TU WIEN Informatics

Frontier-based Autonomous Exploration with a Quadruped Robot in 2D

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Victor Mittermair
Matrikelnummer 11809916

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Hannes Kaufmann
Mitwirkung: Projektass. Dr. Franceso De Pace
Univ.Ass. Dipl.-Ing. Soroosh Mortezapoor

Wien, 22. September 2022

Victor Mittermair | Hannes Kaufmann

Erklärung zur Verfassung der Arbeit

Victor Mittermair

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. September 2022

Victor Mittermair

Victor Mittermair

Acknowledgements

I would like to express my gratitude to Hannes Kaufmann, Franceso De Pace and Soorosh Mortezapoor for providing guidance throughout the whole project. In the early stages of this project, Soroosh Mortezapoor and Franceso De Pace helped me tremendously by explaining key aspects of navigation with ROS, thus making it much simpler to get a grasp of the navigation stack. Furthermore, thank you to Franceso De Pace for helping me with the real-world experiment by providing support through the whole process.

Most importantly, without the assistance of Hannes Kaufmann from the beginning to the end of this project, none of this could have happened. Thank you.

Abstract

Autonomous exploration is an important topic in robotics, as the uses cases of an autonomous mobile robot are rapidly expanding. This project uses the quadruped robot Spot from Boston Dynamics for the autonomous exploration. Spot is already capable of navigating through prerecorded paths automatically, even in 3D, but autonomous exploration without human guidance is still not available. Thus, a fully autonomous exploration system is proposed, built with the robotics framework ROS and Google's Cartographer SLAM algorithm. To investigate the aptitude of the system, a real-world experiment was conducted with two different scenarios. In scenario 1, Spot explores an empty space, in scenario 2 an obstacle covers part of the test site. For evaluating the results of the exploration, the maps built with SLAM are compared with ground truth maps, which were constructed by an accurate 3D model of the test site. To quantify the error between the built and ground truth map, an error based on the k nearest neighbour algorithm (NNE), the Structure Similarity Index Measure (SSIM) and the map coverage were used. In both scenarios, Spot completed the exploration with no frontiers left to explore. The quality and the coverage of the map were acceptable in both cases, especially in the first scenario with a map coverage of 99% and almost no NNE.

Contents

CHAPTER 1

Introduction

One of the most important problems of robotics is autonomous navigation. The ability of a mobile robot to explore its environment without human guidance is highly sought after in many different use cases such as search and rescue [1] [2], factory inspection [3], or surveillance [4].

In search and rescue applications robots must navigate through unknown environments, making the autonomous navigation task an exploration task: the robot has to simultaneous map the environment, locate itself on the map and calculate collision-free paths to the next unknown area. The combination of the first two subtasks is called the SLAM (Simultaneous Localization and Mapping) problem, which is an essential and highly researched topic in robotics [5] [6], because for the robot to navigate through its surroundings it first must know the environment and where the robot's position in this environment is. After localizing the robot and the obstacles, a collision-free path to a goal can be computed. The goal is the nearest area where parts of the map are missing, making the robot explore the environment until no unknown areas are available. This type of autonomous exploration will be implemented in this project using the robotics framework ROS [7] and the quadrupedal mobile robot Spot from Boston Dynamics [8], which is based on the MIT Cheetah robot model [9].

Spot already provides features of autonomous navigation, but it still needs a human intervention. Specifically, the human operator can teach Spot a new path by manually teleoperating it. During the teleoperation, the path is recorded, allowing Spot to play it autonomously [10]. To make Spot fully autonomou, a frontier-based approach for autonomous exploration is implemented via ROS. The focus of this project is on 2D indoor environments with LiDAR-based SLAM via Google's Cartographer [11].

CHAPTER 2

Related Work

Yamauchi Brian first proposed the frontier-based approach for autonomous exploration [12]. In his paper, he used a mobile robot fitted with sixteen sonar sensors, sixteen infrared sensors and one laser rangefinder. The combination of the sonar sensors with the laser rangefinder enabled better quality in the occupancy grid, as the main problem with sonar sensors is reflections of the sound pulse that could feed wrong range data. This was mitigated by the laser-limited sonar system by limiting the maximum range of the sonar range data by the range data of the laser rangefinder. If the range computed by the sonar is larger than the range of the laser the sonar range reading is ignored and the range reading from the laser is taken instead.

With these onboard sensors, an occupancy grid can be constructed. This is done by mapping the range data onto a cartesian grid. This grid consists of cells, each with a probability of occupancy. At initialization, each cell has a prior probability of occupancy and after every new sensor reading the probability of occupancy updates accordingly.

Now that an occupancy grid is constructed with the onboard sensors, the next step of the frontier-based exploration is to classify the cells of the occupancy grid into three groups:

- **open**: the occupancy probability < the prior probability
- **unknown**: occupancy probability = prior probability
- **occupied**: occupancy probability > prior probability

After the classification of each cell, frontier edge cells are labeled. These are all open cells that are adjacent to unknown cells. Adjacent frontier edge cells are then grouped into frontier edge groups depending on a certain threshold (e.g., the size of the robot). If

the size of a frontier edge group is greater than the threshold, a new frontier has been found. Each centroid of the frontier is now a potential goal for the robot to move to. If the robot reaches the nearest frontier, it spins 360° to map the environment and marks the reached frontier as previously visited. If a frontier is unreachable, the frontier is marked as inaccessible and the robot attempts to move to the nearest reachable frontier. This procedure continues until all reachable frontiers are visited and no more unexplored frontiers are available. This system was tested in two real-world office environments, where the robot successfully mapped its environment autonomous.

With this approach, Yamauchi made considerable improvements to autonomous exploration techniques. The main advantages over exploration techniques at that time were that obstacles and walls could be in arbitrary orientations and that it can explore efficiently by moving to the most likely place to find new map data.

Although this greedy approach of always going to the nearest frontier might seem inefficient, meaning long trajectories for the mobile robot, Holz et al. [13] found that even with more sophisticated exploration strategies, the overall path length was not significantly shorter than the greedy closest frontier algorithm.

To map the unexplored environment, SLAM are often used for autonomous mobile robots. Yagfarov et al. [14] compared several LiDAR-based SLAM algorithms in ROS [15] and concluded that Cartographer is one of the most efficient SLAM algorithms for building 2D maps. To come to that conclusion, they used a Laser Tracker to measure the ground truth map and compared the occupancy grid obtained through the ROS SLAM algorithm. They compare Cartogrpapher with Hector [16] in four different conditions:

1. robot slow with smooth rotations
2. robot fast with smooth rotations
3. robot fast with fast rotations
4. robot slow with without loop closure

The map built by Cartographer had the least amount of error compared to the ground truth in three of the four movement scenarios. Only in the third scenario Hector SLAM achieved a greater accuracy than Cartographer.

CHAPTER 3

Design

3.1 ROS (Robot Operating System)

The Robot Operating System (ROS) has been chosen for developing the frontier-based approach, as it is a widely used framework for robotics in industry and research, providing multiple tools and already implemented algorithms for navigation and exploration.

3.1.1 Introduction to ROS

ROS [15] [7] uses a peer-to-peer network to exchange and elaborate data, forming the ROS Computation Graph. Each process in this graph is called a **Node**. A ROS system usually consists of multiple nodes working together, forming a modular system at a fine-grained scale.

Communication between nodes is handled by the **Master**. The master is responsible for name registration and lookup in the Computation Graph, enabling the nodes to communicate with each other. To know what kind of data is passed through this node-to-node communication ROS uses a strictly typed data structure called **Message**. Each message can hold multiple primitive data types (integer, floating point, boolean, etc.), arrays of primitive data types and even other messages.

There are two types of communication between nodes. Asynchronous and synchronous. The former is realized with a publish-subscribe model through **Topics**. When nodes are publishing messages, they do that to a certain topic, which is defined by a simple string. Should a node be interested in a certain type of data, they subscribe to a topic reflecting said data, thus receiving messages corresponding to the topic. The number of nodes publishing or subscribing to a certain topic is not limited.

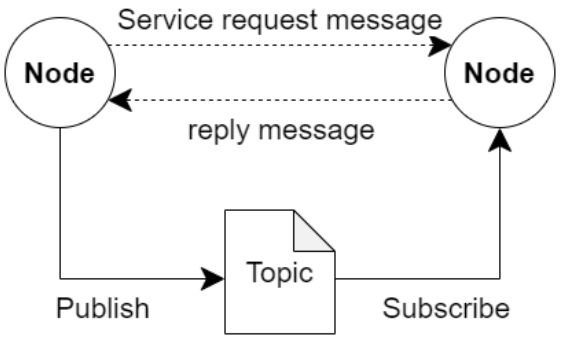


Figure 3.1: Node communications in ROS

Synchronous communication with a request/reply interaction is realized with **Services**, which are defined by two messages, one for the request and one for the reply, and the name of the service. If a node wants to invocate a service it has to send a request message to another node providing said service.

3.1.2 tf (Transform Library)

One of the most important core packages of ROS is the tf [17] library. Its main goal is to keep track of different coordinate frames and transform data within an entire system. Furthermore, individual components in the system should be able to query data within a specific coordinate frame without knowing all coordinate frames at a specific time.

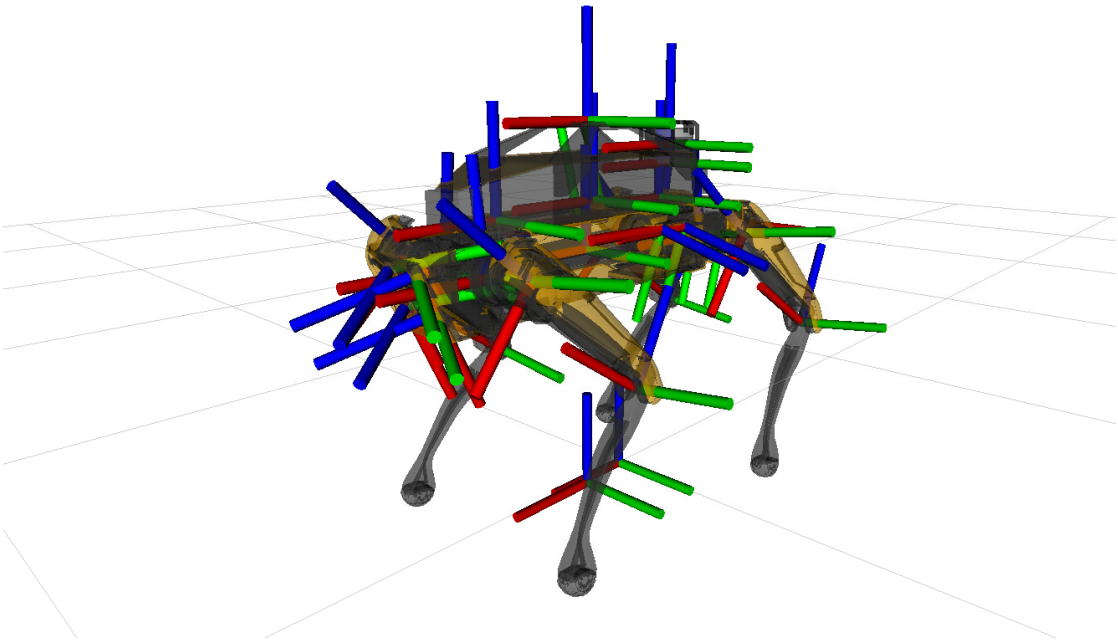


Figure 3.2: View of all tf frames of Boston Dynamics Spot. Cylinders represent the x, y and z axes of coordinate frames.

This is accomplished by two modules: The Listener and the Broadcaster. The tf broadcaster's goal is to publish coordinate frames and their transforms relative to other coordinate frames at a specific frequency or if an update is heard by the listener. On the other hand, the listener receives the transform information, providing a searchable data structure. This data structure can be described as a tree with nodes as coordinate frames and edges as transforms. These edges are directional, meaning that if one wants to move up the tree, the transform must be inversed.

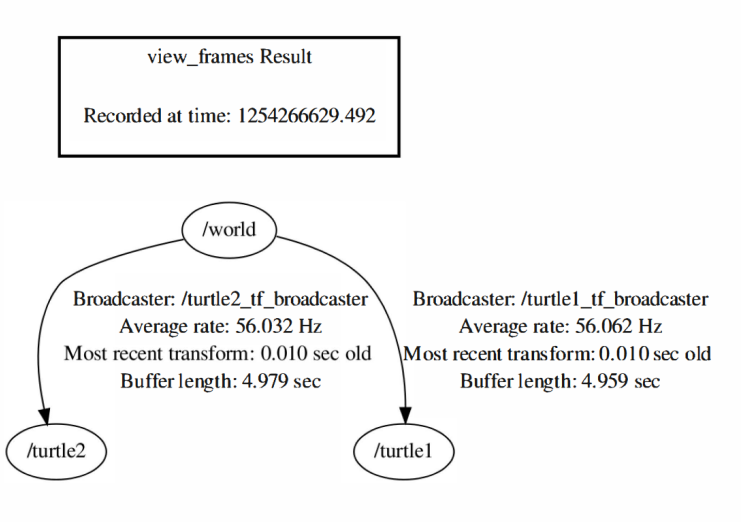


Figure 3.3: tf tree of two turtles showing debugging information [17]

For tf to work, all tf transformable data must have a **Stamp**. A stamp is comprised of the coordinate frame in which it is represented and the time in which the data is valid. Should an algorithm now want to know the transform from the coordinate frame of the stamped data to the coordinate frame relevant for the algorithm at a specific time, the tf library can be called. The listener will now try to compute the transform between the source and the target frame by forming a spanning set. This is done by walking up the tf tree edges until a common parent is found. If a common parent is found the listener computes the transformation between the source and target frame. If an edge must be traversed in the opposite direction the transform has to be inversed, otherwise the transform from one frame to the other is simply applied.

3.2 Navigation Stack

The Navigation Stack is a ROS package that includes several functionalities for navigating a mobile robot [18] [19]. This section will briefly explain the core functionalities of the navigation stack and relate it to the used methods of this project and is based on [18] [19]. Figure 3.4 is a general overview of the nodes working in a ROS system using the navigation stack.

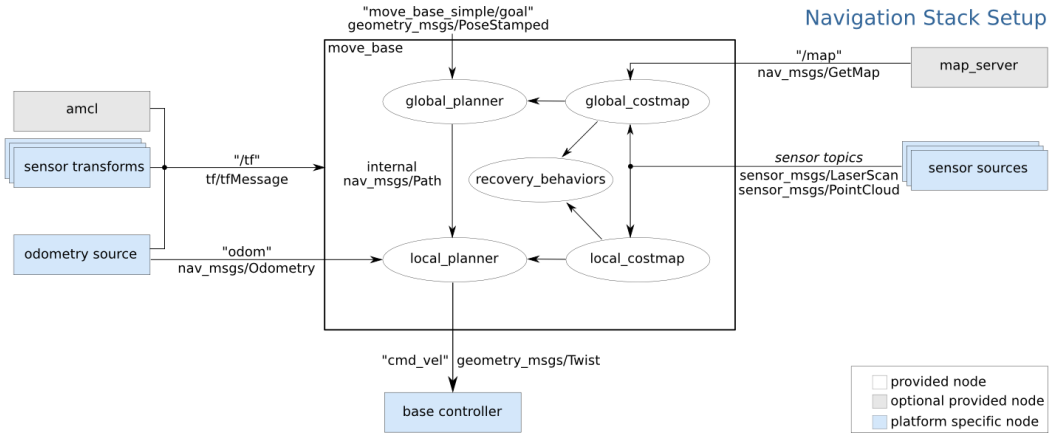


Figure 3.4: Overview of the ROS Navigation Stack [19]

3.2.1 SLAM (Simultaneous Localization and Mapping)

One of the most important tasks for autonomous mobile robots is to locate themselves within their environment. To explore unknown areas autonomously a robot has to create a map, localizing itself in it. This problem is called the SLAM (Simultaneous Localization and Mapping) problem [5]. The main advantage of SLAM is that mobile robots don't have to know predefined maps, also called static maps, to navigate through space, making it much more independent of a priori knowledge of the environment.

ROS supports multiple implementations of SLAM algorithms and this project uses Cartographer [11]. Details of the Cartographers SLAM algorithm will be discussed in Chapter 4.

3.2.2 Sensors and Costmaps

The Navigation Stack needs several sensors to function properly. As depicted in Figure 3.4, sensor sources from range finding sensors are needed to build local- and global costmaps, as the range data clears or marks obstacles in costmap [20]. These messages can either be 2D LaserScan messages or 3D PointCloud messages. Costmaps are grid-based maps that contain 2D information about obstacles in the environment. There are two different types of costmaps:

- **local costmap**: Costmap of the environment close to the robot. The costmap moves with the robot, thus making the map a scrolling window.
- **global costmap**: Costmap of the whole mapped environment.

Each cell in these maps has a certain cost associated with it, with obstacle cells having high cost and free cells having zero cost. To make sure robots don't come too close to obstacles, an inflation radius around the obstacles is added, giving cells near obstacles a higher cost than free space. The main goal now is to feed path planners the information of these costmaps so they can plan a path with the least amount of cost, thus avoiding obstacles.

3.2.3 Path planners

Like costmaps, path planners are also divided into local- and global path planners. The global planner is responsible for planning a path using the given global costmap provided by the Navigation Stack (more specifically provided by the *move_base* package of ROS). The goal of the global planner is to find the path with the least amount of cost. On the other hand, the local path planner relies only on information of the local costmap, which it extends only a few meters ahead of the robot. During the movement phase, it allows the robot to avoid unexpected obstacles. Consequently, the local planner's main job is to compute local paths with the fewest cost relative to the global path while dynamic obstacles could be inserted into the robot's global trajectory. In Figure 3.5 a global and local path planner's trajectory can be seen.

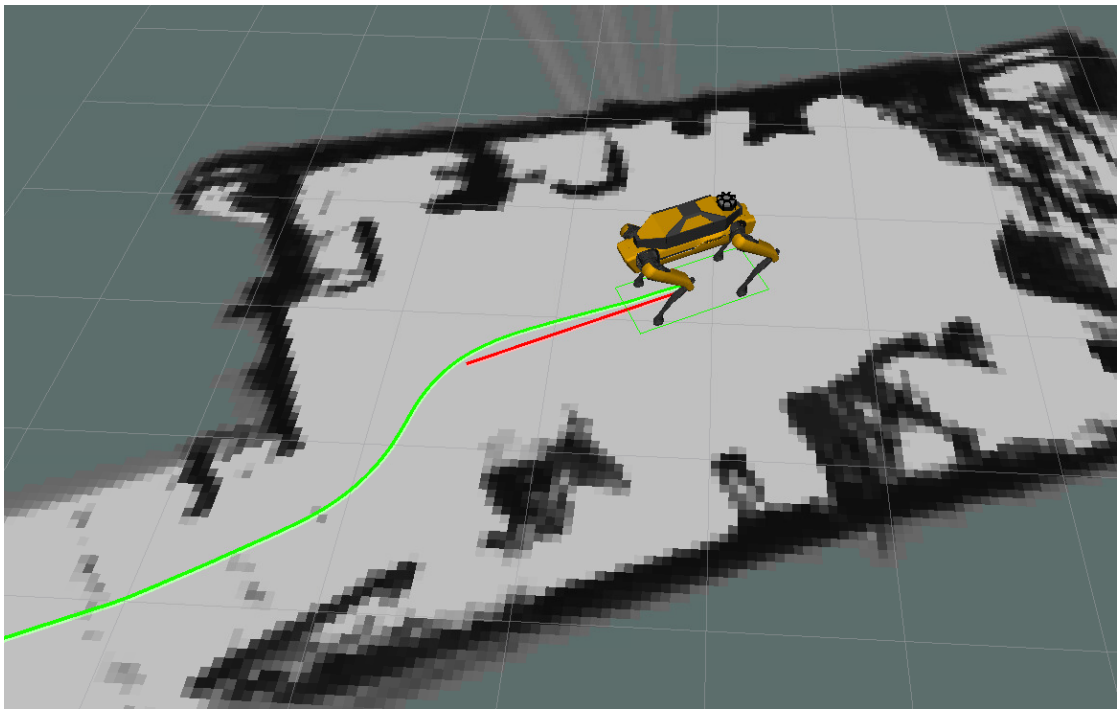


Figure 3.5: Global and local paths of the *move_base* package in ROS visualized in rviz. The local path is visualized in red, the global path in green.

3.3 System Overview

3.3.1 Hardware and Software

This project uses the Boston Dynamics Spot robot with the additional EAP (Enhanced Autonomy Payload) [21]. This additional payload consists of: (i) an additional Personal Computer (PC) (SpotCORE), mounted on top of Spot which is connected via ethernet to the internal PC of Spot, allowing for low-latency and high-bandwidth connection, and (ii) a 3D LiDAR sensor composed of 16 channels with a range of 100m [21]. Spot has the following on-board sensors [22]: black and white fisheye camera, 5 different depth-infrared cameras (range 4m): front-left, front-right, left, right, back. With these sensors, the overall field of view is 360°.

Component	**Configuration**
CPU	Intel Core i5-8365UE
RAM	16 GB DDR4 2666
Software	**Configuration**
OS	Ubuntu 18.04 LTS
ROS	Melodic Morenia

Table 3.1: Hardware and Software specification of the SpotCORE [23]

SpotCORE is used as the ROS PC which communicates directly with Spot through the ROS Wrapper developed by Clearpath Robotics [24]. Through this driver, internal data from Spot, such as odometry, range data from depth cameras, and range data from LiDAR, is fed to ROS, so nodes in the ROS Computation Graph can use the information gathered by the on-board sensors and the EAP from Spot. Figure 3.7 visualizes the used system architecture of the ROS system.

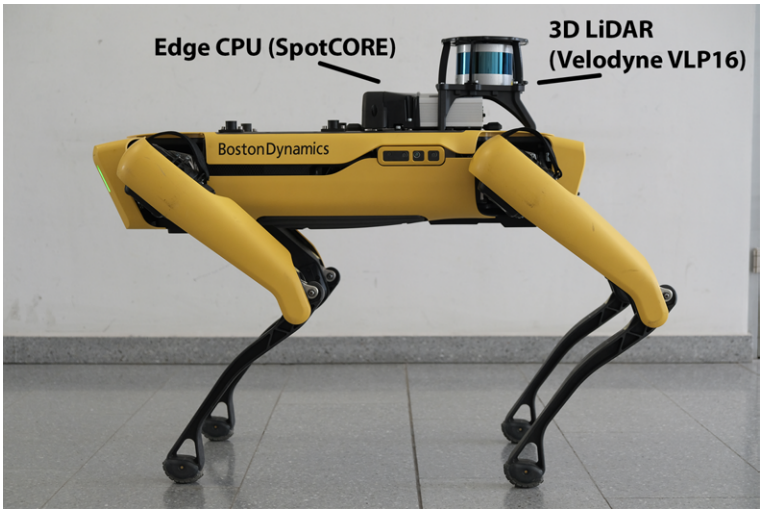


Figure 3.6: Spot with the EAP. Consists of a VLP-16 3D LiDAR and Edge CPU.

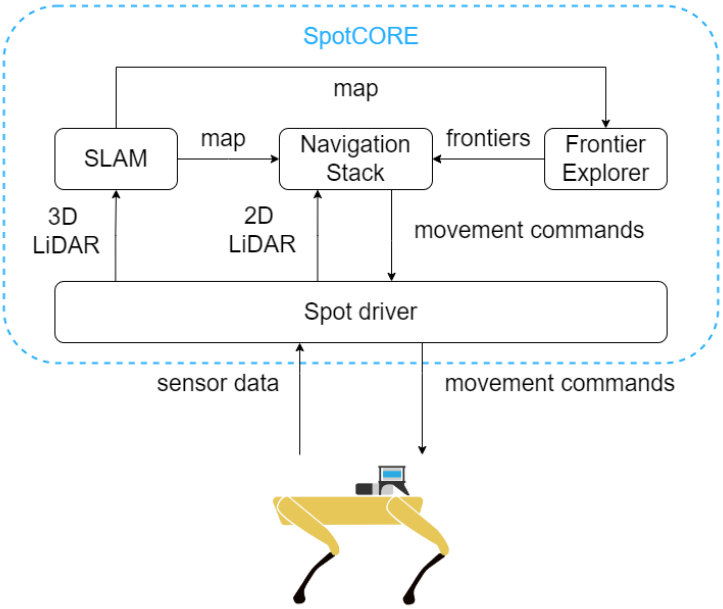


Figure 3.7: Abstract overview of the ROS System used for autonomous exploration.

3.3.2 Robot visualization

The Spot driver from Clearpath Robotics [24] comes with a package for visualizing the robot model in rviz, a visualization tool for ROS. An URDF (Unified Robot Description Format) file is provided with 3D meshes that can be displayed in rviz, see Figure 3.8.

Figure 3.8: Visualization of Spot in rviz.

CHAPTER 4

Implementation

4.1 SLAM (Simultaneous Localization and Mapping)

In this system, the ROS wrapper for Google's Cartographer [25] [26] is used. Because 3D SLAM will be tested in the future and Cartographer provides both 2D and 3D SLAM, Cartographer was chosen as the SLAM algorithm. To tackle the SLAM problem, Cartographer uses two subsystems: local SLAM (also called frontend or local trajectory builder) and global SLAM (also called backend). This section will detail Cartographer, explaining its mapping and localization approaches.

4.1.1 Local SLAM

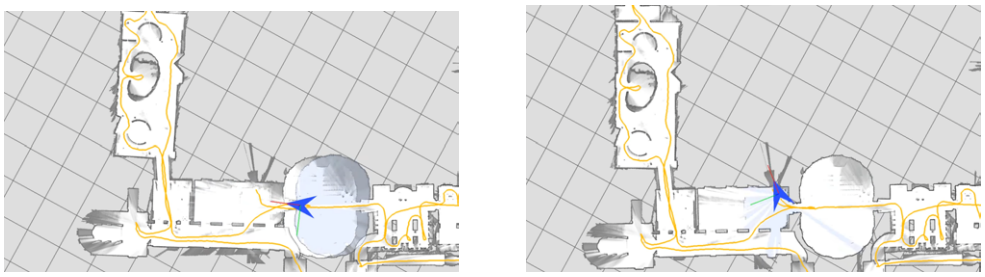
The main job of the local SLAM is to build submaps, which are partial chunks of the environment. This is done by scan matching consecutive scans of laser range data, for example from LiDAR sensors. Submaps take the form of probability grids, which are comparable to occupancy grids as mentioned in Chapter 2, where each grid cells has a certain probability of being obstructed/occupied.

Let's assume a new scan S is taken and the robot is currently moving in the submap M. The local SLAM is now trying to match the scan points $P_S \in \mathbb{R}^2$ of S to the points $P_M \in \mathbb{R}^2$ of the submap M. The goal of this scan matching is to find the pose ξ of the scan frame, which consists of (x, y) coordinates and a rotation angle θ. This can be done by finding the transformation T_ξ, which transforms all P_S from the scan frame into the submap frame. Cartographer uses a Ceres-based [27] scan matcher to find this transformation, by transforming the scan points such that the probability of the scan points in the submap is maximized. If a reasonably good transformation and thus the pose ξ of the scan is found, the scan is inserted into the submap. The drawback is that the error of the local SLAM increases as the number of scans increases. This issue is mitigated by the global SLAM with loop closure.

4.1.2 Global SLAM

Now that multiple submaps are inserted into the map, the next goal is to properly align all of them to form a coherent global map. This is done by the backend or global SLAM in Cartographer. The poses of all scans with their scan points are saved in nodes, building a *pose graph*, where edges are constraints between nodes. Cartographer then optimizes this graph by applying a Sparse Pose Adjustment (SPA) [28].

The main idea of the global SLAM is to find previous nodes or submaps that are eligible for loop closure, meaning that a certain part of the environment was already visited and should now connect to the rest of the built map. To find these connections, a scan matcher specifically designed for real-time loop closure with a *Branch and bound* mechanism is used. If a proper match is found, the optimization of the pose graph starts, resulting in rearranging the poses of submaps. In Figure 4.1 a visualization of the loop closure can be observed. The error of the local SLAM was mitigated by revisiting the oval-shaped room, causing all submaps to the left of the blue arrow to rotate counterclockwise.



(a) Before (b) After

Figure 4.1: Visualization of (a) before and (b) after loop closure in Cartographer. [29]

Given that Cartographer knows where the different scans were taken in a submap and the robot knows where the origin of the LiDAR is, the transform between the robot's body coordinate frame and the world coordinate frame can be published to `tf`, see Figure 4.2.

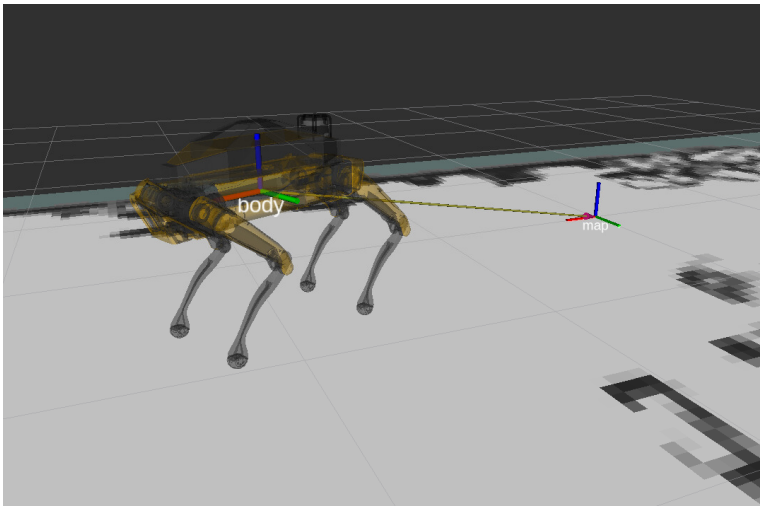


Figure 4.2: The tf link between the body frame and the map frame.

4.2 Navigation Stack

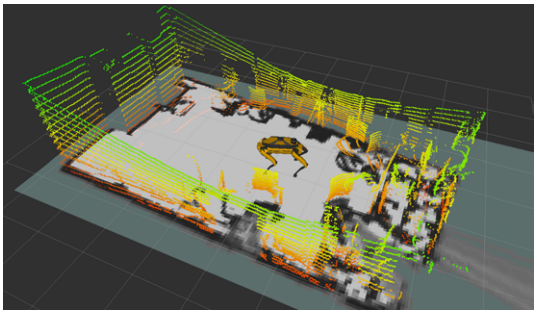
The tuning of the ROS Navigation Stack is crucial for a good performance of autonomous exploration for mobile robots. Zheng et al. [30] proposed a certain methodology to find the optimal configurations for several parts of the Navigation Stack such as path planners and costmaps.

4.2.1 Map

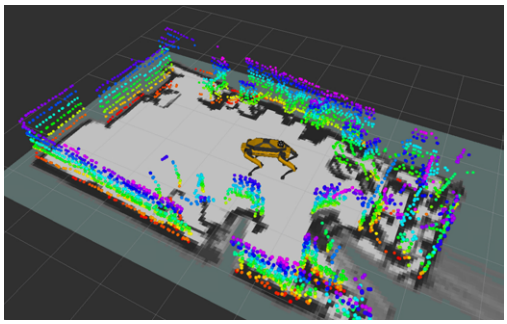
The map built by Cartographer is published in the form of `OccupancyGrid` [31] [11], where the values of probabilities range from $[0, 100]$ and unknown cells are equal to -1. The laser range data in form of a `PointCloud2` message is fed into Cartographer. The range of the Velodyne VLP-16 LiDAR is $100m$, but to limit the computational load the maximum range Cartographer considers is set to $10m$. Furthermore, the maximum height of the scan points is set to $0.6m$ as the points above are not important for Spot to move in 2D, consequently improving SLAM performance. See Figure 4.3 for the different laser range data that is used in this system.

4.2.2 Costmaps

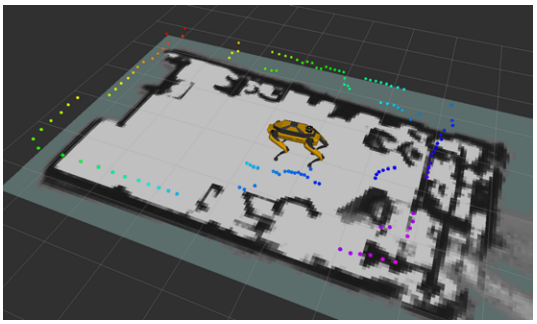
In ROS the *costmap_2d* package [20] implements the concept of costmaps. The purpose of costmaps is to enable path planners to plan an obstacle-free trajectory for the robot. Like the map provided by Cartographer SLAM, costmaps are occupancy grids, where occupancy is associated with a cost value in the range from $[0, 254]$ where 0 is free space and 254 an obstacle. [12] The obstacle layer, responsible for marking and clearing obstacles, in the local costmap is fed with 2D laser range data, as the 3D `PointCloud2` data would be a huge overhead for the SpotCORE. This is acceptable because the internal



(a) Untrimmed 3D laser range data from LiDAR



(b) Scan matched points from Cartographer.



(c) 2D range data.

Figure 4.3: Different laser range data used in the different components in this system.

collision avoidance of Boston Dynamics Spot performs adequately, making computational expensive object detection with 3D data would be meaningless.

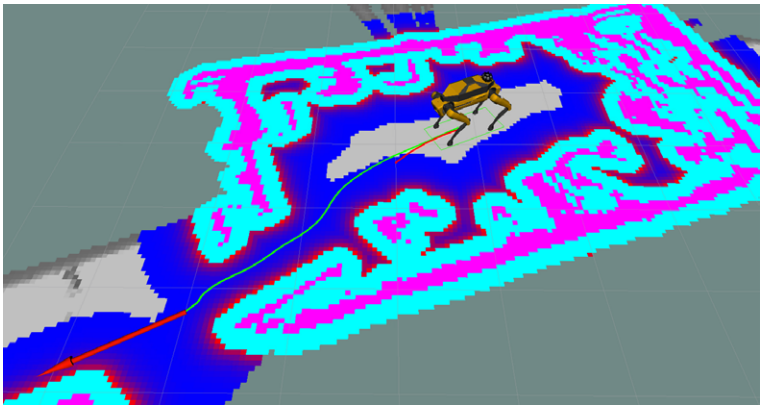
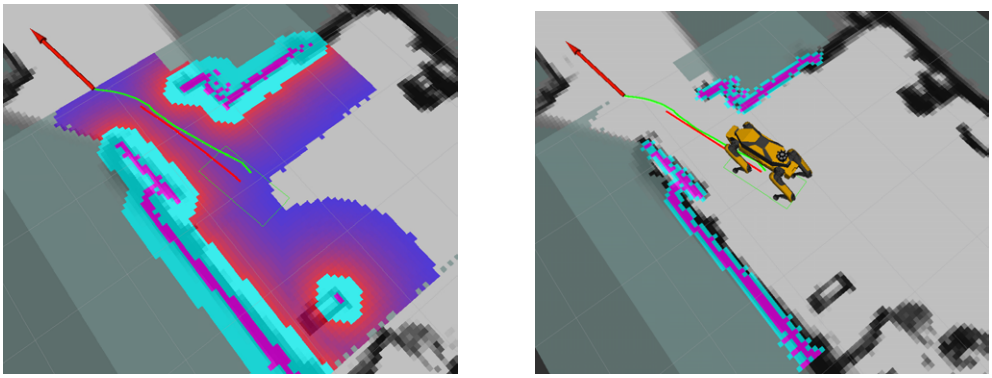


Figure 4.4: Global path in green, planning through the middle of a doorway with the global costmap. Cells in pink are obstacles, cyan to blue the inflation radius.

The inflation radius of the inflation layer is configured differently from the global

and local costmap as the inflation radius on the global costmap was chosen so the global path planner chooses a path in the middle between two obstacles as seen in Figure 4.4 compared to the local costmap where a high inflation radius would hinder the robot to move between narrow openings such as doorways. Thus, a minimal inflation radius was chosen ($0.01m$), so the robot can move, even in narrow openings, see Figure 4.5. The robot would not be able to move through the doorway shown in Figure 4.5a.



(a) Big inflation radius (b) Small inflation radius

Figure 4.5: Comparison between a big inflation radius (a) and a small inflation radius (b) on a local costmap.

4.2.3 Path Planners

The used global planner for this project is the `navfn` package [32]. `navfn` implements the global path planner using the Dijkstra algorithm. It computes a path with minimal cost from a starting grid cell to the end grid cell. The default behaviour of this path planner is that unknown cells are allowed to be traversed, but that would sometimes plan paths through walls if the costmap has holes in it, see Figure 4.6. Thus, the parameter `allow_unknown` is changed to `false` to avoid this behaviour.

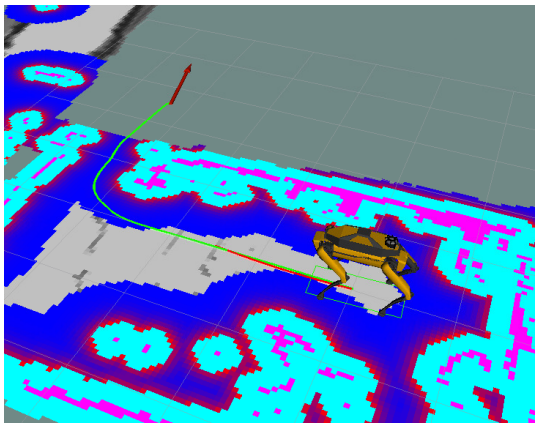


Figure 4.6: Global path planner planning through unknown grid cells.

The `base_local_planner` package [33] implements a local planner using two approaches. The *Trajecory Rollout* (TRA) and the *Dynamic Window Approach* (DWA). These two algorithms work very similarly in that they score potential trajectories the robot could take in a certain amount of time and choose the trajectory with the best score. Since calculating all possible trajectories in the near vicinity of the robot would be computational unfeasible, only certain velocities (uniformly sampled translational velocities in x and y and rotational velocity) are simulated for a certain amount of time (simulation time) which lead to a handful of trajectories that can be scored. Trajectories that are colliding with obstacles are discarded. The score is dependent on many factors such as:

- proximity to global path, so the robot does not deviate too much from the global path
- proximity to goal, so the robot does not overshoot the goal
- proximity to obstacles, so the robot moves not too close to obstacles

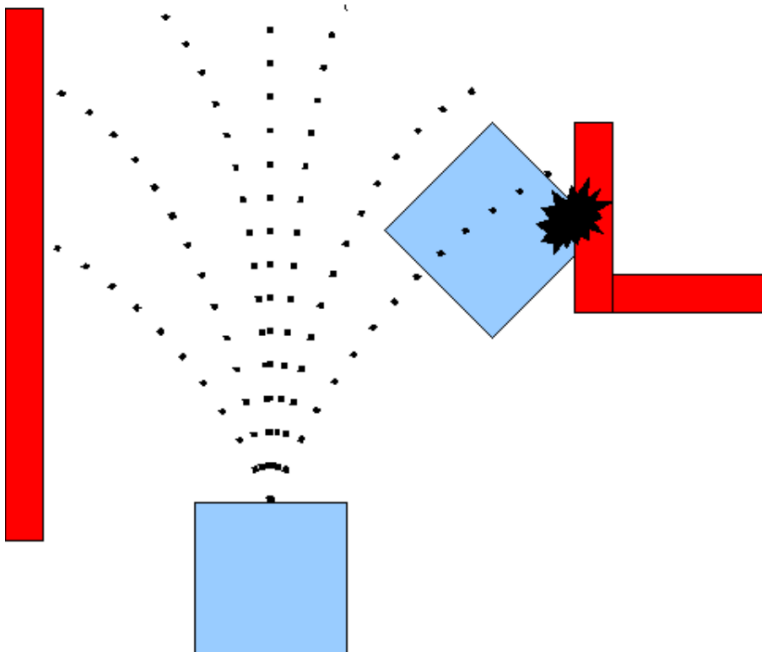


Figure 4.7: Trajectory Rollout Approach with simulated trajectories in black, obstacles in red and the mobile robot in blue. [33]

The difference between the TRA and the DWA is that DWA only considers simulating the sampled velocities within one simulation step so the robot can safely stop, thus making DWA theoretically faster as it must score fewer trajectories [34] [33].

However, while testing the local planner with the `base_local_planner` package, DWA resulted in significantly poorer performance with Spot, thus the TRA was chosen instead.

4.3 Frontier Exploration

Once the mapping of the environment and the robot localization are done, as well as the path computing, the last step for autonomous exploration is to find unexplored areas. As described in Chapter 2, one way is to find frontiers in the map.

The package `explore_lite` [35] implements this approach of frontier exploration. Contrary to other implementations such as `frontier_exploration` [36], explore_lite does not create its own costmap, making it more lightweight, thus this package was chosen over others. The general approach of this algorithm is to first find frontier cells and them. Then a filter is applied, so frontiers under a certain size get discarded. After that, a cost of a frontier depending on their distance to the robot (Euclidean distance) and size of the frontier is computed. The last step is to send the centroid of the frontiers as `goal` messages to `move_base`. If a frontier is unreachable, it gets marked as such and the next available frontier is chosen as the goal.

To search frontier cells, a BFS (Breadth-first search) is implemented. Starting from the cell the robot is currently on, a neighbourhood of 4 cells is examined. If a cell is free (has a value of 0) and was not visited before, it is inserted into a queue. If a cell is unvisited, has an unknown value associated with it and has at least one free cell adjacent to it, a BFS from this frontier cell is initiated to find neighbouring frontier cells. This time, the iteration is over all 8 neighbouring cells. Should the size of the frontier be over the minimum size limit, the frontier is added to the list of frontiers the robot should explore. Should the search terminate, the pose of the centroid of the frontier with the least cost, defined as the difference between the minimal Euclidean distance of the frontier to the robot and the size of the frontier, is sent to `move_base`.

The calculation of the centroid of the frontier is used with world coordinates and is averaged out over the size of the frontier which can result in goals that land on unknown cells. Furthermore, the algorithm does not adhere to the definition of frontier cells of [12], where frontier cells must be free cells, thus making the centroid more likely to fall on unknown cells. As described in Section 4.2.3, the global planner is set so it does not plan through unknown cells, thus thus limiting this drawback of `explore_lite`. Instead of making the centroid of the frontier the goal, the nearest free cell to the centroid is chosen as the goal of the frontier. This is done via the same 8-neighbourhood BFS `explore_lite` uses for finding adjacent frontier cells, but this time the search stops if a free cell is found.

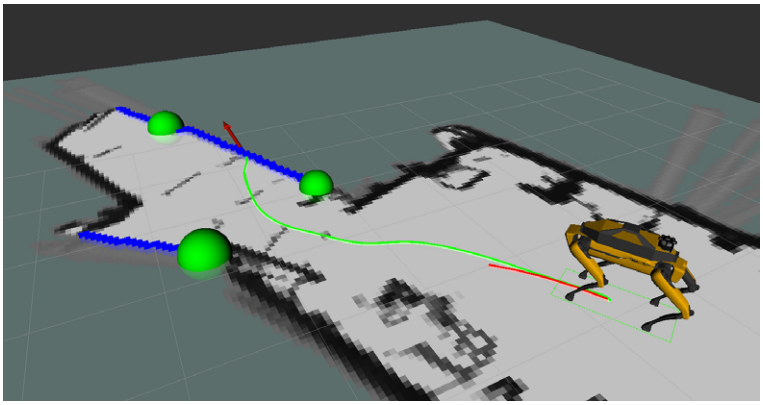


Figure 4.8: Frontiers visualized in rviz. Blue cells indicate the frontiers, green spheres are the initial found frontier cells and the size is the cost of the frontier, the bigger the cost the smaller the sphere is.

4.4 ROS System

To get an overview of the ROS Computation Graph used in this project Figure 4.9 shows the simplified version of it. There are some details hidden for better visibility of the graph and a better understanding of the system. This section will describe the various nodes that take part in this graph and explain what topics they subscribe to / publish.

At the start of launching the `spot_driver` package provided by Clearpath Robotics [24] it automatically launches a velodyne_node, which is a separate package developed for Velodyne LiDARs for ROS [37]. The node takes the laser range data from the LiDAR and converts it into ROS messages and publishes them. It publishes two kinds of topics `/velodyne_points` (3D LiDAR data) and `/scan` (2D LiDAR data), see Figure 4.3 for visualization.

Cartographer then subscribes to the point cloud topic `/velodyne_points`, the `/odometry` topic and to `/tf` for constructing the map and localizing Spot. If the map is generated, Cartographer publishes the `/map` topic in form of an occupancy grid and publishes the link between the map and body frame of Spot to `tf`, thus localizing the robot in the map coordinate frame.

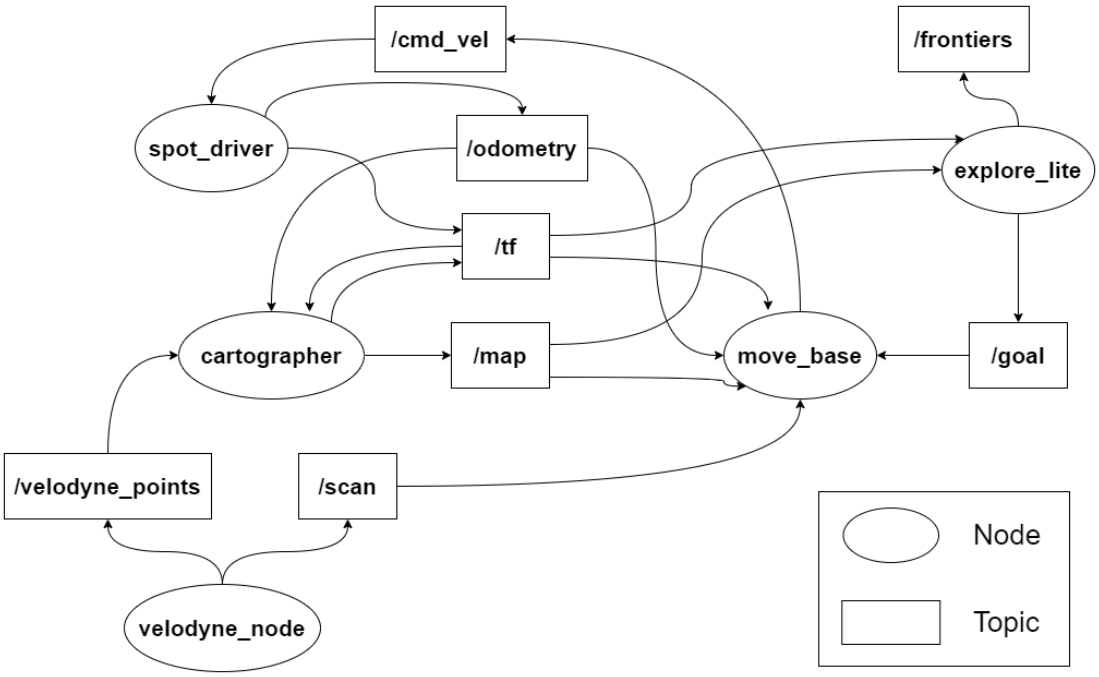


Figure 4.9: Simplified ROS Computation graph of the system.

`Move_base` needs to know where the robot is located in the world and what is occupied in the environment, thus it subscribes to the `/tf` and `/map` topic. Furthermore, it needs range data and some general state of motion of the robot for the `base_local_planner`, so `move_base` subscribes to the 2D laser range data topic `/scan` and the `/odometry` topic.

The `explore_lite` node is responsible for finding frontiers in the map, so it subscribes to the `/map` topic. To know when Spot has reached a frontier the location of the robot has to be known, so it subscribes to the `/tf` topic. If the node finds frontiers it publishes them (for visualization) and publishes the `/goal` topic of the nearest frontier for `move_base`, which is a pose, consisting of (x, y) coordinates and a rotation angle θ. `Move_base` subscribes to the `/goal` topic and computes movement commands on an obstacle-free path and publishes them via the `/cmd_vel` topic. The `/spot_driver` subscribes to the `/cmd_vel` topic and sends movement commands to Spot.

CHAPTER 5

Results and Discussion

5.1 Metrics

To evaluate the autonomous exploration of mobile robots, different metrics can be used. Yan et al. [38] proposed multiple performance metrics for multi-robot exploration, which also can be used for single-robot exploration. For this project, two different metrics were used: (i) quality and (ii) completeness of the map. To calculate the completeness of the map, a comparison with a ground truth map, which is built with the ground floor plan of the building, is done. Let the built map with SLAM be B and the ground truth map G. The completeness C of the map is defined as the ratio between the area of B and G:

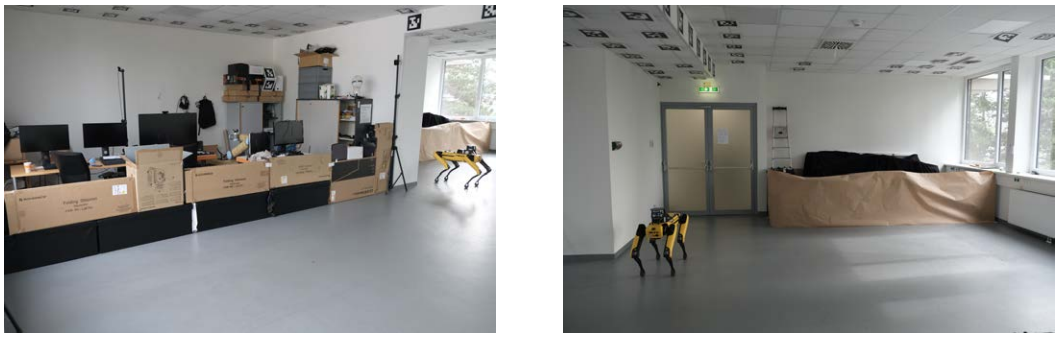
$$C = \frac{A(B)}{A(G)}$$

To define the quality of the map two different metrics are used. Santos et al. [6] proposed the **normalized k nearest-neighbour error (NNE)**, which computes the average distance of nearest neighbours of occupied cells between the ground truth and the built map using the nearest-neighbour approach. This provides a global perceived error of the map, as the metric is used on the whole map, which is why a second metric is used to benchmark local structure quality. This type of metric combination was proposed by [39].

The **Structure Similarity Index Measure (SSIM)** [40] uses local windows to compare two images. It calculates the mean intensity of the luminance, the standard deviation to estimate the contrast and the covariance for the structural comparison of both images. The result of the comparison is between $[-1, 1]$ with 1 being the most similar. The value of 1 is only possible if both images are exactly the same. This SSIM enables for better estimation of local structural similarity in the constructed map by SLAM, hence it was used besides NNE.

5.2 Real World Experiment

To test the autonomous exploration with Spot, a room at the TU Vienna Science Center was chosen to be the test site. The room was prepared/arranged to create a free walkable area, see Figure 5.1.



(a) Left Barrier (b) Right barrier

Figure 5.1: Test site at the TU Vienna Science Center with added barriers.

The ground-truth is represented by a 3D model of the room, created by using a 3D software tool. The room sizes have been manually measured. The 3D model was updated to represent the new barricades set up for the experiment, see Figure 5.2. The ground truth map is now gathered by making a planar cut of all obstacles resulting in an occupancy grid, similar to one generated by a SLAM algorithm.

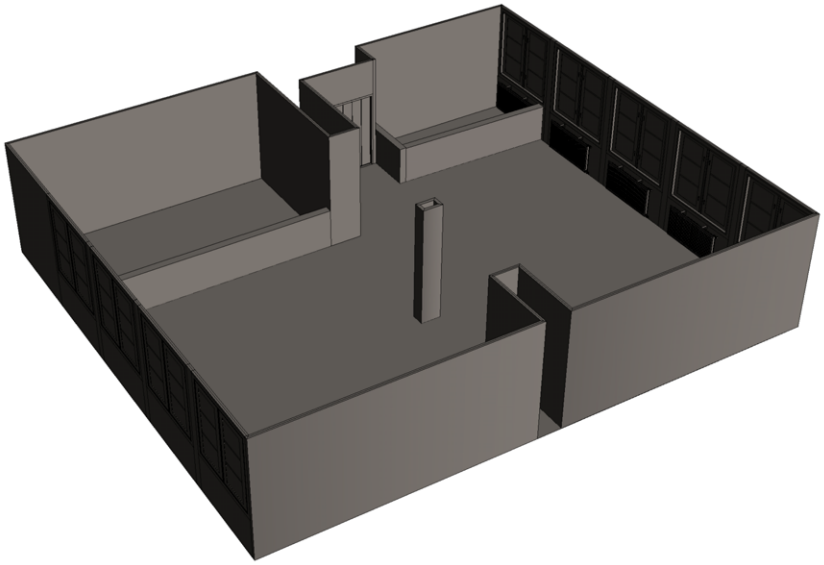
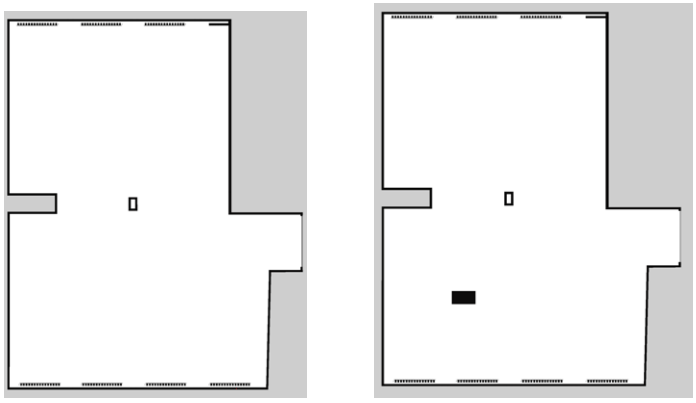


Figure 5.2: The 3D model of the test site.

Two scenarios were chosen for the autonomous exploration. In scenario 1, the room was empty. In scenario 2, one obstacle was added into the room to test how Spot reacts to small obstacles in an empty room, Figure 5.3 shows the ground truth maps of both scenarios. The experiment is over if no frontiers are left to explore or the last frontiers are not reachable.



(a) Scenario 1: no obstacles

(b) Scenario 2: obstacle in map

Figure 5.3: Ground truth maps of both scenarios. Black cells are obstacles, white is free space and gray is unknown space.

5.3 Evaluation Pipeline

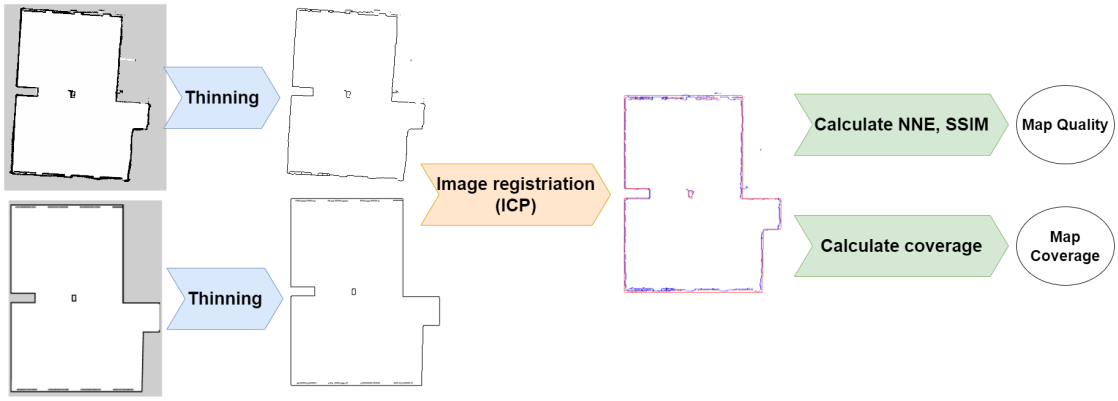
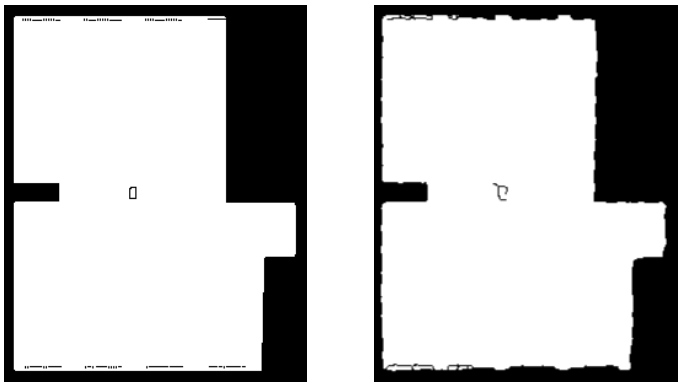


Figure 5.4: Evaluation pipeline used in this project. The red walls are the ground truth map and the blue the built map.

For comparing the ground truth map with the built map several steps must be taken to obtain accurate results. Firstly, the built map is saved with the `map_server` package. Then, a thinning operation with OpenCV [41] on both the ground truth map and the

built map is applied, as the width of the walls can vary. After that, the two maps must be aligned to compute the error metrics. This is done with the ICP (Iterative closest point) algorithm [42] in MATLAB, which provides an affine transformation that best aligns the two maps. To compute the NNE, the `knnsearch` function provided by MATLAB is used. The NNE is computed by calculating the distances to the nearest neighbour of all occupied cells in the ground truth map. Then, the mean of the sum of all distances is taken, resulting in the NNE. The SSIM is calculated with the built-in function of MATLAB. The input is the two thinned and aligned maps.

To derive the coverage of the built map, the area of all free cells in the built map is compared to the area of all free cells in the ground truth map. This is done by marking the area outside the walls as occupied, see Figure 5.5, meaning we have a binarized image where the function `bwarea` of MATLAB can be used to estimate the area of the free space.



(a) Ground truth area in scenario 1

(b) Built map area in scenario 1

Figure 5.5: Area of ground truth map (a) and built map (b) with no obstacles.

5.4 Results

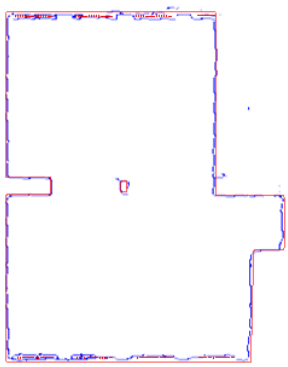
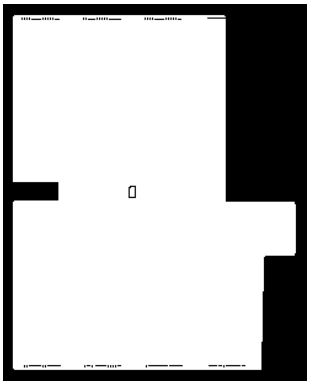
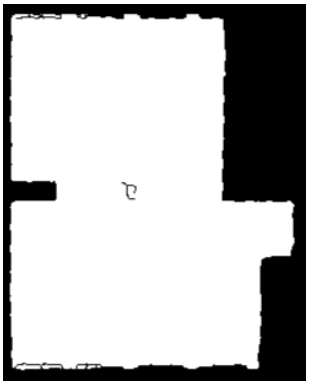
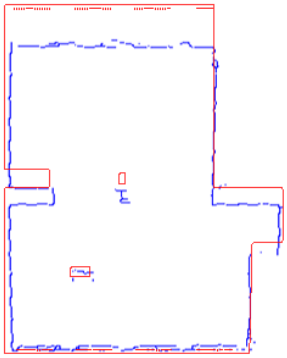
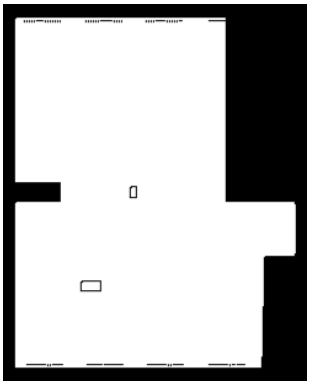
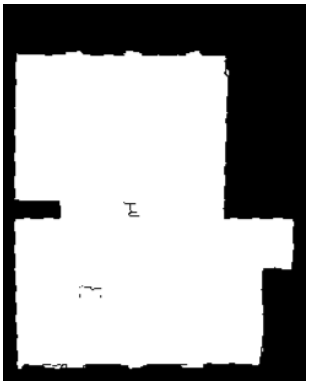
Type	Overlap	Area GT	Area SLAM
Scenario 1			
Scenario 2			

Table 5.1: Results of both scenarios. The blue line represents the built map and the red the ground truth map.

Scenario	NNE	SSIM	Map Coverage
1: no obstacles	0.81	0.89	99.04%
2: obstacles	5.30	0.82	89.04%

Table 5.2: Error metrics and map coverage results for both scenarios. NNE the lower the better, SSIM value 1 is perfect similarity.

In both scenarios, Spot reached all frontiers with no frontiers left to explore.

The results show a good construction of the map in scenario 1, with a map coverage of 99.04% and an NNE of 0.81 and an SSIM of 0.89. In contrast to scenario 1, the results of scenario 2 show a mediocre result as the map is appearing to be compressed in the y-direction, hence the map coverage is only 89.04% and the NNE is 5.30 while the SSIM is 0.82. This error potentially happened, because Spot took a different trajectory, caused by the new frontier behind the obstacle.

5.5 Limitations

This system currently only works in 2D. Spot is already capable of moving up and down stairs but allowing autonomous exploration without a first manual initialization is currently not available. Furthermore, a lot of ROS packages for navigation are designed for use in 2D, making it more time-consuming to design an autonomous exploration system for 3D as several parts must be implemented from scratch.

The LiDAR mounted on Spot is used for mapping the environment. LiDARs have a huge disadvantage in environments with glass, as the laser pulses emitted by the LiDAR travel through glass, thus making the laser range data inaccurate and making glass panels invisible in the occupancy grid. The internal object detection of Spot can't detect glass panels either so Spot collides with glass panels if a path goes through a glass panel.

CHAPTER 6

Future Work

The next step for using Spot to explore the environment autonomously would be to enable navigation through 3D. Spot should be able to climb stairs and map its environment in 3D autonomously. Cartographer [11] was used for 2D SLAM but it also has a mode for 3D SLAM. In the 3D mode of Cartographer, an IMU (Inertial measurement unit) is required for helping with building maps. Spot has an internal IMU but the sensor data is not accessible, so installing an external IMU would be necessary to run 3D SLAM with Cartographer on Spot. Finding / Implementing ROS navigation and exploration packages that work in 3D is also the next step for designing a navigation stack that works in 3D.

As there are no simulation software provided by Boston Dynamics for Spot, one must either disregard simulating with Spot or try third-party simulation software [43] [44]. A future first-party simulation software would be really useful, as developers can test algorithms without needing the physical robot. Furthermore, optimal settings for the navigation stack can be found by simulating the robot moving through several environments with different configurations, until the best set of configuations is found.

Currently, the autonomous exploration with Spot is highly dependent on having no glass panels in the environment, as the LiDAR is not capable of sensing glass panels. One approach to tackle this problem is to fuse multiple sensors together, where one of the sensors is capable of sensing glass panels such as ultrasonic sensors. This multi-sensor fusion was tested by Wei et al. [45]. Alternatively, Wang et al. developed a method of detecting glass panels based on detecting the specular reflection of laser beams and modified an existing SLAM method for integration with ROS [46] [47]. A modification to Cartographer in 2D and 3D or setting up a multi-sensor fusion with sonar sensors would make Spot able to detect glass, so this step is highly favourable as most buildings have some sort of glass panels making the autonomous exploration for Spot nearly impossible without covering all glass panels.

The acquired map scanned with the SLAM algorithm is useful for Spot using the navigation or getting a rough overview of the environment but if one wants to have high accuracy 3D scans of the environment an extra payload such as a high accuracy laser scanner should be used. The TU Vienna has purchased a Riegl [48] laser scanner which can be used to scan accurate point clouds of the environment, and also can be mounted on Spot. To combine autonomous exploration while simultaneously scanning the environment in short intervals is the goal of further research.

CHAPTER 7

Conclusion

This thesis presented a system that allows a mobile robot to autonomously explore its environment in 2D. Spot, a quadrupedal robot from Boston Dynamics, was used as the mobile robot, equipped with a 3D LiDAR. The autonomous exploration was implemented with a frontier-based approach with the open-source robotics framework ROS combined with Google's Cartographer SLAM algorithm [11].

To test the system, a real-world experiment was conducted with two different scenarios at the TU Vienna Science Center. In the first scenario, the lab was empty and in the second one an obstacle was added in the middle of the free space. The first goal of the experiment was to verify the effectiveness of the implemented frontier-based exploration. After reaching all frontiers, the quality and coverage of the built map were examined. A ground truth map was constructed with a 3D model of the lab and was then used to compare it to the built map by using two different error metrics NNE and SSIM and the map coverage.

Spot was able to reach all detected frontiers and map the lab in both scenarios. The map coverage and quality in both scenarios were acceptable. In the first scenario, SLAM produced a highly accurate map with almost no NNE and very good map coverage of almost 100%. The error produced in scenario 2 is potentially due to the different trajectory Spot took after sensing the frontier behind the object. More research must be done to come to a better understanding of the impact different trajectories have on the SLAM algorithm. Furthermore, to enable better loop closure by the SLAM algorithm the mobile robot should be programmed to return to the starting position after the exploration is done.

List of Figures

Bibliography

[1] J. Jennings, G. Whelan, and W. Evans, "Cooperative search and rescue with a team of mobile robots," in *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, pp. 193–200.

[2] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, "3D path planning and execution for search and rescue ground robots," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 722–727.

[3] H. Unger, T. Markert, and E. Müller, "Evaluation of use cases of autonomous mobile robots in factory environments," vol. 17, pp. 254–261. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978918311600 (Accessed 2022-09-09).

[4] D. Di Paola, D. Naso, A. Milella, G. Cicirelli, and A. Distante, "Multi-Sensor Surveillance of Indoor Environments by an Autonomous Mobile Robot," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, pp. 23–28.

[5] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," vol. 17, no. 3, pp. 229–241.

[6] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6.

[7] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An open-source Robot Operating System," p. 6.

[8] Spot® - The Agile Mobile Robot | Boston Dynamics. [Online]. Available: https://www.bostondynamics.com/products/spot (Accessed 2022-09-09).

[9] J. Lee, "Hierarchical controller for highly dynamic locomotion utilizing pattern modulation and impedance control : Implementation on the MIT Cheetah robot." [Online]. Available: https://dspace.mit.edu/handle/1721.1/85490 (Accessed 2022-09-06).

[10] Getting started with Autowalk. [Online]. Available: https://support.bostondynamics.com/s/article/Getting-Started-with-Autowalk (Accessed 2022-09-09).

[11] Cartographer ROS Integration — Cartographer ROS documentation. [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/ (Accessed 2022-08-27).

[12] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'.* IEEE Comput. Soc. Press, pp. 146–151. [Online]. Available: http://ieeexplore.ieee.org/document/613851/ (Accessed 2022-08-19).

[13] D. Holz, N. Basilico, F. Amigoni, and S. Behnke, "Evaluating the Efficiency of Frontier-based Exploration Strategies," in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, pp. 1–8.

[14] R. Yagfarov, M. Ivanou, and I. Afanasyev, "Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1979–1983.

[15] Documentation - ROS Wiki. [Online]. Available: http://wiki.ros.org/Documentation (Accessed 2022-08-23).

[16] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160.

[17] T. Foote, "Tf: The transform library," in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pp. 1–6.

[18] R. L. Guimarães, A. S. de Oliveira, J. A. Fabro, T. Becker, and V. A. Brenner, "ROS Navigation: Concepts and Tutorial," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*, ser. Studies in Computational Intelligence, A. Koubaa, Ed. Springer International Publishing, pp. 121–160. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9_6 (Accessed 2022-08-24).

[19] Navigation/Tutorials/RobotSetup - ROS Wiki. [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup (Accessed 2022-08-24).

[20] Costmap_2d - ROS Wiki. [Online]. Available: http://wiki.ros.org/costmap_2d (Accessed 2022-08-28).

[21] Spot Enhanced Autonomy Payload (EAP). [Online]. Available: https://support.bostondynamics.com/s/article/Spot-Enhanced-Autonomy-Package-EAP (Accessed 2022-08-25).

[22] Spot specifications. [Online]. Available: https://support.bostondynamics.com/s/article/Robot-specifications (Accessed 2022-08-25).

[23] Spot CORE payload reference. [Online]. Available: https://support.bostondynamics.com/s/article/Spot-CORE-payload-reference (Accessed 2022-08-25).

[24] "Spot ROS Driver," clearpathrobotics. [Online]. Available: https://github.com/clearpathrobotics/spot_ros (Accessed 2022-05-12).

[25] "Cartographer ROS Documentation," p. 52.

[26] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278.

[27] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 3 2022. [Online]. Available: https://github.com/ceres-solver/ceres-solver

[28] 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*, pp. 22–29.

[29] "Cartographer 2D SLAM Loop Closure Demo." [Online]. Available: https://www.youtube.com/watch?v=-EQAJOoRqEQ (Accessed 2022-08-28).

[30] K. Zheng, "ROS Navigation Tuning Guide," p. 23.

[31] Nav_msgs/OccupancyGrid Documentation. [Online]. Available: http://docs.ros.org/en/api/nav_msgs/html/msg/OccupancyGrid.html (Accessed 2022-08-28).

[32] Navfn - ROS Wiki. [Online]. Available: http://wiki.ros.org/navfn (Accessed 2022-08-28).

[33] Base_local_planner - ROS Wiki. [Online]. Available: http://wiki.ros.org/base_local_planner (Accessed 2022-08-28).

[34] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," vol. 4, no. 1, pp. 23–33. [Online]. Available: http://ieeexplore.ieee.org/document/580977/ (Accessed 2022-08-30).

[35] Explore_lite - ROS Wiki. [Online]. Available: http://wiki.ros.org/explore_lite (Accessed 2022-08-28).

[36] Frontier_exploration - ROS Wiki. [Online]. Available: http://wiki.ros.org/frontier_exploration (Accessed 2022-08-30).

[37] Velodyne - ROS Wiki. [Online]. Available: http://wiki.ros.org/velodyne (Accessed 2022-08-31).

[38] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi, "Metrics for performance benchmarking of multi-robot exploration," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3407–3414.

[39] X. S. Le, L. Fabresse, N. Bouraqadi, and G. Lozenguez, "Evaluation of Out-of-the-Box ROS 2D SLAMs for Autonomous Exploration of Unknown Indoor Environments," in *Intelligent Robotics and Applications*, ser. Lecture Notes in Computer Science, Z. Chen, A. Mendes, Y. Yan, and S. Chen, Eds. Springer International Publishing, vol. 10985, pp. 283–296. [Online]. Available: http://link.springer.com/10.1007/978-3-319-97589-4_24 (Accessed 2022-08-17).

[40] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," vol. 13, no. 4, pp. 600–612.

[41] OpenCV: OpenCV modules. [Online]. Available: https://docs.opencv.org/4.x/ (Accessed 2022-09-22).

[42] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," vol. 10, no. 3, pp. 145–155. [Online]. Available: https://www.sciencedirect.com/science/article/pii/026288569290066C (Accessed 2022-09-15).

[43] SoftServeSAG/robotics_spot at temp. GitHub. [Online]. Available: https://github.com/SoftServeSAG/robotics_spot (Accessed 2022-09-06).

[44] "Champ," CHAMP. [Online]. Available: https://github.com/chvmp/champ (Accessed 2022-09-06).

[45] H. Wei, X.-e. Li, Y. Shi, B. You, and Y. Xu, "Multi-sensor Fusion Glass Detection for Robot Navigation and Mapping," in *2018 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, pp. 184–188.

[46] X. Wang and J. Wang, "Detecting glass in Simultaneous Localisation and Mapping," vol. 88, pp. 97–103. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889015302670 (Accessed 2022-09-06).

[47] "Real-time indoor SLAM with glass detection," UTS Magic Lab. [Online]. Available: https://github.com/uts-magic-lab/slam_glass (Accessed 2022-09-06).

[48] RIEGL - Produktdetail. [Online]. Available: http://www.riegl.co.at/nc/products/terrestrial-scanning/produktdetail/product/scanner/48/ (Accessed 2022-09-06).