors used simple sensors together with a graphical square root smoothing SLAM algorithm to carry out target reacquisition missions undersea autonomously. The system includes simple vehicle sensors and a forward-looking sonar, but no specialized navigation equipment such as a Doppler Velocity Log was implemented. This Autonomous Underwater Vehicle (AUV) was capable of performing challenging real-world tasks with a high level of autonomy. This technology could be used to perform dangerous missions underwater, which currently requires the use of humans.

3.3.2 Intelligent Robotic Lawn Mower

The main goal of [36] was to use computer vision, SLAM, A* search [14] [31] and Depth First Search (DFS) [34] to achieve higher efficiency in path planning and obstacle avoidance. The mowing process was separated into 2 rounds. The first one was to create a map of the unknown environment with SLAM and to take images of the grass, while in the second round the mower had to find, with the combination of A* search and DFS, the shortest path and scan all areas which were not mowed. The mover is using a camera to take images of the grass and together with a simple CNN structure, it should recognize if the grass is mown or not. The figure 3.4 is showing the Machine Learning process of the mower with training images.
The results of this work showed that the efficiency of path planning and obstacle avoidance can be increased by using A*search and DFS, but the recognition algorithm if the grass is already mown or not, needs still to be improved.

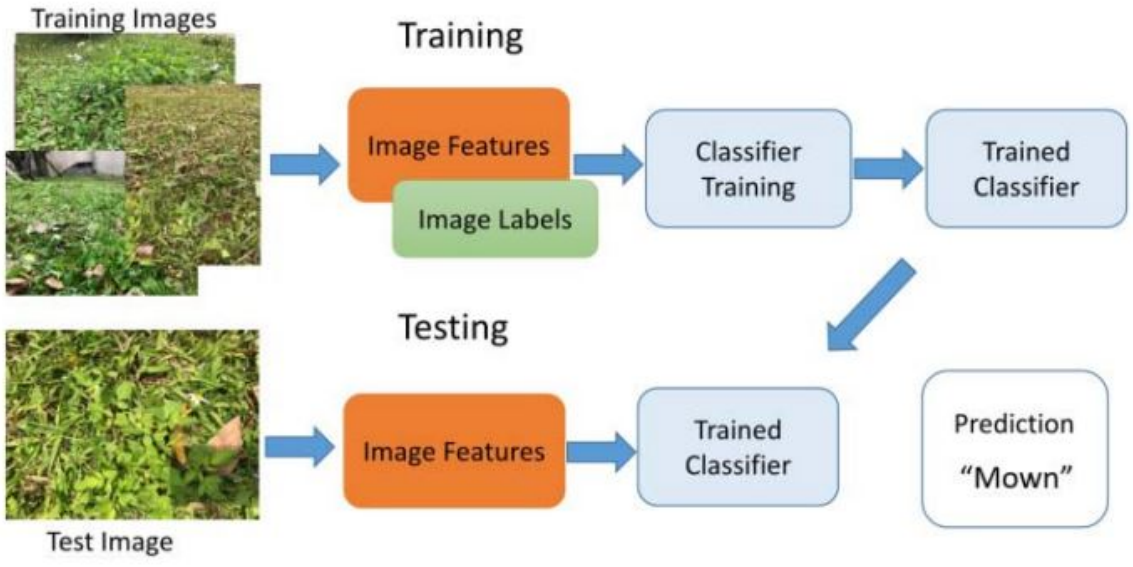
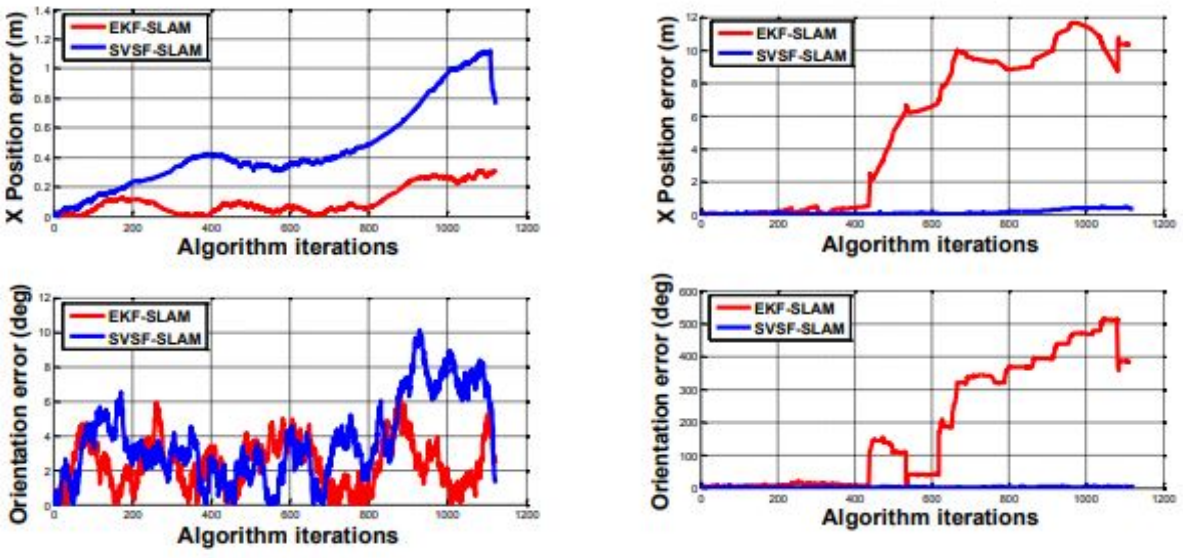


Figure 3.4: Structure of Machine Learning process. Source:[36]

3.3.3 Unmanned Ground Vehicle

In [6] is a SLAM approach with a smooth variable structure filter (SVSF) [13] for an Unmanned Ground Vehicle (UGV) used to generate a map of the environment and to locate the vehicle in it. SVSF-SLAM offers a recursive predictor-corrector estimation, which can handle the uncertainties from the laser navigation system and the odometry. This work also compares the performance of the SVSF-SLAM with the EKF-SLAM algorithm by influencing the system with Gaussian noise and colored noise caused by a variable bias.

Figure 3.5 is showing, that the estimation error of EKF-SLAM is significantly higher than that of SVSF-SLAM. The results of the experiment proved that EKF-SLAM was able to generate accurate maps with Gaussian noise, even if the estimation of the position and orientation increases with more algorithm iterations, but with colored noise, this method failed with high error. SVSF-SLAM was able to detect often loop closures, which reduces the estimation error significantly. The results of this work lead to the conclusion that the SVSF-SLAM is, compared to EKF-SLAM, an efficient and robust method, which can handle uncertainties well.



(a) Estimation error with centered Gaussian noise

(b) Estimation error with colored noise

Figure 3.5: Estimation errors of EKF/SVSF-SLAM with noise. Source:[6]

3.4 Approaches with Deep Learning Extensions for VSLAM

The work [7] describes the evolution of Visual SLAM and the use of Deep Learning.

The following approaches are representing the state-of-the-art techniques, which are currently used or under development.

3.4.1 Deep Neural Networks for Visual SLAM

Convolutional Neural Network is the most popular deep neuronal network, which uses convolutional layers as filters to detect patterns. This network needs and tensor as an input tensor, which is similar to a filter matrix, to create an output feature map of an image. The different layers are bounded local to each other, which reduces the parameter and results in better efficiency.

3.4.2 Depth Estimation with Deep Learning

The depth estimation can be realized with two different methods. The difference is that supervised methods require images and patterns, while unsupervised methods use Deep Learning to recognize structure and patterns. The second method is more commonly used because the images need a lot of memory and each pattern must be compared to the environment, which leads to a higher computational effort.

3.4.3 Ego-Motion Estimation with Deep Learning

The ego-motion between two images can be estimated by using Deep Learning. This work showed that unsupervised methods achieved good results and therefore the following techniques are based on it.
SfMLearner is often used for systems with monocular cameras. This solution is cost-effective, but a single camera is not able to calculate the absolute scale of the depth and pose.
UnDeepVO is an approach that requires a pair of cameras to learn from the environment. With 2 images from different perspectives, it is possible to calculate the depth and the pose, but this also leads to higher costs in the hardware setup.

3.5 Improved Localization Approaches Based on AMCL

3.5.1 Improved Localization of Mobile Robotic System

In [4] an improved approach of the AMCL algorithm has been developed, which significantly improves the localization in complex environments and also provides the possibility to find the correct position of the robot after a localization error occurs.
The figure 3.6 shows the flowchart of the improved AMCL algorithm, where AMCL is executed 5 times before the results are evaluated and finalized. The reason for this is that in a complex environment with the same structures, it is easy to estimate a wrong position, whereas with multiple runs the probability of a correct localization is higher. Next, an estimation of the uncertainties from the current pose is calculated using a covariance matrix. Then, possible positions, that are less than 0.5m away from each other, are considered similar positions and divided into a subgroup. Finally, the number of each group is determined and from the most significant groups, the average of the poses is calculated, which is used as the final position and orientation of the robot. Through this extension of AMCL, it was possible to successfully perform a significant improvement of localization in complex environments.

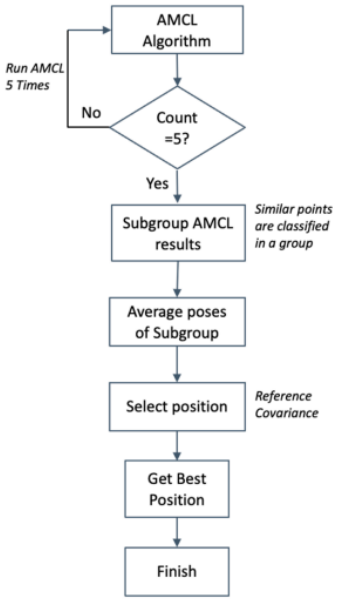


Figure 3.6: Flowchart of improved AMCL Algorithm. Source:[4]

3.5.2 Pose Detection of a Mobile Robot Based on LiDAR Data

A different approach is used in [5], where AMCL is extended with a template matching technique. First, the environment is scanned with an LDS, and the generated map is used as a template, while the available map is still used as a reference. The algorithm of the template matching technique works in such a way that the template image is always rotated by 1°, which is then converted into a binary image with a value matrix of 0 or 255 and compared with the reference image. A score is calculated from this, which represents the correspondence. The image is then rotated by 1° again, resulting in 360 template images. Based on the score of the template image with the highest match to the reference image, the robot then determines the current location within the reference image. Through this extension of AMCL, the robot could be correctly localized in all scenarios.

3.5.3 Improved Localization Algorithm for Intelligent Robot

The authors of [37] use a fusion of binocular camera and LDS in the prediction step, to improve the proposed distribution, which leads to a reduction of the number of particles. While the robot is moving, the binocular camera and the LDS simultaneously scan the environment, and as soon as the camera detects a key feature, the measurements are compared and analyzed with the scans of the LDS. The joint observations are then integrated for the proposed distribution to concentrate the particles in regions with higher probability. In addition, the fusion observation data is also used to calculate the number and distribution of particles during resampling and is taken as input for the next prediction step. The fusion of Camera and LDS led to a significant improvement in the experiments of this paper compared to AMCL.

Chapter 4

Simulation Environment and Hardware Components

This chapter describes all parts of the simulation environment and the used robot vacuum cleaner together with their hardware components.

4.1 Operating System

It is recommended to use a Linux operating system to be able to work with ROS without constraints.
The first simulations were developed with version 16.04.7, but several tests of the environment failed, and some packages occurred with compatibility issues. To get rid of these problems, it was necessary to upgrade the operating system to version 18.04.6.
With this update, it was possible to get all packages compiled without errors and the simulation running without constraints.

4.2 ROS Environment

The Robot Operating System is a middleware that provides several packages for robot software development, which includes packages for hardware abstractions, different drivers for peripheral devices, and features for visualization and simulation.

4.2.1 Turtlebot3

Turtlebot3 4.2.1 is a mobile robot model for simulations and prototyping, which is available as a package for ROS. It provides multiple sensors and actuators that are integrated with ROS. This reduces the programming effort significantly and allows to test programs for robots easily. One of the main functionalities of the Turtlebot3 is SLAM and the Navi-

gation through environments.

In this work, the Waffle Pi robot model was used, because it has implemented a 360° LDS and a camera. Both are necessary to be able to develop simulations for SLAM with LDS and VSLAM.

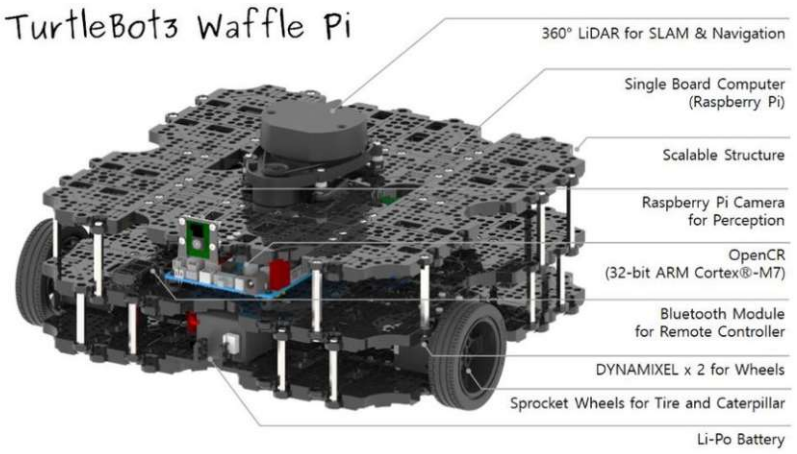


Figure 4.1: Turtlebot3 Waffle-Pi [21]

4.2.2 Gazebo

Gazebo is a 3D robot simulation software, which is compatible with ROS. For this work, it was used to combine the measurements from the Turtlebot3, the calculations of SLAM, and the navigation package to move the robot. With this platform, it was also possible to design my apartment 4.2 with the furniture and to specify the collision behavior.

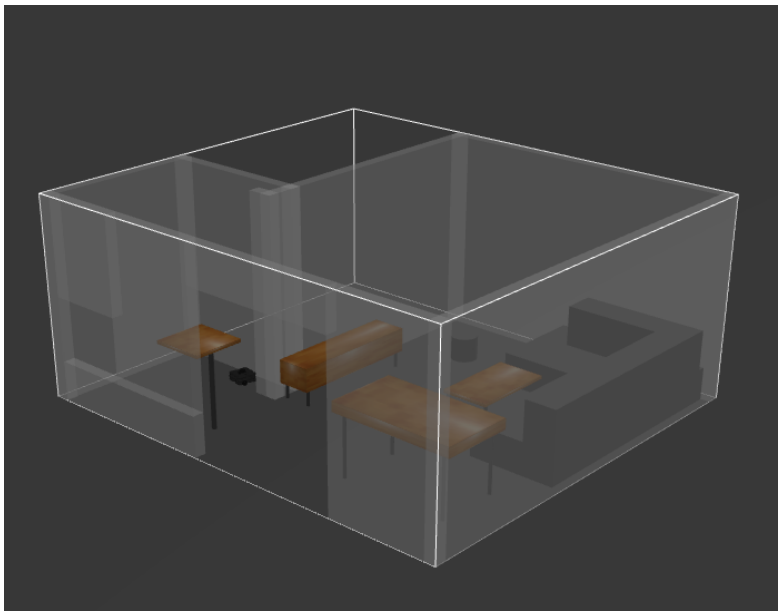


Figure 4.2: Flat designed in Gazebo

4.2.3 Rviz

Rviz is a 3D visualization Framework for ROS, which is used to visualize the robot model and the corresponding sensor. It allows to import all services from different packages and especially to visualize data points. This program was necessary, to generate an accurate occupancy grid map from the data points of the SLAM algorithm.

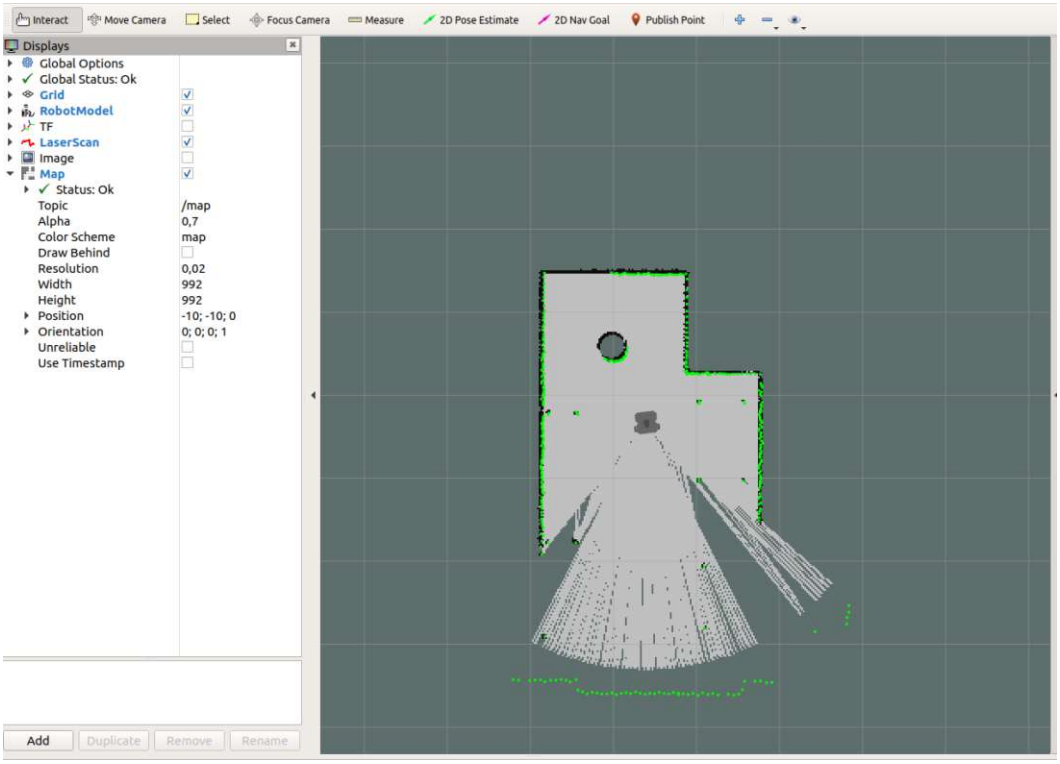


Figure 4.3: Visualization in Rviz

4.3 Hardware Components

In this thesis, the evaluation was performed in a simulation, which requires only a computer, but due to the high computational effort by Gazebo and Rviz in particular, it must have sufficient resources. In the chapters 7 and 8 tasks are executed by two different vacuum cleaning robots, which differ fundamentally in the used sensor technology.

4.3.1 General sensors

- **Ultrasonic Sensor:**
  Ultrasonic sensors are implemented in most robot vacuum cleaners to detect obstacles in front of the robot. After the ultrasonic sensor detects something that is not already on the map, the robot will slow down and continue slowly, until the light-touch bumper hits it.

- **Light-touch bumper sensor:**
  The light-touch sensor is mounted on the front and is used to collide with obstacles that were neither marked on the map nor detected by the ultrasonic sensor.

- **Infrared sensor:**

  This sensor is used to find targets with an infrared receiver, like a charging station of a robot vacuum cleaner. After a normal cleaning process, the robot would drive back to the start position and would try to detect an infrared signal of the station. If the station is now moved, the robot vacuum cleaner would have to drive through the whole environment to search the charging station.

- **Cliff sensor:**

  This sensor is needed to detect changes on the ground, such as stairs.

4.3.2 Roborock S5 Max

Additional Sensor System

- **LDS system:**

  The LDS is mounted on the top of the robot and is rotating 360° while emitting laser pulses in the environment. The reflected pulses are measured and the time of flight is taken to calculate the traveled distance. This allows to scan the environment quite accurately and create a map of it. The LDS can detect most of the obstacles in a room, except objects with small dimensions or ones out of glass.

Implemented Algorithm

The Roborock S5 Max is using a LDS system, which enables to save created maps and navigate the robot with target positions through the rooms. The main tasks of the robot are separated into first creating a map and secondly cleaning it.
The map creation process needs more time because it is scanning every room by rotating 360° and afterward, it is driving through the whole environment. This map is then used, to optimize the cleaning process.
Another feature is also the possibility to define restricted areas, which the robot cannot enter.

Figure 4.4: Robot Vacuum Cleaner - Roborock S5 Max

4.3.3 Roomba 974

Additional Sensor System

- **iAdapt-environmental camera:**
  The iAdapt camera is mounted on top of the robot and scans the ceiling and parts of the upper wall. This means, that all objects which are between the robot and the upper wall will be scanned. According to the manufacturer, it is not taking pictures of the environment but key points of a specific structure. If there are objects like a table or a shelf in front of the camera, it recognizes structures like edges or corners and saves them.
  It should be able to detect up to 230.400 key points within one second.

- **Floor tracking sensor:**
  The floor tracking sensor on the bottom of the robot is tracking the traveled distance and the orientation.
  Another task of this sensor is recognizing if the robot is drifting, for example on a carpet, and to correct the odometry before a pose estimation via SLAM is executed. This should increase the accuracy and the speed of creating the map.

Implemented Algorithm

The Roomba 974 is using VSLAM to create the map and locate itself. By scanning the ceiling with the iAdapt camera, the algorithm calculates the corresponding layout of the ground. This works well until the ceiling is not sloping and as long as it has the same layout as the ground.
An advantage in comparison with vacuum cleaning robots without cameras is that if an already scanned object is moved, like a table or a chair, the robot should recognize it and will correct the position on the map.
A problem with VSLAM and the single-camera is the light variance. If the room is very dark, the robot will have issues detecting key points and the ceiling in the room. Therefore, it is recommended from iRobot turn on the lights while the robot vacuum is cleaning.
When the cleaning process is finished, the created map is stored in the app and will be deleted afterward. This means, that the map is not used in the second run to improve the efficiency.

Figure 4.5: Robot Vacuum Cleaner - Roomba 974

Chapter 5

Setup and Scenario Specification

This chapter describes the scenarios that were defined for mapping, as well as for localization, to evaluate the different SLAM methods and AMCL. It also presents all the components of the setup that were needed to perform these scenarios.

5.1 Preparations for SLAM

For the evaluation, the three SLAM methods Gmapping, Hector, and Karto are used to create maps of the environment, which are compared to a floor plan of the virtual apartment. For this purpose, on the one hand, a consistent format of the map has to be defined and on the other hand, a program has to be developed, which compares the floor plan of the apartment with the maps and evaluates it.

5.1.1 Portable Graymap (PGM)

PGM was chosen as the format for the generated maps because each pixel from an image is defined as a grayscale value between 0 and 255, where the value 0 is equal to black and 255 corresponds to the color white. This structure is well suited for the generated maps because SLAM normally only distinguishes between occupied, free and unexplored areas and therefore only requires 3 different grayscale values. The figure 5.1 shows the image as well as the underlying ASCII format of an image in PGM format.

```
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   100 100 100 100 0   0   150 150 150 150 0   0   200 200 200 200 0
0   100 0   0   100 0   0   150 0   0   150 0   0   200 0   0   0   0
0   100 100 100 100 0   0   150 150 150 150 0   0   200 0   0   0   0
0   100 0   0   100 0   0   150 0   0   150 0   0   200 0   0   0   0
0   100 0   0   100 0   0   150 150 150 150 0   0   200 200 200 200 0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

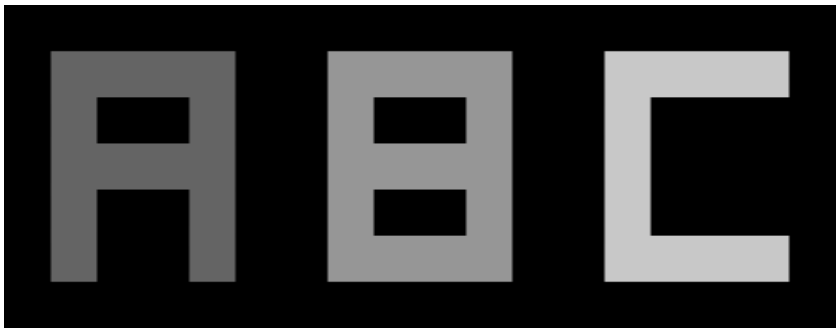
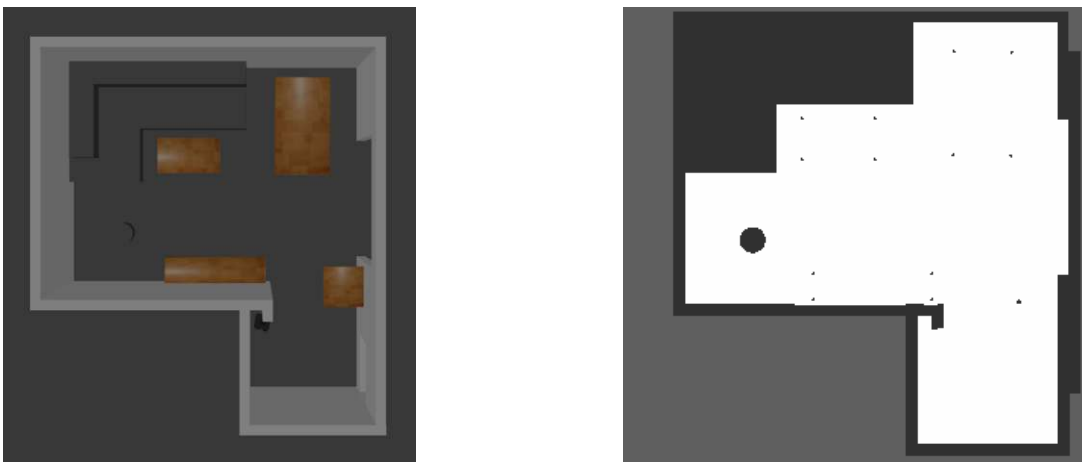


Figure 5.1: structure of PGM image

5.1.2 Creating Reference-Map

To evaluate the results of each algorithm, an ideal map of the environment had to be made to serve as a reference for comparisons with the created maps. Since any mapping method of environments cannot provide perfect accuracy, the map had to be generated directly from Gazebo. The open-source package "pgm_map_creator" was used here to generate an ideal map 5.2a in PGM format from the simulated world 5.2b. However, this ROS package was developed for Ubuntu 16.04 and the ROS version Kinect, which required some adjustments in the source code to make it compatible with Ubuntu 18.04 and the version Melodic.

A major drawback of the map created was that the program can only distinguish between occupied and free areas, and therefore the free pixels inside the apartment had the same grayscale value as the outside area. Since the interior of the apartment is essential for the evaluation, the map had to be manually reworked so that the exterior area is represented with its value.



(a) Top view of the created world

(b) Reference-map of the world

Figure 5.2: Created World and corresponding map

5.1.3 Comparing generated Maps

The map comparison proved to be a major challenge because no ordinary pixel comparison of the images could be performed since the same scaling or even positioning cannot be guaranteed when saving the map. Therefore, in some tests with direct pixel comparison, very large discrepancies were calculated for almost identical maps because the created image was shifted by a few pixels.

To solve this problem, a program was developed that generates a histogram from the PGM images, as illustrated in Figure 5.3. From this, it can be seen that the free pixels are represented with a value of 254, the occupied pixels are represented with a grayscale value of 0, and the unexplored is represented with a value of 205.

0 : 2144	1 : 0	2 : 0	3 : 0	4 : 0	5 : 0	6 : 0	7 : 0	8 : 0	9 : 0
10 : 0	11 : 0	12 : 0	13 : 0	14 : 0	15 : 0	16 : 0	17 : 0	18 : 0	19 : 0
20 : 0	21 : 0	22 : 0	23 : 0	24 : 0	25 : 0	26 : 0	27 : 0	28 : 0	29 : 0
30 : 0	31 : 0	32 : 0	33 : 0	34 : 0	35 : 0	36 : 0	37 : 0	38 : 0	39 : 0
40 : 0	41 : 0	42 : 0	43 : 0	44 : 0	45 : 0	46 : 0	47 : 0	48 : 0	49 : 0
50 : 0	51 : 0	52 : 0	53 : 0	54 : 0	55 : 0	56 : 0	57 : 0	58 : 0	59 : 0
60 : 0	61 : 0	62 : 0	63 : 0	64 : 0	65 : 0	66 : 0	67 : 0	68 : 0	69 : 0
70 : 0	71 : 0	72 : 0	73 : 0	74 : 0	75 : 0	76 : 0	77 : 0	78 : 0	79 : 0
80 : 0	81 : 0	82 : 0	83 : 0	84 : 0	85 : 0	86 : 0	87 : 0	88 : 0	89 : 0
90 : 0	91 : 0	92 : 0	93 : 0	94 : 0	95 : 0	96 : 0	97 : 0	98 : 0	99 : 0
100 : 0	101 : 0	102 : 0	103 : 0	104 : 0	105 : 0	106 : 0	107 : 0	108 : 0	109 : 0
110 : 0	111 : 0	112 : 0	113 : 0	114 : 0	115 : 0	116 : 0	117 : 0	118 : 0	119 : 0
120 : 0	121 : 0	122 : 0	123 : 0	124 : 0	125 : 0	126 : 0	127 : 0	128 : 0	129 : 0
130 : 0	131 : 0	132 : 0	133 : 0	134 : 0	135 : 0	136 : 0	137 : 0	138 : 0	139 : 0
140 : 0	141 : 0	142 : 0	143 : 0	144 : 0	145 : 0	146 : 0	147 : 0	148 : 0	149 : 0
150 : 0	151 : 0	152 : 0	153 : 0	154 : 0	155 : 0	156 : 0	157 : 0	158 : 0	159 : 0
160 : 0	161 : 0	162 : 0	163 : 0	164 : 0	165 : 0	166 : 0	167 : 0	168 : 0	169 : 0
170 : 0	171 : 0	172 : 0	173 : 0	174 : 0	175 : 0	176 : 0	177 : 0	178 : 0	179 : 0
180 : 0	181 : 0	182 : 0	183 : 0	184 : 0	185 : 0	186 : 0	187 : 0	188 : 0	189 : 0
190 : 0	191 : 0	192 : 0	193 : 0	194 : 0	195 : 0	196 : 0	197 : 0	198 : 0	199 : 0
200 : 0	201 : 0	202 : 0	203 : 0	204 : 0	205 : 940298	206 : 0	207 : 0	208 : 0	209 : 0
210 : 0	211 : 0	212 : 0	213 : 0	214 : 0	215 : 0	216 : 0	217 : 0	218 : 0	219 : 0
220 : 0	221 : 0	222 : 0	223 : 0	224 : 0	225 : 0	226 : 0	227 : 0	228 : 0	229 : 0
230 : 0	231 : 0	232 : 0	233 : 0	234 : 0	235 : 0	236 : 0	237 : 0	238 : 0	239 : 0
240 : 0	241 : 0	242 : 0	243 : 0	244 : 0	245 : 0	246 : 0	247 : 0	248 : 0	249 : 0
250 : 0	251 : 0	252 : 0	253 : 0	254 : 41622	255 : 0				

Figure 5.3: histogram of generated map

One problem with comparing the SLAM-generated maps and the reference image is that the ideal map was designed from the top view, and thus the thickness of the wall, as well as the depth of the couch, is significantly different from the generated maps, as can be seen when comparing 5.2b and 5.4. The reason for this is that the robot only measures the distance to obstacles with the LDS, but cannot measure the depth of the objects as well. Therefore, to still be able to perform a quantitative comparison of different images, only the free cells were used for the calculation.

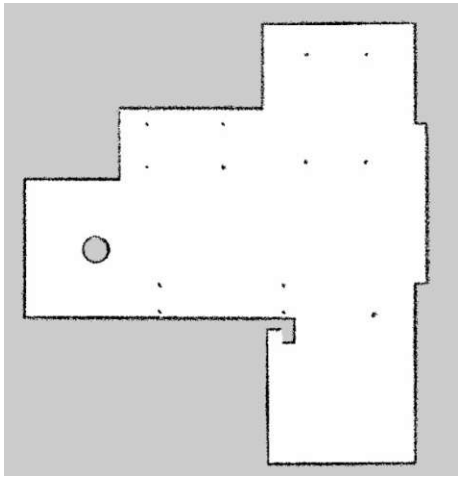


Figure 5.4: SLAM-generated map

Figure 5.5 shows the first part of the developed program. Lines 13 - 22 store the information of the two images, i.e. the header and the grayscale values of each pixel, and then the number of pixels in a line is extracted from their headers. Since the images were saved in an ASCII-PGM format, all grayscale values of the pixels are given in a row, which is divided into the actual number of pixel widths into rows 25 and 26 and saved in a list. To be able to indicate the difference in percentage related to the reference map, the number of free pixels in the reference map is determined in the following. For the evaluation of the difference, only the free pixels are needed, which is why in 34 - 41 all lines of the list are discarded that do not contain free pixels. Afterward, the free pixels per list element are counted, which correspond to the lines of the images.

```
12      #Open both maps
13      file1 = open(args.map1)
14      file2 = open(args.map2)
15
16      #Store all lines of the maps in lists
17      content1 = file1.readlines()
18      content2 = file2.readlines()
19
20      #Extract the number of pixels in each line
21      LengthImage1 = int(content1[2][:content1[2].find(" ")])
22      LengthImage2 = int(content2[2][:content2[2].find(" ")])
23
24      #Split the pixelvalues in a list with the length of the image
25      TempLineArrayImage1 = [content1[4][i : (i + LengthImage1)] for i in range(0, len(content1[4]), LengthImage1)]
26      TempLineArrayImage2 = [content2[4][i : (i + LengthImage2)] for i in range(0, len(content2[4]), LengthImage2)]
27
28      LineArrayImage1 = []
29      LineArrayImage2 = []
30
31      #Count all free pixel in ReferenceMap
32      totalFreePixel = content1[4].count(chr(254))
33
34      #Check all lines for free pixels and delete all others
35      for i in TempLineArrayImage1:
36          if i.find(chr(254)) > -1:
37              LineArrayImage1.append(i)
38
39      for i in TempLineArrayImage2:
40          if i.find(chr(254)) > -1:
41              LineArrayImage2.append(i)
42
43      listFreePixelMap1 = []
44      listFreePixelMap2 = []
45
46      #Count free pixels in each line and add them to the list
47      for x in LineArrayImage1:
48          listFreePixelMap1.append(x.count(chr(254)))
49
50      for x in LineArrayImage2:
51          listFreePixelMap2.append(x.count(chr(254)))
```

Figure 5.5: First Part of the Program

The next figure 5.6 shows the second part of the program, where at the beginning it is checked which of the two maps contains the larger number of free pixels, which is essential for the calculation of the percent difference. If the number of free pixels of the SLAM-generated map is larger than that of the reference map, this indicates that due to the inaccuracy of the LDS individual free pixels have been inserted, for example, in walls. Therefore, in lines 60 -72 a filter is applied, which removes these outliers from the list and sums them up so that the free areas of the two images start and end in the same line. Afterward, the differences of the free pixels of these lists are still compared line by line and added to the outliers, whereby the program has calculated the total number of differences of the free pixels considered line by line. Finally, the difference is calculated as a function of the number of free pixels in the image of the reference map.

```
53      status = True
54      NumberOfDifferentPixels = 0
55
56      #Check if ReferenceMap has less lines of free pixels
57      if (len(listFreePixelMap1) < len(listFreePixelMap2)):
58          while status:
59              #Filter to move free pixels which results from Uncertainties in the upper wall, to separate List
60              if ((listFreePixelMap2[0] < 70) and (len(listFreePixelMap1) < len(listFreePixelMap2))):
61                  NumberOfDifferentPixels = NumberOfDifferentPixels + listFreePixelMap2[0]
62                  listFreePixelMap2.pop(0)
63              else:
64                  status = False
65
66          #Move all lines at the end of ComparedMap to separate List, to achieve two lists with same number of lines
67          listOfDifferentPixels = listFreePixelMap2[len(listFreePixelMap1):]
68          listFreePixelMap2 = listFreePixelMap2[:len(listFreePixelMap1)]
69
70          #Sum up all moved lines to the removed free pixels from the filter
71          for i in listOfDifferentPixels:
72              NumberOfDifferentPixels = NumberOfDifferentPixels + i
73
74      else:
75          #Same procedure if ReferenceMap has more lines of free pixels
76          listOfDifferentPixels = listFreePixelMap1[len(listFreePixelMap2):]
77          listFreePixelMap1 = listFreePixelMap1[:len(listFreePixelMap2)]
78
79      #Add the difference of the number of pixels from each line to the moved free pixels
80      for i in range(len(listFreePixelMap1)):
81          NumberOfDifferentPixels = NumberOfDifferentPixels + abs(listFreePixelMap1[i] - listFreePixelMap2[i])
82
83      #Calculate difference
84      Difference = float(NumberOfDifferentPixels) / float(totalFreePixel) * 100.0
```

Figure 5.6: Second Part of the Program

5.2 Preparations for VSLAM

For the evaluation of the VSLAM approaches, a 3D spare point map had to be created from ORB-SLAM and Mono-SLAM respectively, which will also be compared with the floor plan of the virtual apartment. To create a 2D Occupancy grid map from the 3D map, an open-source wrapper [30] is used, which takes the sparse points from a specific height for the 2D map. For the implementation of ORB-SLAM and Mono-SLAM in the simulation, the packages from [25] and [32] are used.

5.3 Preparations for AMCL

The AMCL algorithm from the Turtlebot3_Navigation Package was used for the autonomous localization of the robot. A precondition for this localization is an existing map of the environment and an initial pose estimation.
This algorithm also contains a recommended parameterization, which however is based on a manual optimization of the current position. This required the user to estimate the robot's position on the map, whereupon the robot used AMCL to try to find the true position based on the sensor data. This process had to be repeated until the exact position was found. The parameters were therefore adjusted in a way that the robot performed the position estimation in its immediate environment.
But the goal in the scenarios was for the robot to independently locate itself in the entire environment using AMCL. The figure 5.7 shows the configuration used for localization.

```
<!-- AMCL -->
<node pkg="amcl" type="amcl" name="amcl">

  <param name="min_particles"             value="500"/>
  <param name="max_particles"             value="3000"/>
  <param name="kld_err"                   value="0.0001"/>
  <param name="update_min_d"              value="0.20"/>
  <param name="update_min_a"              value="0.20"/>
  <param name="resample_interval"         value="20"/>
  <param name="transform_tolerance"       value="0.5"/>
  <param name="recovery_alpha_slow"       value="0.001"/>
  <param name="recovery_alpha_fast"       value="0.1"/>
  <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
  <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
  <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
  <param name="gui_publish_rate"          value="50.0"/>

  <remap from="scan"                      to="$(arg scan_topic)"/>
  <param name="laser_max_range"           value="3.5"/>
  <param name="laser_max_beams"           value="180"/>
  <param name="laser_z_hit"               value="0.5"/>
  <param name="laser_z_short"             value="0.05"/>
  <param name="laser_z_max"               value="0.05"/>
  <param name="laser_z_rand"              value="0.5"/>
  <param name="laser_sigma_hit"           value="0.2"/>
  <param name="laser_lambda_short"        value="0.1"/>
  <param name="laser_likelihood_max_dist" value="2.0"/>
  <param name="laser_model_type"          value="likelihood_field"/>

  <param name="odom_model_type"           value="diff"/>
  <param name="odom_alpha1"               value="0.1"/>
  <param name="odom_alpha2"               value="0.1"/>
  <param name="odom_alpha3"               value="0.1"/>
  <param name="odom_alpha4"               value="0.1"/>
  <param name="odom_frame_id"             value="odom"/>
  <param name="base_frame_id"             value="base_footprint"/>

</node>
```

Figure 5.7: parameter of AMCL

The following parameters [2] had to be adjusted through extensive experimentation in order to perform a autonomous localization within the environment:

- **kld_err:**
  This parameter manipulates the maximum error between the true distribution and the estimated one. This value was reduced by a factor of 10, forcing the algorithm to search in more detail for a suitable distribution that corresponds to the low error rate. The result of this adjustment was that the particles were distributed in a significantly larger radius around the robot.

- **resample_interval:**

  With this parameter, it was possible to increase the number of filter updates before the algorithm started resampling all particles. This allows the robot to extract and evaluate more information from the environment before resampling the particles. This value is set to 2 by default, but it was necessary to increase it to 20 filter updates.

- **recovery_alpha_slow:**

  The recovery_alpha_slow parameter is used to decide when the algorithm should recover by adding random poses. It was off by default, which meant that particles were not changed when AMCL judged a pose to be correct, even if the robot pose was incorrect. Changing this value to 0.001 redistributed the particles if new calculations deemed the current position to be not probable enough.

- **recovery_alpha_fast:**

  This parameter, different from recovery_alpha_slow, is used to set the fast average weight filter, which means that the response time is much higher. This value has also been disabled by default and set to 0.1.

5.4 Scenario Specification for SLAM

The goal of this test scenario was to evaluate the accuracy of the generated maps from the different SLAM methods throughout the program. The technical specifications of the Turtlebot3 regarding odometry and laser navigation are described in a URDF file and can be modified as required. The figure 5.8 shows the configurable parameters of the LDS, which were used to simulate inaccuracies.

```
<gazebo reference="base_scan">
  <material>Gazebo/FlatBlack</material>
  <sensor type="ray" name="lds_lfcd_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>$(arg laser_visual)</visualize>
    <update_rate>5</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>360</samples>
          <resolution>1</resolution>
          <min_angle>0.0</min_angle>
          <max_angle>6.28319</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.120</min>
        <max>3.5</max>
        <resolution>0.015</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_lds_lfcd_controller" filename="libgazebo_ros_laser.so">
      <topicName>scan</topicName>
      <frameName>base_scan</frameName>
    </plugin>
  </sensor>
</gazebo>
```

Figure 5.8: LDS specification of Turtlebot3 Waffle_Pi

An important criterion for this scenario was to have the robot use the same path for mapping in the simulation to eliminate inaccuracies due to different navigation. Figure 5.9 shows the default path that was followed in all simulations.

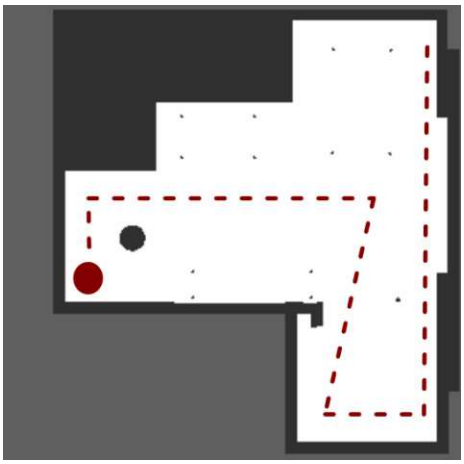


Figure 5.9: path to generate map

The process of this scenario differs only from the LDS parameters and looks like this:

- Configuration of LDS-parameter in URDF-file
- Load environment and initialize Turtlebot3
- Navigation through the environment by the specified path
- Save map as PGM-file
- Calculate the differences from the reference map

5.5 Scenario Specification for VSLAM

The goal for the scenarios for VSLAM was to create maps with ORB-SLAM and Mono-SLAM with different resolutions of the camera. For mapping, the same path as in 5.4 was followed to not generate additional inaccuracies. Therefore the scenarios of VSLAM and SLAM are identical until changing the resolution of the camera instead of the parameters of the LDS.

5.6 Scenario Specification for AMCL

The goal of these scenarios was to determine the limits of self-localization with AMCL in simulation. To accomplish this, the robot had to find its correct position while the tasks increased the complexity and difficulty of the conditions in the environment.

In all simulations, the first step was to perform a pose estimation to place the robot at the desired position and with the required orientation. Then a reference process was started, which is divided into 3 phases. In the first one, the robot tries to find the current position on the map with a reference rotation of 360°. If this resulted in no success or only inaccurate localization, two more rotations were performed. In the last referencing phase, the robot was navigated along the walls in the virtual apartment to also take dynamic measurements of other areas of the apartment.

Since autonomous localization position, as well as the orientation of the robot, are important, the following scenarios were defined to evaluate AMCL under different conditions, while the configured parameters remained unchanged.

- Localization with correct pose estimation and correct orientation
- Localization with correct position and shifted orientation
- Localization with shifted position and correct orientation
- Localization with shifted position and shifted orientation

Chapter 6

Evaluation

This chapter evaluates all the results of the scenarios defined in the 5 section.

6.1 Evaluation of SLAM

For the evaluation of the impact of the different parameters of the LDS, three maps were generated for each SLAM method and for each value, which was compared with the program. From these results, the mean value was then calculated, which is entered in the respective tables as the difference between the SLAM-generated maps and the reference maps.

6.1.1 Default Parameter Set

These parameters correspond to the specifications of the LDS-01 [19] sensor, which is installed by default on the Turtlebot3 Waffle Pi. The table 5.8 shows that 360 samples are taken per revolution, the standard deviation is 0.01m, and the update rate of the sensor is set to 5Hz. These parameters are considered optimal for the Turtlebot3 and therefore should also produce the best result in terms of mapping with SLAM.

Standard deviation[m]	Samples per revolution	Update rate[Hz]
0.01	360	5

Table 6.1: Default parameter

The figure 6.1 shows the reference map, as well as the three SLAM-generated maps. Visually, the maps are almost identical, and also the evaluation showed, as shown in table 6.2, only a difference of the free pixels of less than 1%. However, if we look at the number of faulty pixels, for example in Gmapping and Karto, it becomes clear that Karto-SLAM has about 37% more faulty pixels in its generated map. The reason for the small percentage difference in the reference map is that the total number of free pixels is very high and thus the percentage is very small. Therefore, even the smallest differences in the results are an indicator for several outliers, which occur incorrectly in objects and especially the walls.

This scenario therefore already provides us with a preliminary ranking of the 3 examined algorithms with ideal parameterization of the LDS, in which Gmapping takes the first place, Hector the second, closely followed by Karto with the third place.

(a) Reference map

(b) SLAM - Gmapping

(c) SLAM - Hector

(d) SLAM - Karto

Figure 6.1: Generated maps with default parameter

	Gmapping	Hector	Karto
Difference[%]	2.05	2.64	2.81
Pixel Errors	888	1142	1219

Table 6.2: Results with default parameter

6.1.2 Standard Deviation

The standard deviation is a measure of the dispersion of the measured distances to objects and is crucial for the precision of the LDS. The variation of this parameter, therefore, leads to measurement inaccuracies for objects and obstacles, which have to be corrected by the SLAM methods. The table 6.3 shows the differences to the reference map at standard deviations of 0.01m to 0.2m, which corresponds to the inaccuracy of 0.06% to 5% at up to 4m range.

	Difference[%]		
Standard deviation[m]	Gmapping	Hector	Karto
0.01	2.05	2.64	2.81
0.02	2.36	3.37	3.50
0.03	2.43	Failed	4.58
0.05	3.07	Failed	5.94
0.06	7.40	Failed	5.46
0.08	7.75	Failed	4.91
0.1	12.18	Failed	5.89
0.2	31.61	Failed	11.75

Table 6.3: Result with different Standard Deviation

The most remarkable thing during these runs was that Hector-SLAM already failed to map successfully from a standard deviation of 0.03m, as can be seen in the image 6.2. Up to the first half of the apartment, the robot was still able to map the environment without errors, but a shift in the map occurred during the first rotation, which led to an overlay. Firstly, this behavior is based on the fact that the Hector algorithm does not use odometry data for mapping, which makes the measurement of the distance solely dependent on the LDS. Therefore, the greater the measurement inaccuracy, the more difficult this SLAM approach is to optimize.

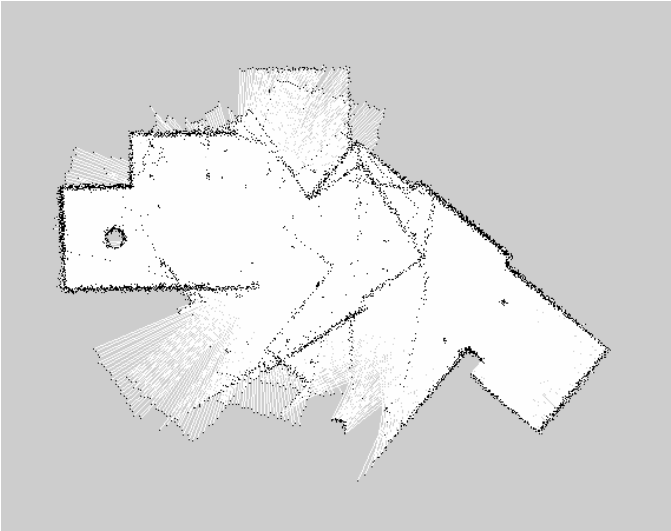


Figure 6.2: Map of Hector-SLAM with Standard Deviation of 0.03m

The second reason is that Hector-SLAM is based on the EKF, which calculates and stores the position and orientation of the robot, the position of all landmarks, and the corresponding uncertainties between them. The faster the robot moves and rotates, the more complex the calculations of the position and its uncertainties to the landmarks become, which in turn increases the computational effort. This causes problems to determine the correct position in time, which leads to an overlapping of the scan data.
Therefore, for another experiment, a fast rotation during mapping was performed with the default parameter set and lower resolution, which also resulted in an overlay, as shown in Figure 6.3. While this problem already occurred with very small deviations with Hector-SLAM, this behavior could not be reproduced with either Gmapping or Karto.

.

The evaluation results also show that even with a standard deviation of 0.02m, the ranking of SLAM methods remains

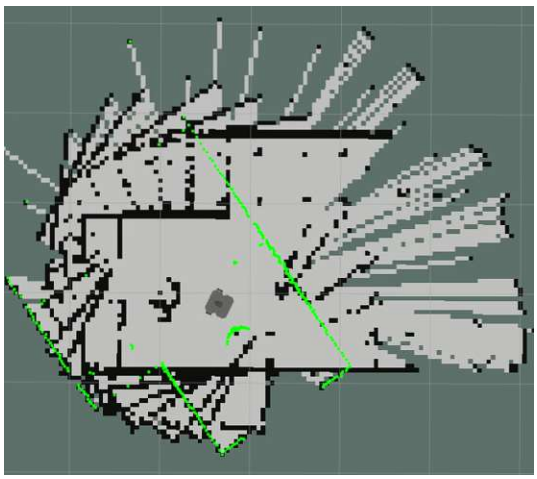
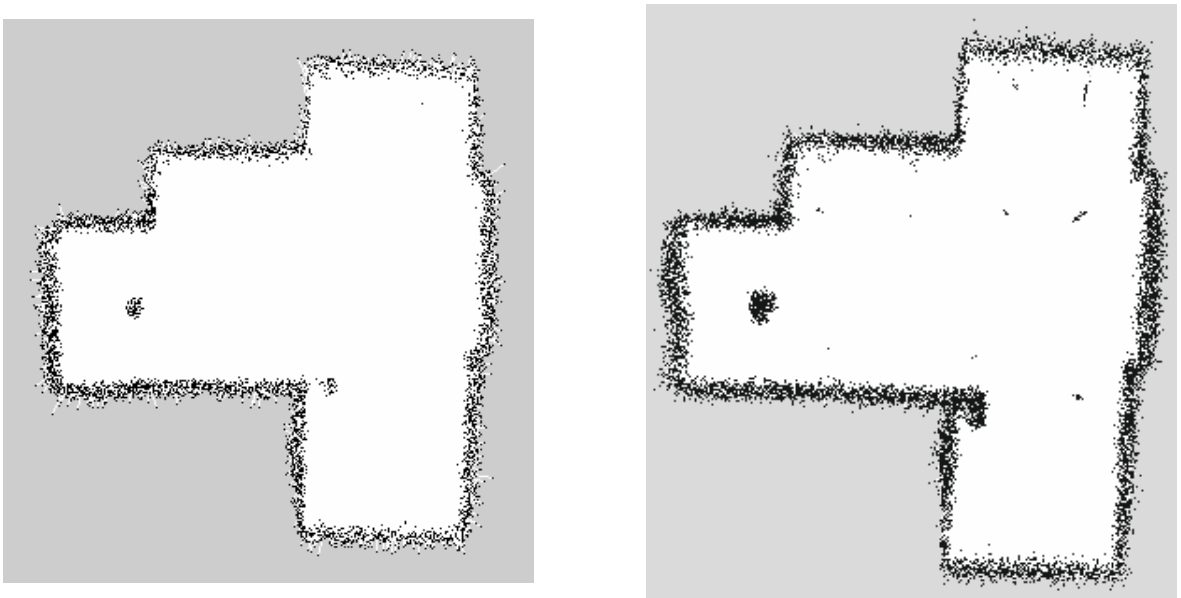


Figure 6.3: Mapping error with Hector SLAM

unchanged. Up to a deviation of 0.05m, the difference for Gmapping increased by about 1%, while the accuracy of maps generated by Karto-SLAM decreased by more than half. It became interesting from the standard deviations of 0.06m up to 0.1m, since Karto achieved almost constant results, while the accuracy of the map of Gmapping decreased by a factor of 4. As Figure 6.4 shows, although the maps of both approaches are strongly distorted, the map of Karto-SLAM provides significantly more details, such as all table legs.



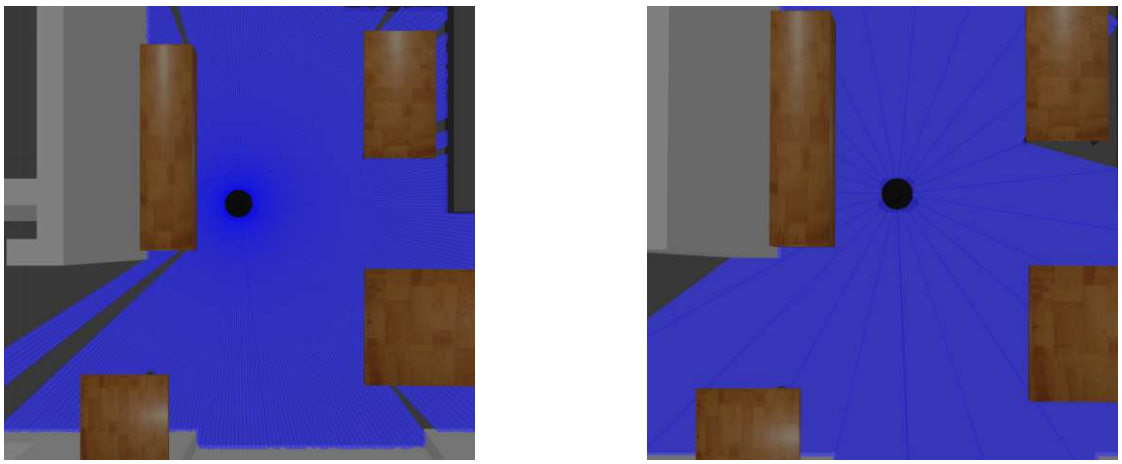
(a) Gmapping-SLAM

(b) Karto-SLAM

Figure 6.4: Generated Maps with Standard Deviation of 0.1m

6.1.3 Samples per Revolution

As shown in Figure 6.5, this parameter specifies the number of measurement points per revolution. In the left image, the 360 samples are as fine as to be almost unnoticeable, while in the right image, the 24 measurement points are easily countable. If this parameter is reduced, fewer measuring points are recorded at the same time, which means that the robot either has to stay longer at positions or has to scan the same area more often to achieve the same coverage.



(a) 360 Samples/Revolution (b) 24 Samples/Revolution

Figure 6.5: Scanning environment with different Samples/Revolution

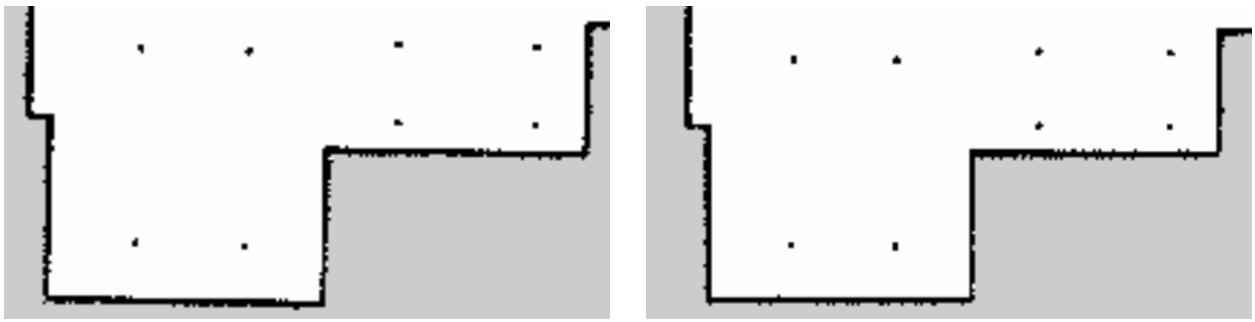
In the table 6.4 it is shown again that Hector-SLAM already has problems with 270 samples/revolution to determine the position of the robot, which again leads to overlap during mapping.

	Difference[%]		
Samples/Revolution	Gmapping	Hector	Karto
720	2.69	2.11	2.35
540	1.83	2.43	2.69
360	2.05	2.64	2.81
270	2.43	Failed	3.64
180	2.87	Failed	4.09

Table 6.4: Result with different Samples/Revolution

While up to 720 samples/revolution led to an improvement for the Hector and Karto methods, Gmapping achieved worse results even with the higher resolution. A closer look at the Gmapping-generated maps showed that the interior walls had significantly more pixel errors than at a lower resolution. The figure 6.6 shows an enlarged section of the maps, where it can be seen that the edges were rounded and the walls had significantly more pixel errors. Gmapping uses the particle filter as the only method, which distributes up to 100 particles in the environment, each of them representing a possible position and orientation of the robot. Increasing the resolution also generates twice as many measurement points with a standard deviation of 0.01m, which have to be compared with the estimated measurements of the particles. This increases the computational effort as well as the error rate of marking the correct pixel as free or occupied in the Occupancy Grid Map with a resolution of 0.02m.

The results also show that a decrease of samples/revolution has a small influence on Gmapping, while with Karto-SLAM the difference increased from 2.81% at 360 samples/revolution to 4.09% at half the resolution.



(a) 720 Samples/Revolution (b) 540 Samples/Revolution

Figure 6.6: Maps generated with Gmapping-SLAM with different Samples/Revolution

6.1.4 Update Rate

The update rate specifies how often per second the 360° measurement is performed. This parameter is especially important if the mobile robot has to perform fast movements and rotations during the measurement. A too low value would lead to either a distorted map or a gap between the measurements. The Turtlebot3 Waffle Pi used for this thesis achieves a maximum translational velocity of 0.26m/s and a maximum rotational velocity of 1.82rad/s. Therefore, even with an update rate of 1Hz, it could perform a 360° measurement every 0.26m, which should be sufficient for most environments. The assumption is also confirmed by the results in the table 6.5 since the difference in the maps differs by less than 1%. Only Karto SLAM records a significant change in the difference to 4.44% with the lowest updater rate.

	Difference[%]		
Update rate[Hz]	Gmapping	Hector	Karto
500	2.05	2.26	2.64
250	2.01	2.30	2.52
100	2.00	2.31	2.49
50	2.00	2.36	2.46
5	2.05	2.64	2.81
1	2.24	3.03	4.44

Table 6.5: Result with different update rate

6.2 Evaluation of VSLAM

The implementation of ORB-SLAM in the existing simulation caused several compilation errors, which could be partially solved by adapting the source code of the package. This made it possible to use the camera to match various feature points, such as the bar table or the TV shelf, as shown in 6.7. However, no 3D map of the sparse points of the scanned environment could be created.

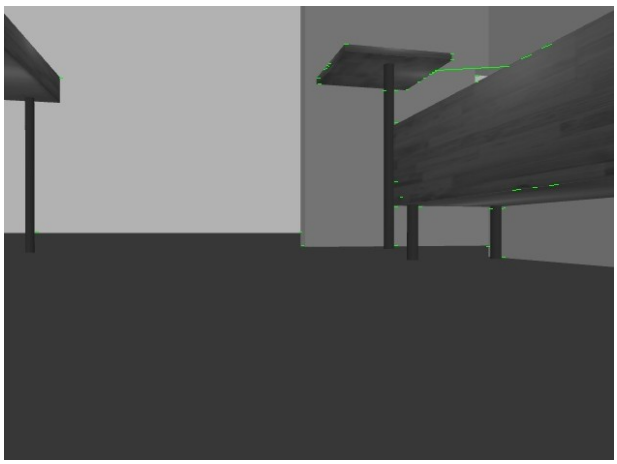


Figure 6.7: ORB SLAM matching BoW

When compiling Mono-SLAM a lot of errors were generated, which could not be solved even by intensive research. Furthermore, it was also not possible to create a 2D map from a 3D map with an open-source wrapper, because there was a version conflict with the simulation and the wrapper was not fully developed yet.

6.3 Evaluation of AMCL

During the evaluation of AMCL, the robot always stayed in the same position, while a pose estimation was performed with the framework Rviz to determine the initial position. Only when localization with the second referencing phase was unsuccessful, meaning multiple rotations were performed, was the robot moved through the environment using manual navigation through the keyboard.

6.3.1 Result of Localization with correct Position and correct Orientation

The figure 6.8 shows the initial position in this scenario, which was defined as close to the real robot position. The light green dots represent the actual measurements of the laser, from which it can be seen that they almost matched the map. The darker green arrows around the robot are the randomly distributed particles of the AMCL, which represent the possible positions and orientations of the robot. The following figure 6.9 represents the result of the localization after a reference rotation. On the one hand, it can be seen that the measuring points of the LDS correspond to the walls as well as to the other objects on the map, on the other hand, the distribution of the particles has focused on the robot's position. This indicates that the localization was successful since all particles assumed the same position and orientation.

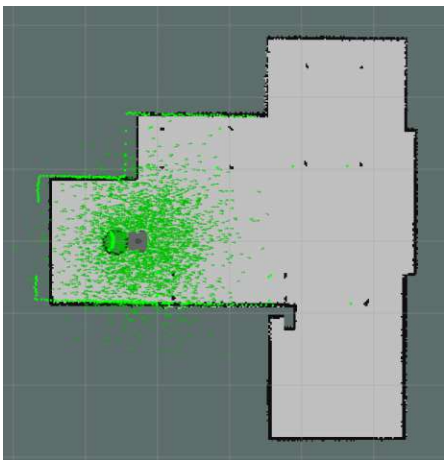


Figure 6.8: Pose Estimation
Correct Pose

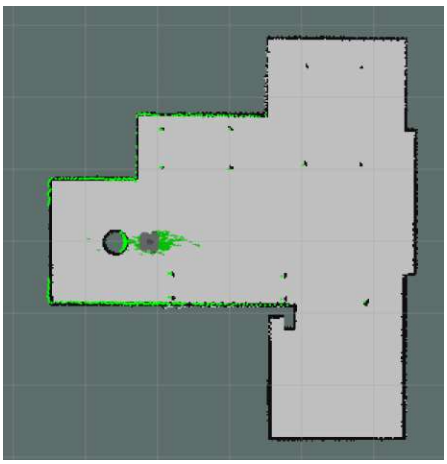
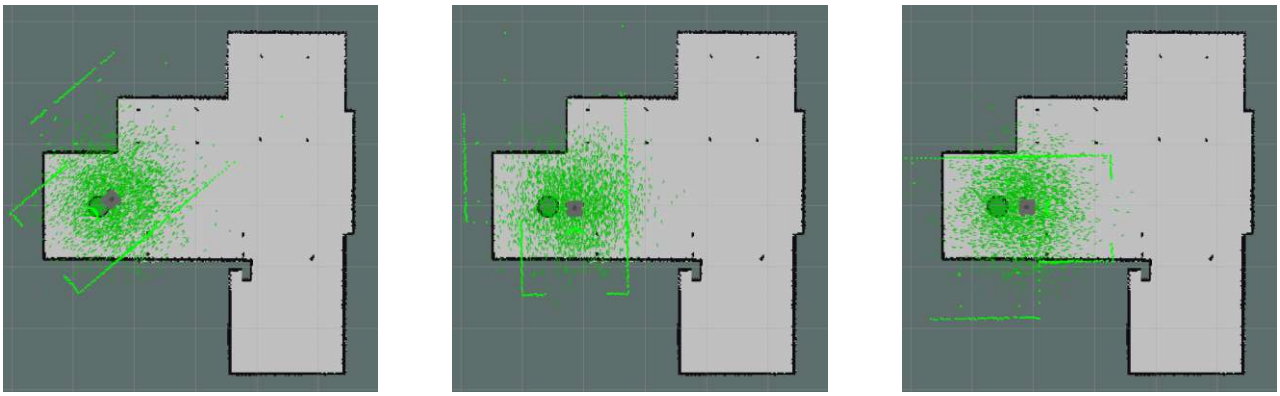


Figure 6.9: Localization after Reference Rotation
Correct Pose

6.3.2 Result of Localization with correct Position and shifted Orientation

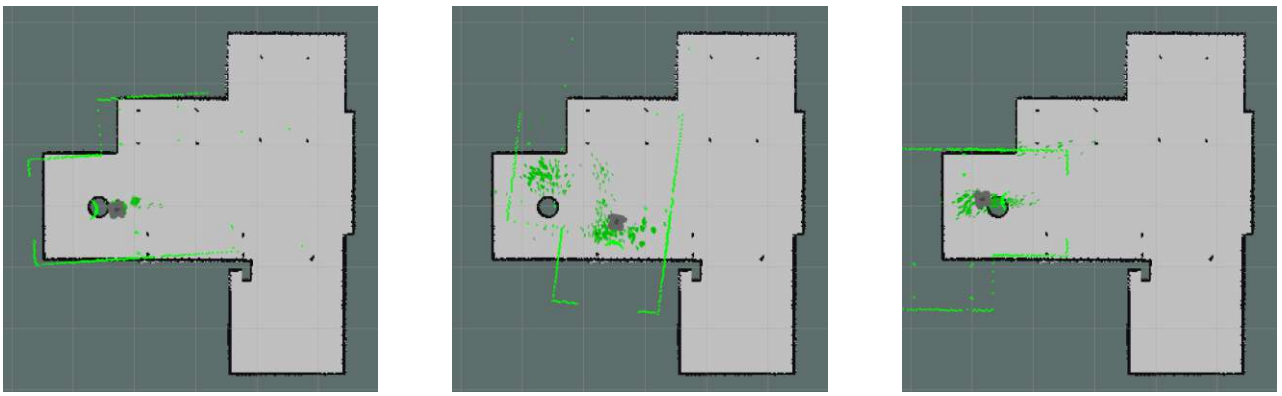
In this scenario, the correct position of the robot was given, but the orientation was changed, as can be seen in figure 6.10. Therefore, to extensively test the localization with an offset orientation, we started with an offset of 45°. To further increase the difficulty, the AMCL was also performed with an offset of 90° and finally with 180°.



(a) Offset: 45° (b) Offset: 90° (c) Offset: 180°

Figure 6.10: Pose Estimation - Correct Position and shifted Orientation

The figure 6.11 represents the results of the AMCL with the three different offsets of the orientation after a reference rotation. From this, it can be seen that with a shift of 45° the localization was already successful after one reference rotation since the measurement points of the LDS almost completely match the map and the remaining particles are positioned in front of the robot, indicating a correct localization. In contrast, with an offset of 90° and 180°, the correct robot position could not be determined even after a third reference rotation. The estimated position was adjusted for both scenarios and the scattered particles were redistributed, but the positioning of the particles suggests a large uncertainty in the AMCL.



(a) Offset: 45° (b) Offset: 90° (c) Offset: 180°

Figure 6.11: Results of AMCL with Correct Position and Shifted Orientation after Reference Rotation

Since the first two referencing phases did not lead to success at an offset of 90° and 180°, the mobile robot was navigated along the wall in the virtual world. This process must be performed until the localization algorithm cannot calculate a possible position with sufficient probability and performs an entire resampling within the virtual world, which can be seen in figure 6.12. This procedure is the third phase in the referencing process and, after a final reference rotation, resulted in success for all scenarios, as demonstrated in Figure 6.13.

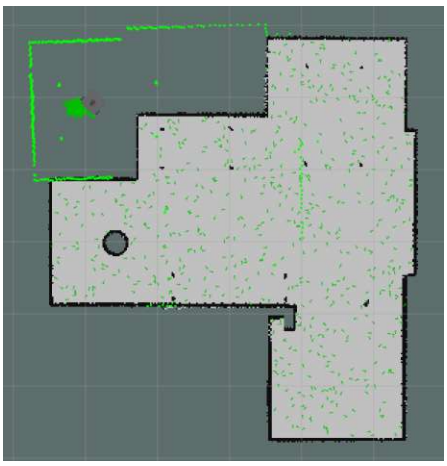


Figure 6.12: Localization after Reference Drive

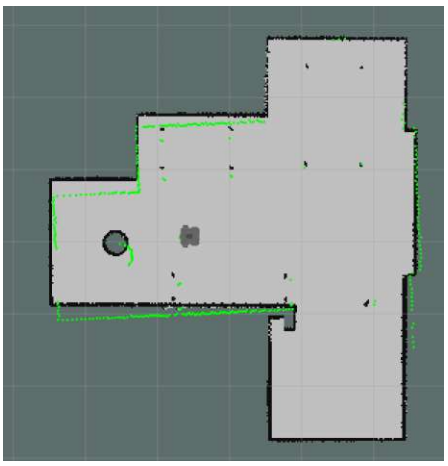
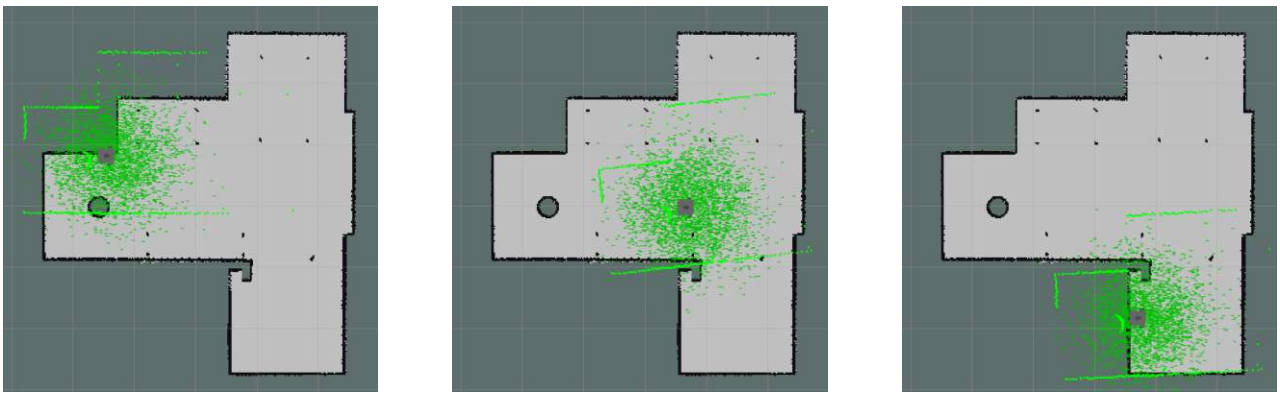


Figure 6.13: Localization after entire Resampling

6.3.3 Result of Localization with shifted Position and correct Orientation

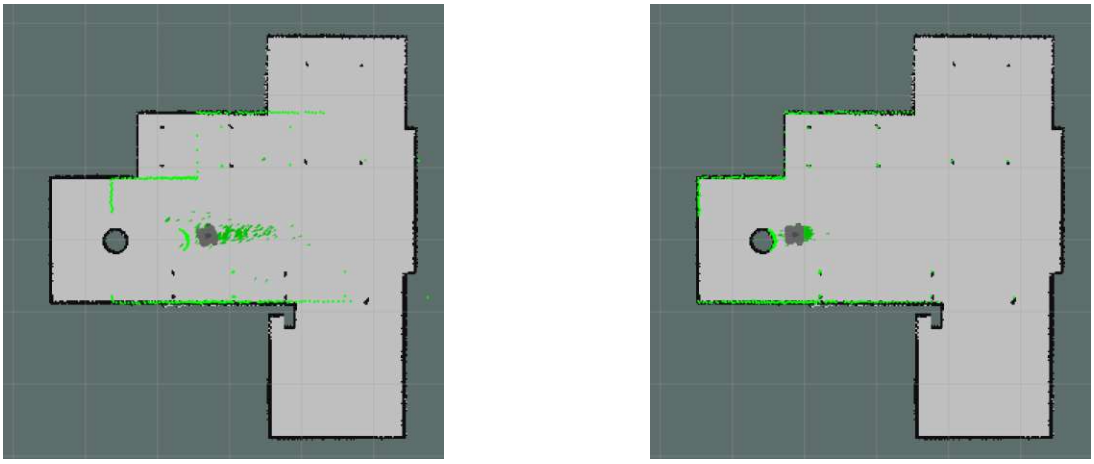
Compared to the previous scenarios, in these the orientation of the robot is not changed, while the position is shifted by 1m, 3m, and 5m, as shown in the figure 6.14.



(a) Offset: 1m (b) Offset: 3m (c) Offset: 5m

Figure 6.14: Pose Estimation - Shifted Position and Correct Orientation

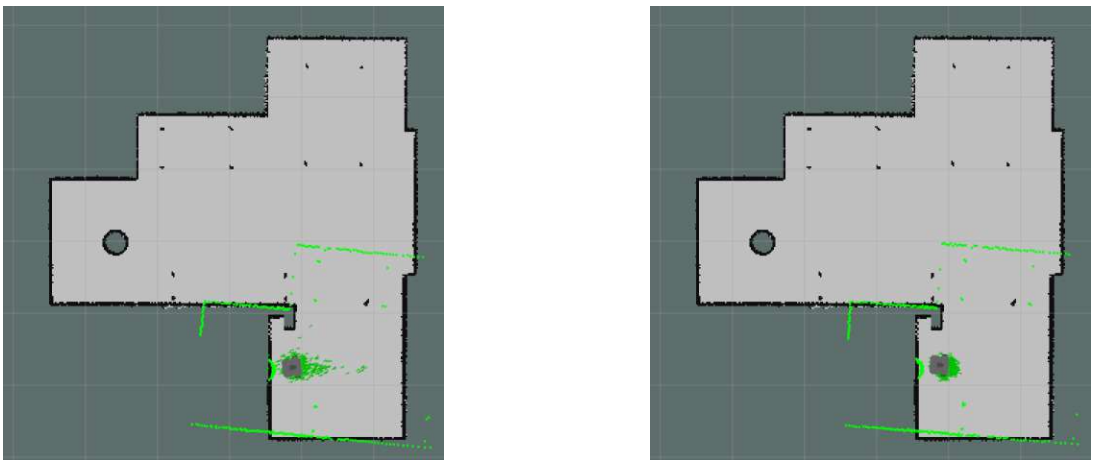
The offset of the robot position by 1m could be corrected properly by AMCL already after the first referencing phase, while for an offset of 3m further reference rotations were necessary, as can be seen in figure 6.15. After the first rotation, the position was already corrected a little bit in the right direction, but you can see based on the particles that there was still an inaccuracy. Only after performing the second referencing phase, the measuring points agree with the map as well as the particles agree with the robot position.



(a) Localization after first Reference Rotation (b) Localization after third Reference Rotation

Figure 6.15: Results of AMCL with Correct Orientation and a Position Offset of 3m

The figure 6.16 shows the results of the AMCL at an offset of 5m. It can be observed that even after performing the second reference phase, the position remained unchanged, only the particles focused more on the wrong robot position. This indicates that the AMCL algorithm incorrectly assumes with sufficient probability that the estimated position corresponds to the actual one. The reason for this is that some measurement points from the LDS correlate with the walls of the map as well as with the objects. Only after running the third reference phase and the resulting resampling, a correct localization could be performed.

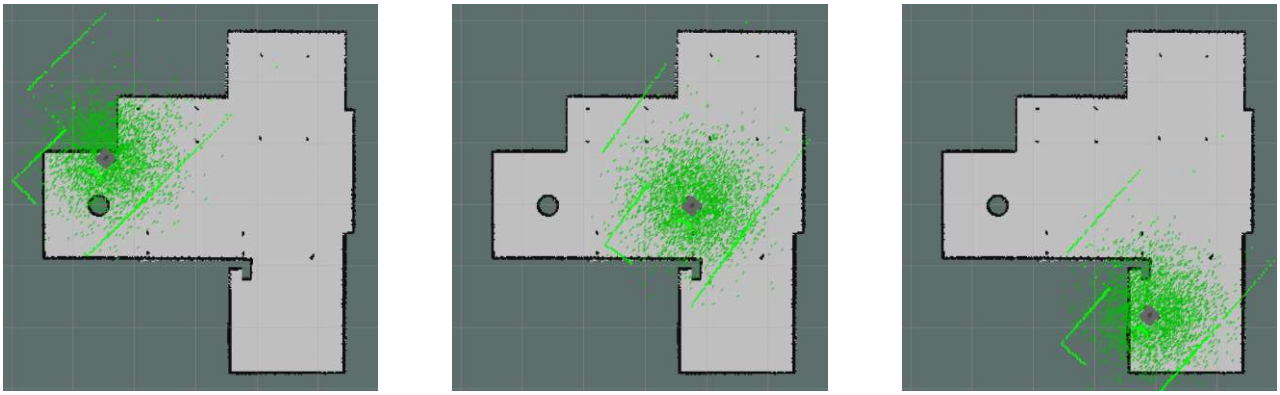


(a) Localization after first Reference Rotation (b) Localization after third Reference Rotation

Figure 6.16: Results of AMCL with Correct Orientation and a Position Offset of 5m

6.3.4 Result of Localization with shifted Position and shifted Orientation

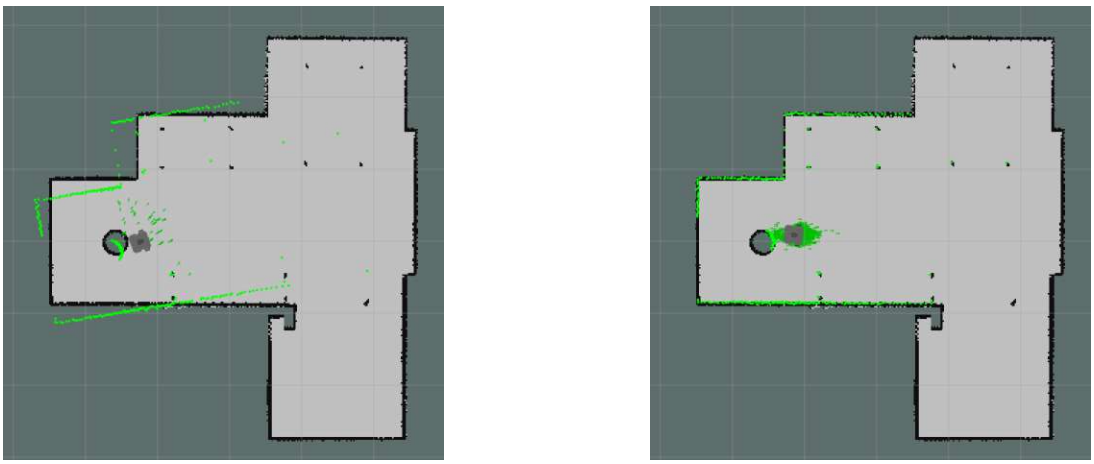
These scenarios evaluate the AMCL at an offset of the position and the orientation and therefore mainly represent initial situations that can arise from medium to strong collisions. As can be seen in the figure 6.17, only an orientation offset of 45° is performed here, since localization already failed with larger angles in the first scenario.



(a) Offset: 1m and 45° (b) Offset: 3m and 45° (c) Offset: 5m and 45°

Figure 6.17: Pose Estimation - Shifted Position and Shifted Orientation

Figure 6.18 shows the successful localization at an offset of 1m and 45° after the second referencing phase. Already after the first reference rotation a good approximation to the actual robot position has been achieved, but it is obvious from the distribution of the particles that the inaccuracy of AMCL was still high.

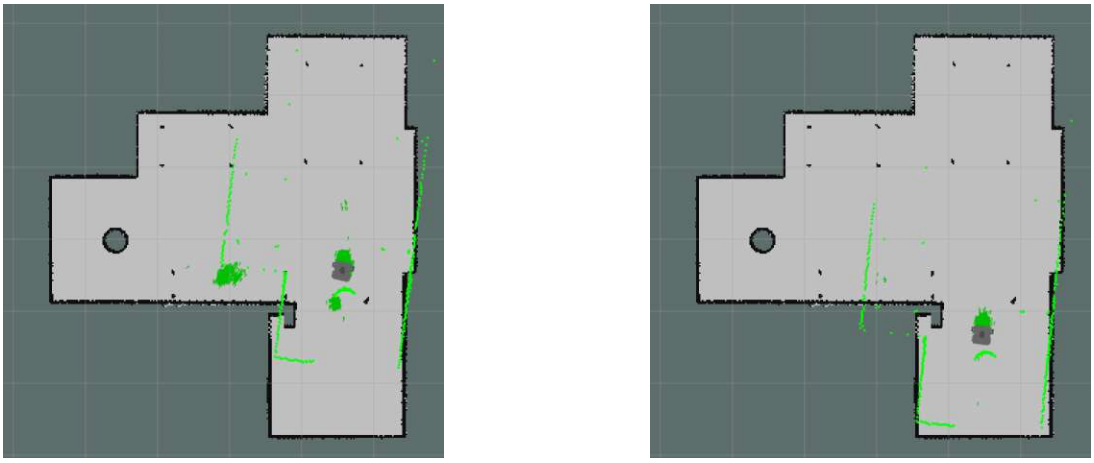


(a) Localization after first Reference Rotation

(b) Localization after third Reference Rotation

Figure 6.18: Results of AMCL with Position and Orientation Offset of 1m and 45°

The localization was much more problematic with an offset of 3m or 5m and 45°, as can be seen in figure 6.19. For both scenarios, it can be seen that the estimated orientation was corrected in the opposite direction. The reason for this is a combination of the initial position with the offset of 45° and the resulting correspondence of the measurement points with the walls of the map. However, what is most interesting is the placement of the particles, which differ in the images. While with an offset of 3m the particles were distributed on three possible positions, with the offset of 5m all were limited to one position, from which it is to be concluded that the measurements of the LDS had here clearly more agreements, even if these were not correct. Nevertheless, in both scenarios, the third reference phase had to be performed to achieve successful localization.



(a) Localization after third Reference Rotation
Offset: 3m and 45°

(b) Localization after third Reference Rotation
Offset: 5m and 45°

Figure 6.19: Results of AMCL with Position and Orientation Offset of 3m|5m and 45°

Chapter 7

Analysis of Robot Vacuum Cleaner

In this chapter, autonomous localization is tested and analyzed for a commercial vacuum cleaning robot with laser navigation through different scenarios.

7.1 Scenario Specification for autonomous Localization

The Roborock S5 Max is a vacuum cleaning robot that also uses SLAM in combination with a LDS to create a map of its environment, and on the other hand, it has an autonomous localization feature that is executed when the kidnapped-robot problem occurs. Another function, which is executed via the manufacturer's app and which is essential for autonomous localization, is the definition of targets to be approached by the robot. To simulate the kidnapping problem, the robot must be lifted and set to a new position. Only by defining a new target in the app, localization is performed by a 360° rotation and then an attempt is made to reach the desired target. This procedure allowed to test the autonomous localization of the robot vacuum cleaner under different scenarios.
Since the solution to the kidnapped-robot problem is essential in vacuum cleaning robots, it was assumed that the algorithm is much more evolved than in the open-source navigation package from the simulation. Therefore, three scenarios were specified where the robot had to perform significantly more difficult tasks to find possible limits as well.

- **Localization with wrong pose estimation**
- **Localization with separated obstacles**
- **Localization with stacked obstacles**

7.2 Results of autonomous Localization

Even though the algorithm of autonomous localization is not known for the vacuum cleaning robot, these scenarios could be used to evaluate how reliably it works and where the limitations are.

7.2.1 Results of Localization with wrong Pose estimation

To achieve the largest possible distance between the last known position and the actual one, the vacuum cleaning robot was positioned as shown in the figure 7.1. The next figure 7.2 shows the desired target, which was defined via the app and which should be approached by the robot after its localization.

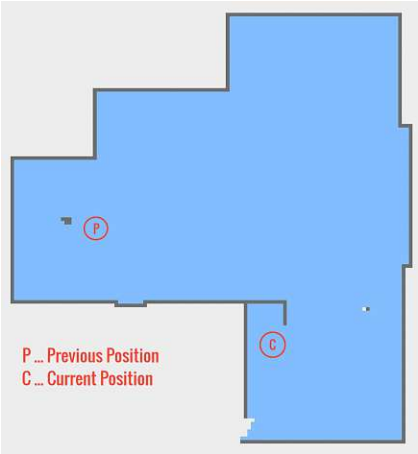


Figure 7.1: Robot Vacuum Cleaner - Previous and Current Position

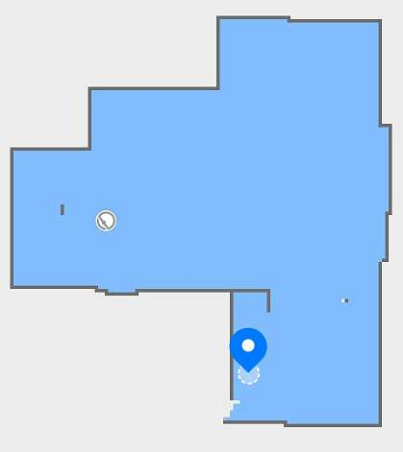


Figure 7.2: Robot Vacuum Cleaner - Defining of new Target

By turning on the robot, the kidnapping problem was detected and the autonomous localization with 360° rotation was performed. The figure 7.3 shows that the localization was successful, as the position in the app map was adjusted, and that the desired position was reached.

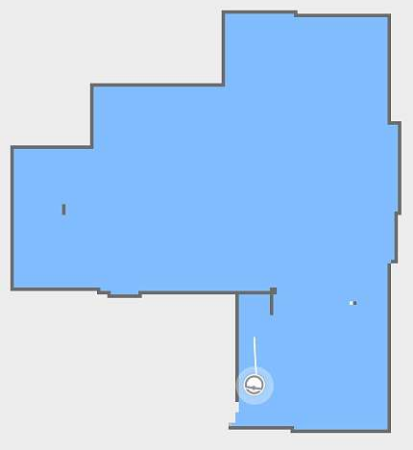


Figure 7.3: Robot Vacuum Cleaner - Successful Localization and reached Target

7.2.2 Results of Localization with separated Obstacles

If new objects are placed in an environment that was not scanned when the map was created, it increases the error rate for localization algorithms. The reason for this is that when these objects are scanned during referencing, the same structure is also searched for within the map, but it does not exist. Therefore, it is the task of the localization algorithm to determine which objects have been newly added to exclude them from the calculations during localization.
In this scenario, two boxes were placed separately around the robot, as shown in figure 7.4, while the robot was placed again in the same position as in the previous scenario. Thus, the initial situation was the same, only with the addition of two new obstacles.

Figure 7.4: Two obstacles separated from each other

This scenario also resulted in a successful localization and subsequent achievement of the desired position, as already shown in the previous figure 7.3.

7.2.3 Results of Localization with stacked Obstacles

In this scenario, the same boxes were used as obstacles, but this time they were placed in a closed corner, as shown in figure 7.5. Since a right angle is a common structure in an ordinary room, this again increases the difficulty for the localization algorithm.

Figure 7.5: Two obstacles stacked together

If the vacuum cleaning robot is not able to perform a localization because no suitable position could be found, then an acoustic signal is emitted by the robot as an error message before switching to the error state. However, the figure 7.6 shows that the vacuum cleaning robot assumed an incorrect position to be the correct one, which caused a new positioning to be performed without subsequently switching to the error state. The reason for this is that the two added obstacles at a 90° angle led to confusion in the algorithm, which finally resulted in a wrong localization. Since the vacuum cleaning robot assumed that it had found the correct position, it attempted to approach the desired target, but this was not possible.

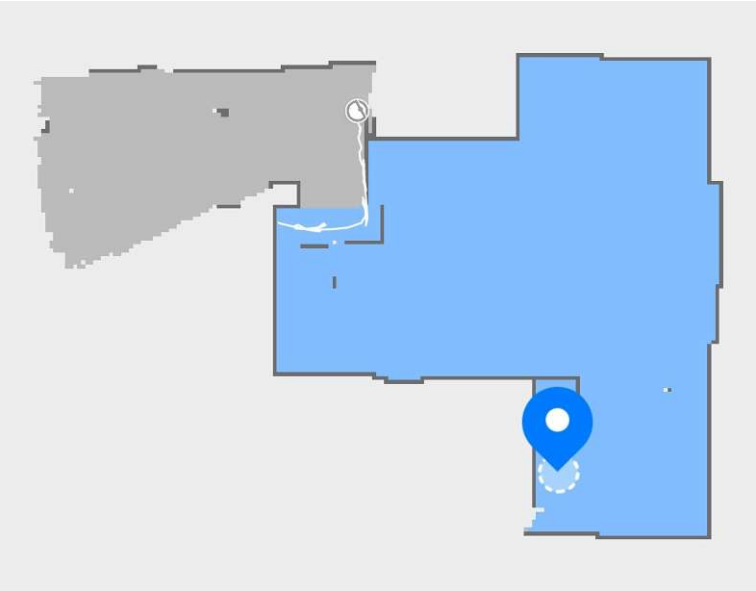


Figure 7.6: Wrong location with two stacked obstacles

Chapter 8

Discussion on VSLAM

This Chapter is describing the opportunities and constraints of Visual SLAM and offers a perspective on all possible applications with this technology.

8.1 Opportunities with VSLAM

8.1.1 Implementation without additional Sensors

A major advantage of VSLAM is that all the required data can be obtained from a single camera. The current position is calculated from the recorded images using a filter, which makes odometry unnecessary.
That means that every system with a camera and enough computational resources could be used for the implementation. This leads to much lower costs for the hardware of the robot, but the programming effort is significantly higher compared with systems using rang detection sensors instead.

8.1.2 Recognition of Objects

All detected feature points are saved in a matrix. This allows the algorithm to recognize an object, which was already scanned before.
Roomba 974, the robot vacuum cleaner which is using VSLAM, has implemented this functionality to optimize the mapping of the environment. If an already detected object was scanned again, but at a different position, the algorithm would specify it as a moving object and delete it from the map.
The following scenario was tested with the robot vacuum cleaners with LDS and camera navigation.
A chair was placed as an obstacle at first in the living room in front of the starting position. The robots started the cleaning process and scanned the object from all sides. After this task was successfully completed, the chair was moved to the kitchen room, where it was scanned again at the very end.
The map of the vacuum cleaner without VSLAM just deleted the chair from the living room and added it in the kitchen. Roomba 974 was able to recognize the object and delete it from both places.
This functionality was quite useful because the cleaning process was often interrupted by a dog. The algorithm was smart enough to mark him as a moving obstacle and to avoid collisions, by waiting some time in front of him until he moves.

8.1.3 Integration of Machine Learning

The performance of VSLAM depends mostly on the algorithm itself. This means that the accuracy and quality of the SLAM functionality can be increased if the robot learns more about the environment and how to detect the objects in it. Machine learning can be used to reach exactly that goal. It allows the robot to study the environment with all objects in it and to learn how to avoid difficult situations, like moving obstacles.
The Turtlebot package provides a machine learning component, which is based on reinforcement learning. This algorithm learns with every iteration, how to reach the goal faster and how to avoid collisions.
This is just one possible approach for using machine learning to improve VSLAM. More possible use cases are mentioned in section 3.

8.1.4 Mapping 3D Environments

LDS sensors are just able to create 2D maps because the distance sensor is measuring just at a specific height. With a camera, it is possible to scan the entire environment from many different perspectives and to create with this range of data a detailed 3D map.
This functionality is necessary for many applications of mobile robots, which can move in 3-dimensional space or just need more information about the environment.

8.2 Applications with VSLAM

Visual SLAM is a solution to integrate intelligent environmental perception cost-effectively in robots. It is already used in many applications to solve complex tasks, but the evolution of VSLAM has just started.
The following applications are just a small selection of the current possibilities with this technology.

8.2.1 AR Navigation and Mapping

Augmented Reality is becoming an important technology for many branches.
VSLAM is needed to represent the physical world precisely in the virtual environment and to allow the user to move in it.
It is often used to visualize indoor environments like a house or a flat. Together with Visual SLAM, it enables now the opportunity to move through every room and inspect everything from any perspective.
This solution has the potential to revolutionize the traditional way of flat viewing, and also take virtual sightseeing to a new level.

8.2.2 Autonomous Tasks for Drones

Autonomous flying drones are very difficult to develop because several challenging tasks have to be handled at the same time.

They are often used at places with bad GPS connectivity, which makes reliable satellite navigation nearly impossible. Another problem is, that they have to avoid collisions at different heights, like a tree or buildings. The biggest difference between indoor applications, like vacuum cleaners is, that they can use the map to learn from the environment and their obstacles. The environment of the drone is usually outdoor and changes all the time.

VSLAM is the perfect navigation and localization method for drones because it is solving all mentioned problems at once and most of them are already using high-definition cameras, which can be used as input for the algorithm.

Autonomous drones are used for example to examine vineyards for diseases or to search for disappeared persons with thermal sensors.

8.2.3 Mars Rover

The Mars Rover was sent to Mars to explore geological conditions and to search for possible signs of water.

The surface of this planet contains a lot of craters and several hills, which makes it very difficult for a robot to drive autonomously without getting stuck.

VSLAM was used for the Mars Rover because this algorithm does not require odometry data. This means, that the robot can create a detailed map and locate itself even if the wheels are drifting on that surface.

The second advantage was, that in combination with machine learning, the robot can find solutions to avoid difficult situations or obstacles.

8.2.4 Digital Inspection

Robots are commonly used to execute inspection tasks to increase efficiency and quality. With the evolution of mobile robots, it was possible to reach difficult and dangerous environments, which reduces safety risks for humans.

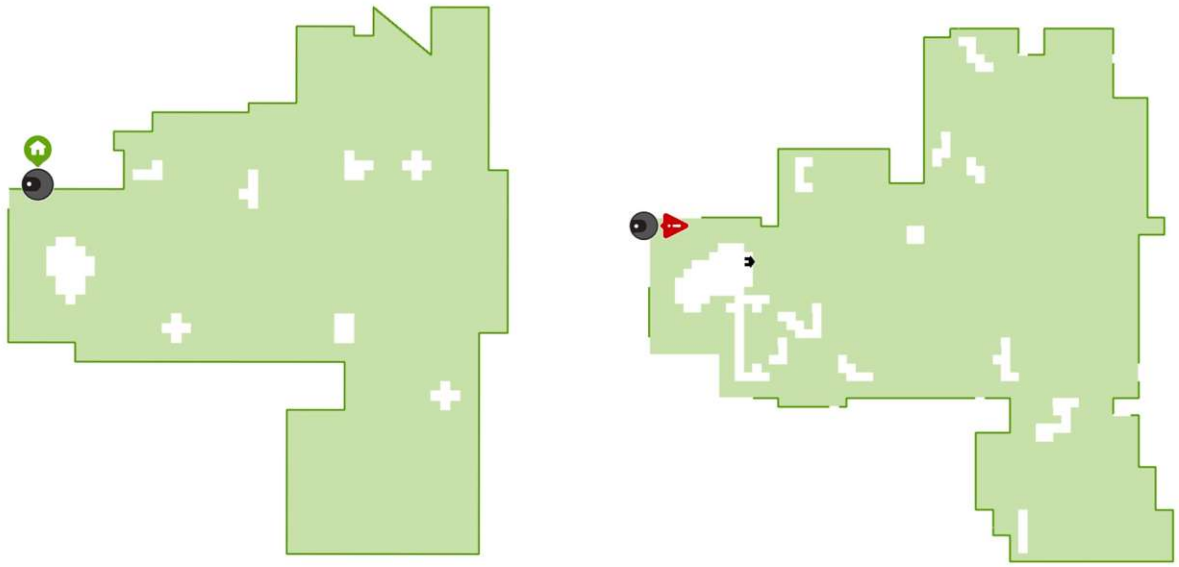
Together with Visual SLAM, the mobile robots were able to execute safety-relevant tasks faster and more accurately than humans would be able to. Therefore, they were used in critical sections, like gas and oil production, to find cracks and other damages on pipelines. Such robots work very efficiently, are cost-effective, and can be used in every environment, even if it is underwater.

8.3 Constraints of VSLAM

8.3.1 Lighting Conditions

It is well known that the quality of an image is influenced by the incidence and intensity of light. To test this scenario, Roomba 974 was used to clean the same environment again, but in the dark.

The figures 8.1 are showing the results of the created maps. The robot was able to create straight walls, but especially in the kitchen, he had some problems with detecting a loop closure. Compared to the valid map, there are also many more unknown places, which are defined by white rectangles.



(a) Map created under normal lighting conditions

(b) Map created in the dark

Figure 8.1: Mapping with different lighting conditions

8.3.2 Complex System

VSLAM reduces the hardware costs compared to other SLAM systems significantly, because no additional sensors for range detection and odometry are necessary, but the programming effort for developers is much higher.
VSLAM methods are capturing a lot of pictures, to be able to calculate the current position and create a precise map. These algorithms are very complex and need to be customized for every application because the system, for example a drone and a mobile ground robot differs a lot.

8.3.3 Computational Power and Memory Space

As described in the previous section 8.3.2 the computational effort is much higher than for normal SLAM methods. Another challenge is also to handle and store the large number of pictures, which are taken by the camera. To be able to create precise 3D maps of the environment and to react in real-time to obstacles a powerful controller with large memory space is needed.

8.3.4 Creation of a 2D Map

Algorithms such as ORB-SLAM are using sparse interest points for mapping and tracking. With these points, it is possible to create a 3D map of the environment. The problem with VSLAM methods is, that they are not able to create a 2D map with these sparse points. It would be necessary to adjust the BoW vocabulary to all given structures in the environment and to define some parameters to learn the mobile robot which objects are in his collision range.
Another solution is described in [33], where a program is used to convert a 3D sparse point environment at a specific height to a 2D occupancy grid map. This means, that all objects above and beyond this height are not going to be included in the map.

8.4 Conclusion

Visual SLAM will become an essential technology for many applications and will replace many robots with different systems. The reason is, that the hardware costs are very low and the algorithm can work as accurately as the other SLAM methods.

This technology will allow mobile robots of all kinds to move or fly autonomously through a difficult environment and execute successfully complex tasks. Section 8.2 describes current applications that would not be feasible without VSLAM. It is used in environments, which are too dangerous for humans or where critical tasks must be executed.

The quality and performance of VSLAM depend mostly on the algorithm itself, which still needs to be improved significantly to achieve a sufficient technological standard. It will be often necessary to use a camera with a higher resolution and this will lead to much more calculated key points. On the one hand, it will be necessary to store a large number of images and data and, on the other hand, to provide sufficient power resources to perform these calculations.

Chapter 9

Conclusion

The results in Chapter 6 of this thesis show that all three SLAM methods were able to produce maps with few pixel errors using the default parameter set, but Gmapping was the most robust and reliable approach. However, in the test scenarios with different standard deviations, two interesting facts were observed. The first was that Hector-SLAM could no longer produce a usable map with a standard deviation of 0.03m. The reason is that this SLAM method does not use odometry data, which means that the distance is measured exclusively with the LDS, but this leads to problems in localizing the robot at higher measurement inaccuracies. Furthermore, the results showed us that Gmapping-SLAM recorded only small losses up to a deviation of 0.05m, but from 0.06m to 0.2m the error rate increased significantly, while Karto-SLAM could still generate a usable map up to a standard deviation of 0.1m. When varying the parameter Samples/Revolution, the SLAM approach Hector again failed to generate a map at a resolution lower than 1°, which corresponds to 360 measurement points at one rotation. The last tested parameter of the LDS was the update rate, which was changed from 1Hz to 500Hz. This parameter did not lead to any significant changes in the generated maps for all three SLAM approaches, because the mobile robot can only move at a low speed, so the update rate does not greatly affect the accuracy of the measurement. The evaluation of SLAM methods, therefore, led to the conclusion that Gmapping and Karto are very robust and reliable SLAM approaches, which are providing good results for high-performance LDS and as well for those for standard applications. In contrast to the results in [20], where Karto-SLAM achieved better map quality than Gmapping-SLAM, the results of this thesis showed that the changed characteristics of the LDS and the higher uncertainties lead to a lower quality with Karto-SLAM. The results of [27] are consistent with those from this work, as Gmapping is the most stable SLAM approach when exploring unknown environments, and the Hector generates large inaccuracies in the map due to the absence of odometry data at even low noise levels.

In this work, it was not possible to generate usable maps with the VSLAM approaches. The reason for this was that implementation of the methods in the open-source simulation turned out to be difficult and, in the case of the mono-SLAM approach, infeasible. Moreover, with the available open-source package, it was not possible to convert a 3D map with sparse points into a 2D map. An approach to solve the conversion problem is described in [33], which was able to create a 2D occupancy grid map using a VSLAM approach with a monocular camera.

The results of the scenarios for localization with AMCL showed that the simulation had major problems as soon as the robot's orientation was shifted by more than 45°. If the position was changed by up to 3 meters, a successful localization with a reference rotation could already be performed. If, on the other hand, the orientation was changed by more than

45°, or the position was shifted by 5 meters, even multiple reference rotations could not determine the correct position. As a result, the robot had to be navigated through the environment until the calculated probability for the correct position was low enough that a complete resampling within the map was performed. By executing another reference rotation, the AMCL algorithm was always able to find the correct position. The evaluation of AMCL showed that the fundamental problems with autonomous localization arise after the kidnapped-robot problem occurs, as described in [5] and [4], but with optimal parameters adjusted to the environment, localization was still always successful.

Autonomous localization was also performed with a vacuum cleaning robot, which also uses laser navigation. Due to the assumption that the manufacturers of vacuum cleaning robots have developed a significantly more effective localization algorithm than the open-source AMCL method from the Turtlebot3 package, localization was performed under more difficult conditions. The results show that localization within a known environment was always successful, regardless of position and orientation. Even when new objects were afterward placed in the environment, which is not represented on the map, it was possible to determine the correct position of the robot without any problems. Only when two obstacles were placed in front of the robot at a 90° angle was it not possible for the vacuum cleaning robot to find the correct position. The reason for this is that structures with this angle occur very often in common indoor environments and therefore the probability of incorrect localization is significantly increased by additional placed objects at an angle of 90°.

Bibliography

[1] Y. Abdelrasoul, A. B. S. H. Saman, and P. Sebastian. "A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2D SLAM". In: *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*. 2016, pp. 1–6.

[2] *AMCL Parameter*. https://http://wiki.ros.org/amcl. Accessed: 2022-03-03.

[3] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós. "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM". In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1874–1890.

[4] M.-A. Chung and C.-W. Lin. "An Improved Localization of Mobile Robotic System Based on AMCL Algorithm". In: *IEEE Sensors Journal* 22.1 (2022), pp. 900–908.

[5] M.-A. Chung and C.-W. Lin. "Pose Detection of a Mobile Robot Based on LiDAR Data". In: *2021 International Conference on Fuzzy Theory and Its Applications (iFUZZY)*. 2021, pp. 1–5.

[6] F. Demim, A. Nemra, K. Louadj, Z. Mehal, M. Hamerlain, and A. Bazoula. "Simultaneous localization and mapping algorithm for unmanned ground vehicle with SVSF filter". In: *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*. 2016, pp. 155–162.

[7] C. Duan, S. Junginger, J. Huang, K. Jin, and K. Thurow. "Deep Learning for Visual SLAM in Transportation Robotics: A review". In: *Transportation Safety and Environment* 1 (Dec. 2019), pp. 177–184.

[8] J. Folkesson and J. Leonard. "Autonomy through SLAM for an Underwater Robot". In: *Autonomy through SLAM for an Underwater Robot*. 2011.

[9] D. Fox. "KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localization". In: (Oct. 2001).

[10] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots". In: Jan. 1999, pp. 343–349.

[11] G. Grisetti, C. Stachniss, and W. Burgard. "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 2432–2437.

[12] G. Grisetti, C. Stachniss, and W. Burgard. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters". In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46.

[13] S. Habibi. "The Smooth Variable Structure Filter". In: *Proceedings of the IEEE* 95.5 (2007), pp. 1026–1059.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[15] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. "A Flexible and Scalable SLAM System with Full 3D Motion Estimation". In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. Nov. 2011.

[16] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent. "Efficient Sparse Pose Adjustment for 2D mapping". In: Oct. 2010, pp. 22–29.

[17] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent. "Efficient Sparse Pose Adjustment for 2D mapping". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 22–29.

[18] M. Labbé and F. Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation: LABBÉ and MICHAUD". In: *Journal of Field Robotics* 36 (Oct. 2018).

[19] *LDS-01 Specification*. https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/. Accessed: 2022-03-03.

[20] X. Le, L. Fabresse, N. Bouraqadi, and G. Lozenguez. "Evaluation of Out-of-the-Box ROS 2D SLAMs for Autonomous Exploration of Unknown Indoor Environments: 11th International Conference, ICIRA 2018, Newcastle, NSW, Australia, August 9–11, 2018, Proceedings, Part II". In: Aug. 2018, pp. 283–296.

[21] T. Lee, B. Chen, T. Hsu, B. Huang, W. Lin, and C. Kuo. "A remote-controlled robot-car in the TPS tunnel". In: *Journal of Physics: Conference Series* 1350 (Nov. 2019), p. 012147.

[22] R. Li, S. Wang, and D. Gu. "Ongoing Evolution of Visual SLAM from Geometry to Deep Learning: Challenges and Opportunities". In: *Cognitive Computation* 10.6 (Dec. 2018), pp. 875–889.

[23] M. Mengelkoch. *Implementieren des FastSLAM Algorithmus zur Kartenerstellung in Echtzeit*. studythesis. 2008.

[24] A. Merzlyakov and S. Macenski. *A Comparison of Modern General-Purpose Visual SLAM Approaches*. 2021.

[25] R. Mur-Artal and J. D. Tardós. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262.

[26] Mur-Artal, Raúl, Montiel, J. M. M., Tardós, and J. D. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163.

[27] E. D. Peter Aerts. "Benchmarking of 2D-Slam Algorithms". In: ACRO RESEARCH GROUP, KU LEUVEN, DEPARTMENT OF MECHANICAL ENGINEERING, CAMPUS DIEPENBEEK, WETENSCHAPSPARK 27, 3590 DIEPENBEEK, BELGIUM. July 2017, p. 28.

[28] P. Pfaff, W. Burgard, and D. Fox. "Robust Monte-Carlo Localization Using Adaptive Likelihood Models". In: vol. 22. Jan. 2006, pp. 181–194.

[29] M. Rojas-Fernández, D. Mújica-Vargas, M. Matuz-Cruz, and D. López-Borreguero. "Performance comparison of 2D SLAM techniques available in ROS using a differential drive robot". In: *2018 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. 2018, pp. 50–58.

[30] *ROS Wrapper*. https://github.com/raulmur/ORB_SLAM2. Accessed: 2022-31-03.

[31] S. J. (J. Russell. *Artificial intelligence : a modern approach*. Includes bibliographical references (pages 1063-1093) and index. Third edition. Upper Saddle River, N.J. : Prentice Hall, [2010] ©2010, [2010].

[32] L. Russo, S. Rosa, B. Bona, and M. Matteucci. "A ROS Implementation of the Mono-Slam Algorithm". In: vol. 4. Jan. 2014, pp. 339–351.

[33] A. M. Santana, K. R. Aires, R. M. Veras, and A. A. Medeiros. "An Approach for 2D Visual Occupancy Grid Map Using Monocular Vision". In: *Electronic Notes in Theoretical Computer Science* 281 (2011). Proceedings of the 2011 Latin American Conference in Informatics (CLEI), pp. 175–191.

[34] R. Sedgewick and K. Wayne. *Algorithms, 4th Edition.* Addison-Wesley, 2011, pp. I–XII, 1–955.

[35] S. Sumikura, M. Shibuya, and K. Sakurada. "OpenVSLAM: A Versatile Visual SLAM Framework". In: *Proceedings of the 27th ACM International Conference on Multimedia* (2019).

[36] K.-S. Wang and C.-K. Huang. "Intelligent Robotic Lawn Mower Design". In: *2018 International Conference on System Science and Engineering (ICSSE).* 2018, pp. 1–5.

[37] H. Zhang, N. Chen, and G. Fan. "An Improved Localization Algorithm for Intelligent Robot". In: *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI).* 2019, pp. 1–5.

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe d er Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Vienna, Austria, 2.April 2022

Milanovic Sinisa