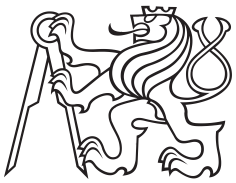
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Mobile Robot Navigation on Rough Terrain

Vratislav Besta

Supervisor: Ing. Jan Chudoba
Field of study: Cybernetics and Robotics
January 2023

Acknowledgements

I would like to express my deepest thanks to my supervisor, Ing. Jan Chudoba, for his invaluable guidance and insights.

My sincere thanks also go to my friend Bc. Tomáš Fiala for the engaging on-topic debates we have had throughout our studies.

Last but not least, I would like to thank all the developers of the software I use in this thesis for their contribution and for showing me the beauty of the open-source world.

Declaration

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

In Prague, January 10th 2023

.....
Author's signature

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 10. ledna 2023

.....
Podpis autora práce

Abstract

Rough terrain poses a great challenge to the navigation task due to its diversity and traversability estimation as opposed to flat surface areas. This bachelor's thesis deals with the point-to-point navigation of a mobile skid-steered ground robot in an unknown, primarily outdoor, rough-terrain environment.

The first part reviews state-of-the-art mapping methods, especially Simultaneous Localization and Mapping (SLAM) algorithms.

The second part presents a design of the navigation system and the implementation of the proposed method as a Robot Operating System (ROS) project. The designed method incorporates Lidar Inertial Odometry via Smoothing and Mapping (LIO-SAM) algorithm for pose estimation and registration of lidar scans. A continuously created 2.5D traversability map serves as an underlay for the path-planning task, which utilizes a combination of the global and local planner, using the Dijkstra and the Dynamic Window Approach (DWA) algorithm, respectively.

Subsequently, several experiments are conducted in a simulator to show both the constraints of the proposed method and the ability to plan a traversable obstacle-free path through a given environment.

Keywords: mobile robot, rough terrain navigation, traversability map, ROS, Gazebo simulator, LiDAR

Supervisor: Ing. Jan Chudoba
Czech Institute of Informatics, Robotics and Cybernetics,
Jugoslávských partyzánů 1580/3,
160 00 Prague 6, Dejvice

Abstrakt

Nerovný terén představuje velkou výzvu pro úlohu navigace díky své rozmanitosti a odhadu jeho sjízdnosti oproti rovným plochám. Tato bakalářská práce se zabývá navigací z bodu do bodu smykově řízeného pozemního mobilního robotu v neznámém, primárně venkovním, nerovném prostředí.

První část shrnuje state-of-the-art metody mapování, především algoritmy simultánní lokalizace a mapování (SLAM).

Druhá část představuje návrh navigačního systému a jeho implementaci v podobě Robot Operating System (ROS) projektu. Navržená metoda využívá algoritmu lidarové inerciální odometrie pomocí vyhlazování a mapování (LIO-SAM) pro odhad polohy a registraci lidarových skenů. Průběžně tvořená 2.5D mapa sjízdnosti slouží jako podklad pro úlohu plánování cesty, využívající kombinaci globálního a lokálního plánovače, které v pořadí používají Dijkstrův algoritmus a algoritmus konceptu dynamického okna (DWA).

Nasledně je v simulátoru provedeno několik experimentů ukazující omezení a schopnost navržené metody naplánovat bezkolizní sjízdnou cestu skrz dané prostředí.

Klíčová slova: mobilní robot, navigace v nerovném terénu, mapa sjízdnosti, ROS, Gazebo simulátor, LiDAR

Překlad názvu: Navigace pozemního robotu v nerovném terénu

Contents

Figures

Tables

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Besta Vratislav** Personal ID number: **491903**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Mobile Robot Navigation on Rough Terrain

Bachelor’s thesis title in Czech:

Navigace pozemního robotu v nerovném terénu

Guidelines:

Topic is targeted on an implementation of navigation method for mobile robot in rough terrain. Input of the method is a target position to be reached. Method has to map robot surrounding environment, evaluate its traversability, plan a suitable path and generate control commands for robot, which will lead him along this path.
• Do a research on environment mapping methods and sensors suitable for the specified task.
• Design a method for the destination point navigation. Discuss the suitable sensoric equipment with the supervisor.
• Implement the designed method and test it in a simulator and/or (according to real conditions) with a laboratory robot.
• Evaluate performed tests and discuss capabilities and limits of the designed method.

Bibliography / sources:

[1] THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. Probabilistic robotics. Massachusetts: MIT Press, c2006. ISBN 9780262201629.
[2] Kelly, A. (2013). Mobile Robotics: Mathematics, Models, and Methods. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139381284.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Chudoba Intelligent and Mobile Robotics CIIRC

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **10.01.2022** Deadline for bachelor thesis submission: **10.01.2023**

Assignment valid until: **30.09.2023**

Ing. Jan Chudoba	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt

Student’s signature

Chapter 1

Introduction

1.1 Context

Mobile robotics technology has advanced significantly in the last two decades, making it a more accessible tool for various industrial sectors, particularly in the field of logistics, where autonomous carriers are commonly used in warehouses and distribution centers. These robotic platforms are capable of safe autonomous navigation in flat, often human-occupied environments.

Recent advancements in sensor technology, computational hardware, and decreases in manufacturing costs have expanded the scope of mobile robotics applications to more challenging outdoor and urban environments, including uneven and rough terrain. These developments have opened up new areas of application for mobile robotics, including agriculture, rescue missions, security, and inspections.

Overall, the use of mobile robotics has allowed for significant automation of processes that previously required human operators, resulting in increased efficiency and reliability.

The process of autonomous mobile robot navigation involves safe travel to pre-defined locations in both known and unknown environments. In order to navigate through an unknown environment, a robot observes its surroundings using onboard sensors, creating an inner map, in which it must be able to determine its location. One solution to this problem is the Simultaneous Localization and Mapping (SLAM) method, which utilizes measurements from various sensors such as wheel encoders, Inertial Measurement Unit (IMU), lidar, camera, and Global Positioning system (GPS).

The continuously built inner map is divided into obstacles and traversable areas based on robot's specifications, for example maximum traversable grade, wheel diameter, and ground clearance, resulting in a traversability map. This map serves as the basis for a path planning algorithm, typically a combination of global and local planning, to find an obstacle-free path to the target goal. The global planner uses the entire map to plan the path to the goal, while the local planner takes into account dynamic obstacles in the robot's mapped proximity and generates corresponding motion commands.

1.2 Specification

The objective of this thesis is to design and implement a point-to-point rough terrain navigation method for a ground mobile robot. This will involve the use of a SLAM algorithm, traversability estimation on the inner map, obstacle-free path planning, and generating motion commands for the robot. The state-of-the-art SLAM methods will be reviewed to determine the most suitable approach for the selected sensor equipment. The method will be implemented as a ROS Noetic project, comprising packages written predominantly in C++ programming language. However, the complete implementation of all components of the navigation system is beyond the scope of a bachelor's thesis, and therefore, existing implementations of selected components will be integrated in the form of ROS packages. Furthermore, the navigation method will be restricted to static environments only. The resulting system should allow the user to set a target goal through an user interface, and offline maps will be integrated for this purpose. Finally, the performance of the resulting navigation system will be experimentally evaluated on various terrains in a simulator.

Chapter 2

Sensors of mobile robotics

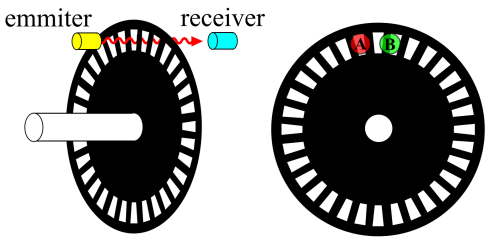
In this section, we introduce the sensors used in mobile robotics with an emphasis on those employed by SLAM methods. The working principle of each sensor is described in detail, along with a discussion of the advantages and disadvantages of using them.

Wheel encoders

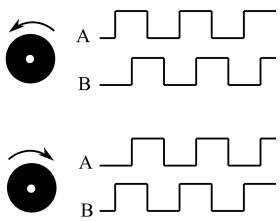
One way to measure the distance or velocity of a robot is to use incremental rotary encoders, which are mechanically coupled with the wheel axles. There are several types of incremental encoders, but this description will focus on those based on the optical principle.

An optical incremental rotary encoder consists of a rotary disk with evenly spaced opaque zones and a combination of two light emitters and receivers shown in Figure 2.1a. These optical components are arranged in such a way that the binary signals produced by the receivers, when an angular rate is applied to the disk, form a quadrature signal depicted in Figure 2.1b. This means that the phase shift between the two signals is a quarter of the period, allowing for the determination of the rotational direction of the disk.

The wheel angular rate can be deduced from the measured pulse count of the encoders, and the wheel's traveled distance can then be calculated using the wheel circumference.



(a) : The working principle of the optical encoder and placement of the pairs emmiter/receiver, respectively.



(b) : The quadrature signal depending on the direction of rotation.

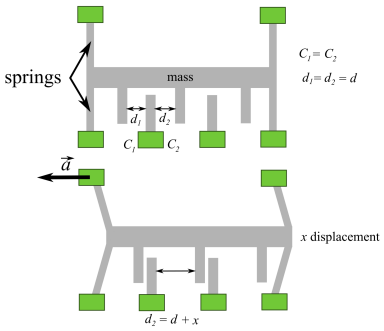
Figure 2.1: Wheel incremental optical encoder.

Wheel encoders provide a simple solution to estimating the velocity and the distance travelled by a robot. However, this approach relies on the assumption of ideal robot kinematics, which is rarely met in practice. For example, wheel odometry can become inaccurate when the wheels are not equally pressurized. Additionally, when a robot is operating on rough terrain, its tires may slightly deform or slip, leading to inaccurate pose estimation using this method [1]. Therefore, the use of wheel encoders is not suitable solution for navigation on rough terrain.

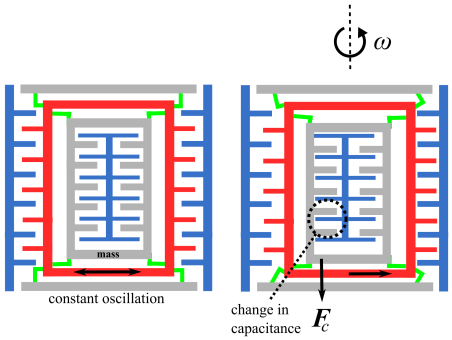
Inertial measurement unit (IMU)

In mobile robotics, the IMU is often in the form of a microelectromechanical system (MEMS) that consists of three orthogonally placed accelerometers and gyroscopes, and sometimes magnetometers. This configuration allows for the measurement of linear acceleration along each of three orthogonal axes and angular rate around these axes, which can be used to estimate the orientation of the sensor and therefore a robot.

The MEMS accelerometer typically operates on the principle of differential capacitance, where motion of the system leads to a change in capacitance that is measured to determine the acceleration acting on the system along the axis of the accelerometer (shown in Figure 2.2a).



(a) : The working principle of differential capacitance MEMS accelerometer. C_i denotes the capacitance, d_i denotes the distance between the opposing plates. When the system accelerates, the inner mass is displaced, resulting in a change in d_i, which changes the difference between C_i.



(b) : The working principle of differential capacitance MEMS gyroscope. The red frame constantly oscillates. When an angular rate ω is applied to the system, the occurring Coriolis force F_C displaces the inner mass that results in a change of capacitance described in (a).

Figure 2.2: The working principles of accelerometer (a) and gyroscope (b) forming the MEMS IMU.

Similarly, the MEMS gyroscope uses the principle of differential capacitance to measure acceleration caused by the Coriolis force, which occurs when an external angular rate is applied to a moving body (shown in Figure 2.2b).

Accelerometers and gyroscopes, which constitute an Inertial Measurement Unit (IMU), are prone to substantial time and temperature drifts, resulting in

non-negligible errors over time. A drift occurs,e.g., when the displaced inner mass does not return to its original position entirely, leading to offset in the measurements. However, accelerometers and gyroscopes have a fast response time, allowing for high sample rates. Despite their potential for inaccuracy over time, IMUs are still a more suitable choice for estimating the position of a robot operating on rough terrain compared to wheel odometry and its drawbacks. Additionally, IMU position estimates can be combined with other types of odometry, such as GPS [2], to provide a more accurate estimate.

Laser scanner

Laser scanners, also known as lidar (acronym of Light Detection and Ranging), are well-established sensors in mobile robotics [3] that use Time of Flight (ToF) method to measure the distance to objects. This method involves the use of laser to emit light, which is reflected off of objects and detected by the lidar. The distance to the object is determined by half the distance that the light travels form the laser to the photodetector.

There are two main categories of lidar: solid state and electromechanical. The electromechanical lidars can be further divided into single plane and multi plane, depending on their vertical and horizontal field of view and their ability to provide 3D scans of their surroundings. While single plane lidars are sufficient for navigating and detecting obstacles on flat surfaces, robotic platforms operating in uneven and outdoor environments often require lidar capable of 3D scanning.

In conclusion, multi-channel lidars provide dense mapping of the surrounding environment. Furthermore, lidar technology can be used both during the day and at night, and its measurements are not affected by changes in air temperature, unlike sonar [4]. However, lidar measurements become inaccurate when attempting to measure distance to reflective surfaces or to objects at a small angle. Furthermore, the effect of refraction negatively impacts the accuracy of lidar measurements in conditions such as heavy rain or low-hanging clouds. Despite these limitations, lidar remains a robust method for dense terrain mapping, providing a solid foundation for the navigation task, and can also be used for odometry purposes.

Optical camera

Cameras are a common device in mobile robotics, providing a wealth of information about a robot's surroundings. A typical monocular camera consists of a lens assembly that focuses a wide angle of light to create a clear image. At the back of the camera is a CMOS (Complementary Metal Oxide Semiconductor) sensor, which consists of the Bayer mosaic filter and an array of photodiodes that detect the visible light spectrum, and sometimes also infrared. The current pulses are then read row by row to create a digital output, resulting in a final image.

A combination of two monocular cameras, called a stereo camera, can be used to estimate the depth of each pixel. However, this configuration fails

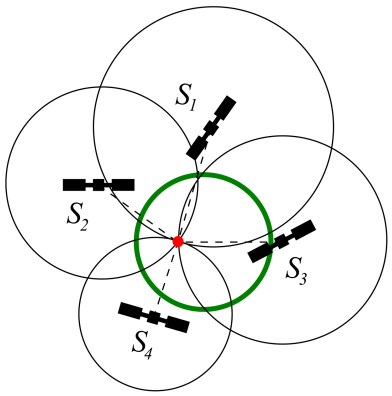
to estimate the depth of monochromatic flat surfaces. This problem can be addressed by adding an infrared dot pattern emitter, which adds features to the captured scene to better register the two images and improve depth estimation.

Despite its popularity in visual odometry [5] and mapping methods for providing dense maps, the use of cameras is limited by their susceptibility to changing light conditions and the high cost of processing their data.

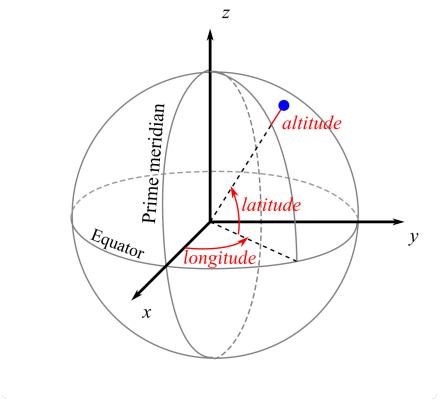
Global Navigation Satellite System (GNSS) receiver

The GNSS receiver is a device used to determine its geographic position by receiving signal transmitted by constelations of satellites [6] (e.g. GPS, Gallileo, and GLONASS).

Transmitted signal consists of messages containing the satellite's pose and a timestamp indicating when the message was sent. Trilateration is then used by receiver to calculate its geographic position based on signal received at least from four satellites (shown in Figure 2.3a). This process involves calculating the distance between each satellite and the receiver based on the time difference between the timestamp and the current time measured by the receiver.



(a) : The estimation of geographic position (red) using trilateration and constelation of four satellites S_i.



(b) : The WGS84 LLA coordinate system. The orthogonal axes denote the Earth-Centered, Earth-fixed system (ECEF).

Figure 2.3: The Global Navigation Satellite System (GNSS).

While satellites use precise atomic clocks, most GNSS receivers implement time measurement using quartz crystals, which introduce errors in the calculated distances from satellites. To minimize this error, four satellites are typically used instead of the three that would be theoretically sufficient. However, using a single GNSS receiver still results in a sub-meter level of accuracy at best, due to errors that arise when the signal is propagated through the atmosphere.

The Real-Time Kinematic (RTK) method enhances the estimated position to centimeter-level accuracy by adding a second, stationary receiver, which

communicates with the moving one. By using this combination of receivers, the atmosphere propagation errors are significantly reduced. In addition, RTK achieves centimeter-level accuracy through the use of advanced correction algorithms, which are outside the scope of this work.

This thesis utilizes the World Geodetic System (WGS84) LLA (latitude, longitude, altitude) coordinates (see Figure 2.3b), also known as geodetic coordinates, to determine the geographic position of a given point. The origin of the coordinate system is located at the center of mass of the Earth. Latitude represents the angle between a point and the Earth's equator, while longitude is the angle between a point and the Prime Meridian. Altitude is defined as the height above mean sea level. The WGS84 LLA coordinate system is widely used, including by the Global Positioning System (GPS).

One of the limitations of using Global Navigation Satellite System (GNSS) to estimate the pose of a robot for navigation purposes is a significant deviation in latitude and longitude and poor accuracy of altitude estimation due to errors in estimating distances to satellites. Additionally, GNSS is often unreliable in urban and forest environments [2], making it unsuitable for continuous pose estimation. However, GNSS can be used in combination with other types of odometry to correct for drift over time.

Chapter 3

Review of three-dimensional SLAM methods

3.1 Approaches

In [7], three main paradigms for SLAM have been observed: Extended Kalman Filter (EKF), Particle Filters (PF), and Graph-based methods. The first two approaches can be broadly classified as probabilistic methods. The graph-based methods can be classified as non-linear methods.

EKF methods

EKF-based methods represent the estimated robot and landmarks in the environment as a state vector. The error covariance matrix, which stores uncertainties in the state estimates, is updated along with the state vector using EKF algorithm and grows with the area explored by a robot. However, the number of values in the covariance matrix increases quadratically, leading to computational inefficiency.

PF methods

Particle Filters methods estimate the real state of the robot and observed landmarks by taking a number of samples, each representing a guess of a real state [7]. The probabilistic methods, in general, do not incorporate loop closure, which can reduce over-time cumulative errors.

Graph methods

Graph-based techniques [8] aim to find a solution to the SLAM problem by optimizing a graph consisting of vertices representing, typically, robot pose estimates and, in some cases, landmarks. The edges connecting exactly two vertices represent odometry between two robot poses, or for example, range measurement between a robot pose and an observed landmark. One popular approach is the use of a factor graph, a type of bibartite graph. This representation distinguishes between two types of nodes: variables, usually representing robot pose estimates and observed landmarks, and factors, which

represent constraints in the form of functions that relate two variables. The edges of a factor graph always connect a variable with a factor. Non-linear optimization methods are then applied to the factor graph to minimize a certain error function. Loop closure can be integrated to the factor graph by adding loop-closure factors between two variables with the same information. The advantages of graph-based methods over probabilistic approaches include efficient updates and linear memory usage, as well as the ability to reduce cumulative errors through loop closure, resulting in more accurate estimates and the ability to cover larger environments.

State-of-the-art graph-based SLAM scheme

The classical framework of many recent graph-based SLAM techniques consists of three main components [9]: an odometer link, an optimization link, and a closed-loop detection link. The odometer link uses sensor data, such as lidar, camera, or IMU measurements, to estimate poses of the robot bonded with factors, thereby constructing a graph of poses. These types of odometry are referred to as lidar odometry, visual odometry and inertial odometry, respectively. The scheme of a typical graph-based SLAM is shown in Figure 3.1.

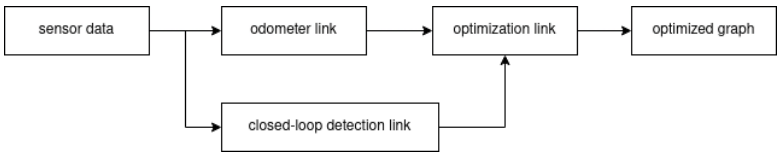


Figure 3.1: A classic framework of the graph-based SLAM method.

Further division of SLAM methods

State-of-the-art SLAM methods using different types of sensors can be further divided into tightly and loosely coupled approaches [10]. The following explanation incorporates the use of lidar and IMU measurements. Loosely coupled methods typically incorporate IMU measurements to assist lidar odometry, but are not involved in the optimization process. Tightly coupled systems, on the other hand, combine inertial and lidar measurements together for state estimation, often by adding different types of factors to the factor graph.

SLAM methods can be also categorized according to the used types of sensors as lidar and visual SLAM. Thanks to the above-stated advantages of the lidar, for example the robustness to variations in illumination and weather conditions [11], this thesis will prefer the use of lidar SLAM method over visual SLAM. State-of-the-art lidar SLAM methods based on non-linear optimization are summarized in the next section.

3.2 Lidar SLAM methods

3.2.1 Loosely coupled methods

Beginning with the loosely coupled methods, in [12], Zhang and Singh introduced a lidar odometry and mapping scheme called LOAM, which has become a foundation for latter subsequent methods. LOAM uses a scanning device consisting of a 2D lidar, constantly rotating in 3D space, resulting in a distorted point cloud when the device is in motion. The scheme consists of three modules: feature extraction, lidar odometry, and lidar mapping [13]. Lidar odometry is performed through feature matching rather than the computationally expensive matching of entire lidar scans. These features are extracted from scans based on local roughness, and classified as edge or planar features. The lidar mapping module takes a set of multiple lidar frames, combined using lidar odometry, which is registered and merged with the global map. Optionally, IMU measurements can be incorporated to provide an initial guess for lidar odometry scan matching. However, this method suffers from drifts during long-during mapping due to the lack of loop closure detection.

The later A-LOAM [14] presents a computationally optimized version of LOAM, using lidar measurements only.

SC-A-LOAM, presented in [15], builds on the A-LOAM implementation and adds loop closure through scan context (SC) [16]. In SC, a point cloud is divided into azimuthal and radial bins, each containing the maximum height of all points within the bin. These bins are then arranged into a matrix to form a robot-centric global point cloud descriptor. A loop is detected based on similarity score, represented as the normalized sum of cosine distances between collumn vectors of two descriptors.

F-LOAM, presented in [17], employs the feature extraction method from LOAM. However, the lidar odometry process is accelerated by replacing the iterative motion estimation from LOAM with the extrapolation of the previous two pose estimates.

In [18], Shan and Englot proposed a ground-optimized version of LOAM, called LeGO-LOAM, which is suitable for ground vehicles. This method relies on the presence of a ground plane, which is segmented from the point cloud. The edge and planar feature extraction is the same as in the LOAM method. Lidar odometry scan matching is performed through a two-step Levenberg-Marquardt optimization process. The first step matches only planar features, estimating the z coordinate, *roll*, and *pitch* of the robot pose. The second step incorporates estimates from the first step and matches edge features, resulting in estimates of x, y coordinates and yaw. As with LOAM, IMU measurements can be incorporated to provide an initial guess for feature matching. LeGO-LOAM employs pose-graph optimization using the iSAM2 [19] algorithm with loop detection based on the Euclidean distance between two poses.

LeGO-LOAM-BOR [20] represents a computationally optimized version

of LeGO-LOAM that has been enhanced of scan context loop detection, resulting in SC-LeGO-LOAM [21].

In [13], Chen et al. proposed a method called R-LIO that uses a multi-channel rotating lidar to obtain spherical coverage of the surroundings. Additionally, IMU measurements are incorporated to de-skew point clouds and also serve as an initial guess for lidar odometry, which utilizes frame-to-submap matching with the feature matching method from LOAM. Loop detection is based on submap-to-submap Iterative Closest Point (ICP) [22] matching.

3.2.2 Tightly coupled methods

The HDL Graph SLAM method, presented in [23], estimates lidar trajectory through iterative NDT scan-matching and uses a pose graph for map optimization with constraints, namely: IMU gravity vector direction, IMU orientation, GPS measurements. The method also detects the floor plane using the RANSAC [24] algorithm and includes it as a constraint. Loop closure is achieved through NDT scan-matching between selected candidates based on the Euclidean distance between two poses. The graph optimization is performed using Gauss-Newton method.

The LIO-M method, presented in [25], consists of two parallel components: a tightly coupled lidar-IMU odometry that optimizes all pose estimates within a local window, and a second component that aligns lidar scans to the global map using information from the optimized poses.

In [10], Shan et al. presented LIO-SAM scheme, which uses a factor graph optimized with the iSAM2 algorithm and incorporates four factors: the IMU pre-integration factor, the lidar odometry factor, the GPS factor, and the loop-closure factor. The IMU pre-integration estimates the motion of lidar with high frequency and serves as an initial guess for the feature-based scan matching in the lidar odometry, which uses the feature extraction method from LOAM. The method also employs sliding window approach, in which only the features from the last N scans are used in feature matching process, resulting in faster lidar odometry. Loop detection is based on the Euclidean distance between two poses.

The SC-LIO-SAM method [26] enhances the LIO-SAM method with a more sophisticated loop detection using scan context.

The intensity and ambient value of points was leveraged in feature extraction process from LOAM to remove unnecessary feature points, resulting in IA-LIO-SAM method [27].

In [28], Wang et al. proposed the LIO-CSI method, which is built on the LIO-SAM method and integrates semantic information to facilitate odometry and loop detection. In the lidar odometry, the semantic information removes dynamic objects, improving accuracy in dynamic environments. The semantic information is obtained using a Sparse Point-Voxel Neural Architecture Search (SPVNAS) [29] deep learning network.

In recent years, there have been several methods that tightly couple multiple lidars, such as M-LOAM [30], and MILIOM [31] that additionally incorporates

IMU measurements.

In [32], Wang et al. proposed D-LIOM method, which can also incorporate multiple lidar inputs. Unlike most previous methods that use feature matching, the D-LIOM registers raw point clouds with a probability map. The method also integrates lidar odometry, IMU pre-integration, and gravity constraint into a local factor graph, forming a submap. Loop detection is performed by matching projected 2D submaps with the help of the RANSAC algorithm.

Characteristics of the above mentioned methods are summarized in the Tab. 3.1.

method	lidar	IMU	GPS	T/L	loop closure	implementation ROS support
LOAM [12]	✓	optional	✗	L	✗	ROS (Kinetic, Melodic)
A-LOAM [14]	✓	✗	✗	-	✗	ROS (Kinetic, Melodic)
SC-A-LOAM [15]	✓	✗	✗	L	scan context	ROS (Melodic)
F-LOAM [17]	✓	✗	✗	-	✗	ROS (Melodic)
LeGO-LOAM [18]	✓	optional	✗	L	ICP scan matching	ROS (Kinetic, Melodic)
LeGO-LOAM-BOR [20]	✓	optional	✗	L	ICP scan matching	ROS (Kinetic)
SC-LeGO-LOAM [21]	✓	optional	✗	L	scan context	ROS(Kinetic)
R-LIO [13]	✓	✓	✗	L	submap-to-submap ICP matching	✗
HDL Graph SLAM [23]	✓	optional	optional	T	NDT scan matching	ROS (Melodic, Noetic)
LIO-M [25]	✓	✓	✗	T	✗	ROS (Kinetic, Melodic)
LIO-SAM [10]	✓	✓	optional	T	keyframe-to-keyframe ICP matching with distance threshold	ROS (Kinetic, Melodic), ROS2 (Foxy)
SC-LIO-SAM [26]	✓	✓	optional	T	scan context keyframe-to-keyframe	ROS (Kinetic, Melodic)
IA-LIO-SAM[27]	✓	✓	optional	T	ICP matching with distance threshold	ROS (Kinetic, Melodic)
LIO-CSI [28]	✓	✓	optional	T	semantic-based	✗
M-LOAM [30]	multiple	✗	✗	T	✗	ROS (Kinetic, Melodic)
MILIOM [31]	multiple	✓	✗	T	✗	✗
D-LIOM [32]	multiple	✓	✗	T	submap-to-submap RANSAC matching	ROS (Noetic)

Table 3.1: Comparison of the reviewed three-dimensional lidar SLAM methods, including used sensors, loop closure method, and ROS support of available implementations. T/L stands for Tightly-coupled/Loosely-coupled sensor fusion.

Chapter 4

Three-dimensional terrain representations for navigation

In the past three decades, several methods for representing 3D terrain have been proposed as they are essential for many navigation systems. The early approaches [33] used 3D cubic grid of equally-sized volumes, called voxels, to discretize the mapped area. However, this method has high memory requirements, is costly to copy when the mapped area needs to be extended, and requires the size of the mapping area to be predetermined.

Another approach is to create the terrain out of direct range measurements in the form of point clouds [34][35]. This allows for easy maintanance and expansion of an existing map by simply appending a vector of points. In [36], point cloud maps are utilized for path planning with ground mobile robot. On the other hand, this representation does not track free and unknown areas and storage of larger ammount of points becomes inefficient with growing dense map. This map representation is shown in Figure 4.1.

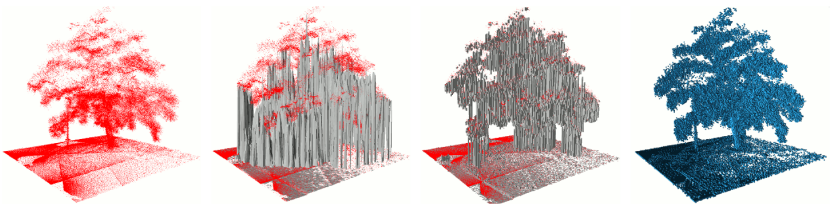


Figure 4.1: The visual comparison of the different map representations. Point cloud, elevation map, multi-level surface map, and voxel map based on the octree structure, respectively. Obtained from [42].

A popular approach to 3D environment modeling is the use of 2.5D maps called digital elevation maps (DEM) (illustrated in Figure 4.1), which are represented as a 2D grid with a height value assigned to each cell [37]. A shortcoming of this method is the inability to capture overhangs occuring in the environment. Although, this representation does not represent the actual environment, thus may pose restrictions to certain applications, overhangs higher than the robot can be ignored, which results in a sufficient terrain approximation for ground mobile robot navigation [38]. The 2.5D approach was enhanced by storing multiple height values for each cell, leading in multi-

level surface (MLS) map (shown in Figure 4.1), introduced in [39] that enables to represent overhangs in general, and multi-level buildings.

However, a more accurate representation of a generic 3D surface provides a voxel approach using octree structure (shown in Figure 4.1), originally proposed in [40]. This method distinguishes between free and unknown space and occupied voxels represent leafs in a tree structure. The octrees allow incremental map building, meaning that the map does not need to be initialized until measurements are integrated. Further advantage of this method is the capability of providing multi-resolution maps based on the specified octree depth. Early works focused on estimating voxel occupancy with boolean approach [41] whereas the most recent methods prefer probabilistic representation [42].

Three-dimensional surface can be represented probabilistically using 3D Normal Distribution Transform (NDT) [43], where each cell of a 3D voxel grid contains three-dimensional normal distribution. Path planning using this method was introduced in [44].

A surface can also be represented with triangular meshes [45][46]. In [47], the mesh surface is utilized for shortest path navigation using vector fields. However, the mesh creation from raw range measurements is computationaly expensive and with new incoming measurements the process has to be repeated. Additionally, a concatenation of two overlapping meshes requires a costly recomputation of significant parts of the meshes.

Chapter 5

Navigation method

5.1 Selected approach

An effective navigation approach relies on accurate robot localization, and mapping, when an environment, in which the robot operates, is not known beforehand. Simultaneous Localization and Mapping (SLAM) techniques address this issue by providing estimates of robot's position in time and generating a map of the environment that can be used for path planning. In this thesis, we decided to incorporate lidar for SLAM due to its advantages over visual approach.

After reviewing various lidar-based SLAM methods (see chapter 3) and assuming the navigation method to be deployed on a middle-sized robot, methods that utilize a single lidar sensor, as it covers most of the robot's surroundings, were considered to be sufficient. However, accumulated errors (i. e. drift) during long-term navigation can result in inaccurate estimates of the robot's position in relation to the actual environment, potentially leading to navigation system failure. SLAM methods that incorporate loop-closure can partially mitigate this problem, and are therefore preferred for integration into the navigation method. We also favor a method with a pre-existing ROS implementation, which would facilitate its integration.

One candidate that meets these criteria is the LeGO-LOAM method and its subsequent versions. However, these types of methods assume a flat ground surface, which may not be present in rough and uneven terrain. Another possibility is the LIO-SAM method, which produces a smaller pose estimation error in a single run compared to HDL Graph SLAM method [48]. In addition, LIO-SAM incorporates IMU measurements and, optionally, GPS measurements, which can be used to set a designated goal, providing an advantage over the SC-A-LOAM method. From the remaining candidates (including versions of LIO-SAM), we selected the pure LIO-SAM method for our navigation system due to its simplicity compared to SC-LIO-SAM and IA-LIO-SAM.

The next part of the navigation system involves the traversability estimation of the mapped terrain. For this purpose, we use a geometric-based approach that estimates traversability based on geometric features, e. g. slope and roughness. While LIO-SAM generates a point cloud map of the environment,

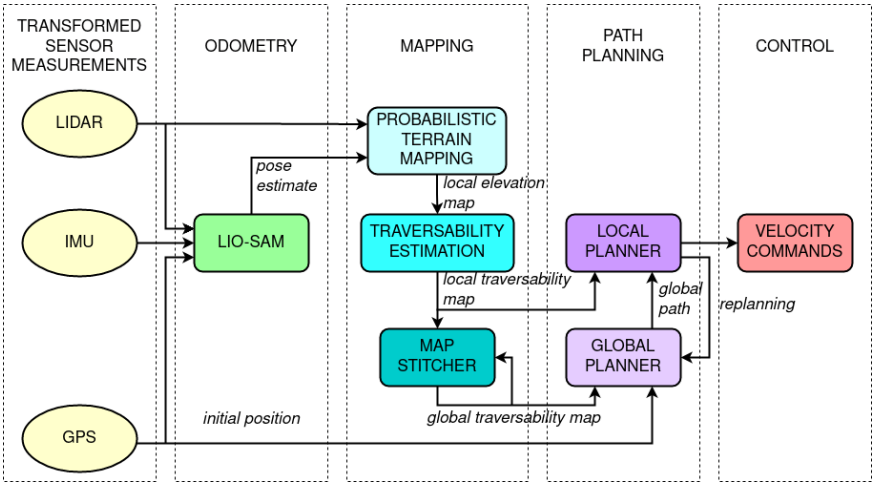


Figure 5.1: Diagram of the proposed navigation method.

it does not provide dense representation of the terrain, which is necessary for precise traversability estimation.Therefore, we only use the provided odometry and utilize the work of Péter Frankhauser et al. [49], which uses the estimated robot poses from LIO-SAM and raw lidar scans to create a robot-centric 2.5D local elevation map of the surrounding terrain. While more memory-efficient map representations such as voxel maps based on octrees (as discussed in chapter 4) exist, the chosen map representation, implemented as 2D array, allows for simplier geometric-based traversability estimation and faster search capability during path planning. The resulting 2.5D local traversability maps are then merged to provide a global map.

With reliable traversability maps available, path planning methods can be used to generate a safe path to a designated goal through the environment. For this, we use a combination of a global and local planner. The global planner employs Dijkstra's algorithm and receives target goals in the form of GPS coordinates. The local planner utilize the Dynamic Window Approach (DWA) algorithm to leverage the robot's motion model in obstacle avoidance and to prompt the global planner to replan the route when an obstacle intersects the current global path. The local planner also generates motion commands for the robot in the form of a twist velocity (v, ω).

A diagram of the designed navigation method is shown in Figure 5.1. The following sections provide more detailed descriptions of each of the aforementioned parts of the navigation system.

5.2 Transformations of sensor measurements

To incorporate measurements from different types of sensors into for example lidar-inertial odometry, the measurements must be taken from a coordinate frame we want to estimate the odometry for, in our case, robot's body frame. Placing sensors in the same location is possible in simulation, however, this approach is not applicable to a physical robot. Therefore, sensors are mounted

on the specific locations on the robot and their data are transformed using homogeneous transformation matrices into robot's body frame (depicted in Figure 5.2). The homogeneous transformation matrix

$$\boldsymbol{T}_{\boldsymbol{B}}^{\boldsymbol{A}} = \left(\begin{array}{c|c} \boldsymbol{R}_{\boldsymbol{B}}^{\boldsymbol{A}} & \boldsymbol{d}_{\boldsymbol{B}}^{\boldsymbol{A}} \\ \hline \mathbf{0} & \mathbf{1} \end{array}\right),$$

represents the transformation of a coordinate frame B with respect to the frame A, and consists of rotation matrix $\boldsymbol{R}_{\boldsymbol{B}}^{\boldsymbol{A}} \in \boldsymbol{SO(3)}$ and translation vector $\boldsymbol{d}_{\boldsymbol{B}}^{\boldsymbol{A}} \in \mathbb{R}^3$.

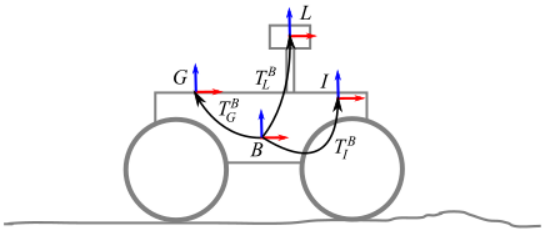


Figure 5.2: The visualization of transformations of sensor coordinate frames into robot's body frame.

The transformation of a point cloud

$$\boldsymbol{P}_{\boldsymbol{L}} = \begin{pmatrix} \boldsymbol{p}_{\boldsymbol{1}_{\boldsymbol{L}}} & \boldsymbol{p}_{\boldsymbol{1}_{\boldsymbol{L}}} & \cdots & \boldsymbol{p}_{\boldsymbol{N}_{\boldsymbol{L}}} \end{pmatrix}$$

of N points

$$\boldsymbol{p}_{\boldsymbol{i}_{\boldsymbol{L}}} = \begin{pmatrix} x_L \\ y_L \\ z_L \\ 1 \end{pmatrix}$$

from lidar frame L to the robot's body frame B, the following transformation is applied:

$$\boldsymbol{P}_{\boldsymbol{B}} = \boldsymbol{T}_{\boldsymbol{L}}^{\boldsymbol{B}} \boldsymbol{P}_{\boldsymbol{L}}.$$

When transforming measurements of the linear acceleration

$$\boldsymbol{a}_{\boldsymbol{I}} = \begin{pmatrix} a_{x_I} \\ a_{y_I} \\ a_{z_I} \end{pmatrix}$$

from IMU frame I to the robot's body frame B, the acceleration of origin of B with respect to the coordinate frame of I is

$$\boldsymbol{a} = \boldsymbol{a}_{\boldsymbol{I}} + \frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} \times \boldsymbol{r}_{\boldsymbol{B}}^{\boldsymbol{I}} + \boldsymbol{\omega}_{\boldsymbol{I}} \times (\boldsymbol{\omega}_{\boldsymbol{I}} \times \boldsymbol{r}_{\boldsymbol{B}}^{\boldsymbol{I}}),$$

where

$$\boldsymbol{\omega}_{\boldsymbol{I}} = \begin{pmatrix} \omega_{x_I} \\ \omega_{y_I} \\ \omega_{z_I} \end{pmatrix}$$

is angular velocity of the robot with respect to the I frame, and $\boldsymbol{r}_{\boldsymbol{B}}^{\boldsymbol{I}}$ is the translation vector between I and B with respect to the I coordinate frame. Then a relative rotation between I and B frame is applied to obtain acceleration values in B frame:

$$\boldsymbol{a_B} = \boldsymbol{R}_{\boldsymbol{I}}^{\boldsymbol{B}} \boldsymbol{a}.$$

The same rotation is applied to obtain angular velocity vector:

$$\boldsymbol{\omega_B} = \boldsymbol{R}_{\boldsymbol{I}}^{\boldsymbol{B}} \boldsymbol{\omega_I}.$$

When transforming GPS WGS84 LLA data, the measurements are first transformed into Cartesian coordinates using Equirectangular projection [50]:

$$x_G = R(\lambda - \lambda_0)\cos\phi_1, \quad y_G = R(\phi - \phi_0), \quad z_G = \theta$$

where λ is the longitude, ϕ is the latitude, θ is the altitude, ϕ_0 is the central parallel, ϕ_1 are the standard parallels, λ_0 is the central meridian, and R is the radius of the globe. Then a homogeneous transformation is applied to obtain projected measurements in the B frame:

$$\begin{pmatrix} x_B \\ y_B \\ z_B \\ 1 \end{pmatrix} = \boldsymbol{T}_{\boldsymbol{G}}^{\boldsymbol{B}} \begin{pmatrix} x_G \\ y_G \\ z_G \\ 1 \end{pmatrix}.$$

5.3 Lidar-inertial odometry via Smoothing and Mapping (LIO-SAM)

The LIO-SAM is a tightly-coupled lidar-inertial odometry technique that iutilizes a factor graph containing one type of variable representing the state of the robot and four factors: IMU preintegration, lidar odometry, GPS, and loop closure. The factor graph is optimized using the iSAM2 algoritm [19] upon the incorporation of a new robot state. An overview illustrating the addition of measurements and factors (described further) is depicted in Figure 5.3. The method uses sensor measurements that are transformed according to the previous chapter.

The state of the robot, which is represented by its body frame B, is expressed as [10]

$$\boldsymbol{x} = \left(\boldsymbol{R}^{\boldsymbol{T}},\ \boldsymbol{p}^{\boldsymbol{T}},\ \boldsymbol{v}^{\boldsymbol{T}},\ \boldsymbol{b}^{\boldsymbol{T}}\right)^{\boldsymbol{T}}, \tag{5.1}$$

where $\boldsymbol{R}$ represents the rotation matrix, $\boldsymbol{p}$ is the position vector, $\boldsymbol{v}$ is the speed, and $\boldsymbol{b}$ represents the IMU bias. The transformation from B to the world frame W is represented as

$$\boldsymbol{T} = \left(\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{p} \\ \hline 0 & 1 \end{array}\right) \ni \boldsymbol{SE(3)}.$$

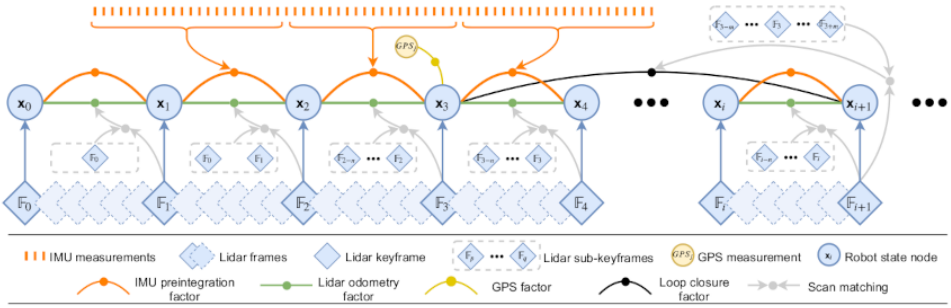


Figure 5.3: The structure LIO-SAM, utilizing a factor graph, along with the incorporation of sensor measurements. Obtained from [10].

The IMU preintegration adops the method described in [51], which defines the angular velocity and acceleration measurements from the IMU at time t as

$$\hat{\boldsymbol{\omega}}_t = \boldsymbol{\omega}_t + \boldsymbol{b}_t^{\omega} + \boldsymbol{n}_t^{\omega}, \tag{5.2}$$

$$\hat{\boldsymbol{a}}_t = \boldsymbol{R}_t^{BW}(\boldsymbol{a}_t - \boldsymbol{g}) + \boldsymbol{b}_t^{a} + \boldsymbol{n}_t^{a}, \tag{5.3}$$

where $\hat{\boldsymbol{\omega}}_t$ and $\hat{\boldsymbol{a}}_t$ represent the raw IMU measurements in B. $\boldsymbol{b}_t$ is the bias and $\boldsymbol{n}_t$ denotes white noise. The rotation matrix from W to B is represented by $\boldsymbol{R}_t^{BW}$ and the gravity vector in W is denoted by $\boldsymbol{g}$.

The IMU preintegration method in [51] estimates the motion of the robot in terms of velocity, position, and orientation at time $t + \Delta t$, where Δt is the sampling period of the IMU, as

$$\boldsymbol{v}_{t+\boldsymbol{\Delta t}} = \boldsymbol{v}_t + \boldsymbol{g}\Delta t + \boldsymbol{R}_t^{\boldsymbol{WB}}(\hat{\boldsymbol{a}}_t - \boldsymbol{b}_t^{a} - \boldsymbol{n}_t^{a})\Delta t, \tag{5.4}$$

$$\boldsymbol{p}_{t+\boldsymbol{\Delta t}} = \boldsymbol{p}_t + \boldsymbol{v}_t\Delta t + \frac{1}{2}\boldsymbol{g}\Delta t^2 + \frac{1}{2}\boldsymbol{R}_t^{\boldsymbol{WB}}(\hat{\boldsymbol{a}}_t - \boldsymbol{b}_t^{a} - \boldsymbol{n}_t^{a})\Delta t^2, \tag{5.5}$$

$$\boldsymbol{R}_{t+\boldsymbol{\Delta t}}^{\boldsymbol{WB}} = \boldsymbol{R}_t^{\boldsymbol{WB}}\mathbf{exp}((\hat{\boldsymbol{\omega}}_t - \boldsymbol{b}_t^{\omega} - \boldsymbol{n}_t^{\omega})\Delta t) \tag{5.6}$$

under the assumption of constant $\boldsymbol{\omega}_t$ and $\boldsymbol{a}_t$ during the integration. According to [51], the relative motion of B between two timesteps i and j can be written as:

$$\Delta\boldsymbol{v}_{ij} = \boldsymbol{R}_i^{BW}(\boldsymbol{v}_j - \boldsymbol{v}_i - \boldsymbol{g}\Delta t_{ij}), \tag{5.7}$$

$$\Delta\boldsymbol{p}_{ij} = \boldsymbol{R}_i^{BW}(\boldsymbol{p}_j - \boldsymbol{p}_i - \boldsymbol{v}_i\Delta t_{ij} - \frac{1}{2}\boldsymbol{g}\Delta t_{ij}^2), \tag{5.8}$$

$$\Delta\boldsymbol{R}_{ij} = \boldsymbol{R}_i^{BW}\boldsymbol{R}_j^{WB}. \tag{5.9}$$

The lidar odometry utilizes the feature extraction method from [12], which evaluates the smoothness of incoming lidar points over a local region as:

$$c = \frac{1}{|S| \cdot ||\boldsymbol{p}_{i,j}||}||\sum_{k \in S, k \neq j}(\boldsymbol{p}_{i,j} - \boldsymbol{p}_{i,k})||, \tag{5.10}$$

where $\boldsymbol{p}_{i,j}$ is a j-th point in i-th lidar scan and S represents a set of consecutive points of $\boldsymbol{p}_{i,j}$. By applying lower and upper threshold to c values, edge F_i^e and

planar F_i^p features can be obtained, forming a lidar frame $\mathbb{F}_i = \{F_i^e, F_i^p\}$. The authors introduce the concept of keyframes, which represent a new robot state $\boldsymbol{x_i}$ along with the associated lidar frame $\mathbb{F}_i$ that is added to the factor graph when a specific threshold on the robot's change in pose is exceeded. In order to enhance the real-time performance of the method, a sliding-window approach is implemented, which incorporates features of the N most recent keyframes $\{\mathbb{F}_{i-N}, \ldots, \mathbb{F}_i\}$, referred to as sub-keyframes, which are transformed into W using the associated transformations $\{\boldsymbol{T_{i-N}^{WB}}, \ldots, \boldsymbol{T_i^{WB}}\}$.

Two separate voxel maps M_i^e, M_i^p containing edge and planar features, respectively, are created from the sub-keyframes and downsampled to eliminate duplicate features. A newly-obtained keyframe $\mathbb{F}_{i+1}$ is then transformed into W, using the initial transformation $\boldsymbol{T_{i+1}^{WB}}$ produced from the IMU preintegration step, and is matched to the voxel maps using a scan-matching technique.

The scan-matching process finds the closest correspondences of $\mathbb{F}_{i+1}^e$, $\mathbb{F}_{i+1}^p$ in the voxel maps M_i^e, M_i^p respectively. During the scan-matching process, the closest edges and planar patches from M_i^e and M_i^p are found for each of $\mathbb{F}_{i+1}^e$ and $\mathbb{F}_{i+1}^p$ features, respectively, and the distances between these correspondences are calculated according to [12] as:

$$\boldsymbol{d_{e_k}} = \frac{|(\boldsymbol{p_{i+1,k}^e} - \boldsymbol{p_{i,u}^e}) \times (\boldsymbol{p_{i+1,k}^e} - \boldsymbol{p_{i,v}^e})|}{|\boldsymbol{p_{i,u}^e} - \boldsymbol{p_{i,v}^e}|}, \tag{5.11}$$

$$\boldsymbol{d_{p_k}} = \frac{\begin{vmatrix} (\boldsymbol{p_{i+1,k}^p} - \boldsymbol{p_{i,u}^p}) \\ (\boldsymbol{p_{i,u}^p} - \boldsymbol{p_{i,v}^p}) \times (\boldsymbol{p_{i,u}^p} - \boldsymbol{p_{i,w}^p}) \end{vmatrix}}{|(\boldsymbol{p_{i,u}^p} - \boldsymbol{p_{i,v}^p}) \times (\boldsymbol{p_{i,u}^p} - \boldsymbol{p_{i,w}^p})|}, \tag{5.12}$$

where $\boldsymbol{d_{e_k}}$ denotes the distance between an edge feature $\boldsymbol{p_{i+1,k}^p} \in \mathbb{F}_{i+1}^e$ and an edge formed by two edge features $\boldsymbol{p_{i,u}^e}, \boldsymbol{p_{i,v}^e} \in M_i^e$, and $\boldsymbol{d_{p_k}}$ denotes the distance between a planar feature $\boldsymbol{p_{i+1,k}^p} \in \mathbb{F}_{i+1}^p$ and a planar patch formed by three planar features $\boldsymbol{p_{i,u}^p}, \boldsymbol{p_{i,v}^p}, \boldsymbol{p_{i,w}^p} \in M_i^p$.

Minimizing the sum of these distances by adjusting $\boldsymbol{T_{i+1}^{WB}}$ using the Gauss-Newton method yields the optimal transformation between the last two robot states [10],

$$\Delta \boldsymbol{T}_{i,i+1} = \boldsymbol{T_i^{BW}} \boldsymbol{T_{i+1}^{WB}}, \tag{5.13}$$

which is added as a lidar odometry factor to the factor graph. The result of lidar odometry is subsequently used to estimate the IMU bias.

The GPS factor incorporates absolute measurements to address the issue of drift during long-term navigation. A new GPS factor is only added if the covariance of the estimated robot position exceeds the covariance of the GPS measurement.

Loop closure is performed upon the addition of a new state $\boldsymbol{x_{i+1}}$ to the factor graph. All prior states within a specified Euclidean distance relative to $\boldsymbol{x_{i+1}}$ are searched. Using the scan-matching process from lidar odometry, a new keyframe $\mathbb{F}_{i+1}$ is matched with all sub-keyframes $\{\mathbb{F}_{j-m}, \ldots, \mathbb{F}_j, \ldots, \mathbb{F}_{j+m}\}$, $m \in \mathbb{N}$, corresponding to each of the searched

candidates $\boldsymbol{x_j}$. The obtained relative transformations $\Delta \boldsymbol{T_{j,i+1}}$ are added to the factor graph as loop closure factors.

5.4 Probabilistic terrain mapping

The probabilistic terrain mapping method, as described in [49], generates probabilistic terrain estimates in the form of a grid-based robot-centric elevation map. This method was originally intended for use with relative proprioceptive-based localization, such as wheel odometry. The method distinguishes between two types of map updates: *map update from range measurements*, which updates the map with new data, and *map update from robot motion*, which propagates the uncertainty of robot pose estimates into the robot-centric map.

The notation in the following definitions may differ from [49] in order to conform to the notation used in this thesis. The authors introduce four coordinate frames (illustrated in Figure 5.4): the world-fixed frame W, the robot body frame B, the sensor frame (in our case, the lidar frame) L, and the map frame M. The method assumes the existence of a known static transformation $\boldsymbol{T_L^B}$ and an available pose estimation $\boldsymbol{T_B^W}$, where the rotation matrix $\boldsymbol{R_B^W}$ can be written as

$$\boldsymbol{R_B^W} = \boldsymbol{R_{\tilde{B}}^W}(\psi) \cdot \boldsymbol{R_B^{\tilde{B}}}(\theta, \phi), \tag{5.14}$$

where $\boldsymbol{R_{\tilde{B}}^W}(\psi)$ represents the rotation of the B frame about ψ around the vertical axis e_z^W, and $\boldsymbol{R_B^{\tilde{B}}}(\theta, \phi)$ represents the rotation between $\tilde{B}$ and B with pitch θ and roll ϕ angles. The transformation $\boldsymbol{T_M^B}$ is user-defined, with the rotational matrix $\boldsymbol{R_M^B}$ such that e_z^W and e_z^M are aligned, and the yaw angle between W and M is equal to the yaw angle between W and B. The position of a map grid cell is defined as $P_i = (x_i, y_i, \hat{h}_i)$, where x_i and y_i denote the horizontal position, and $\hat{h}_i$ represents the estimated height of the terrain.

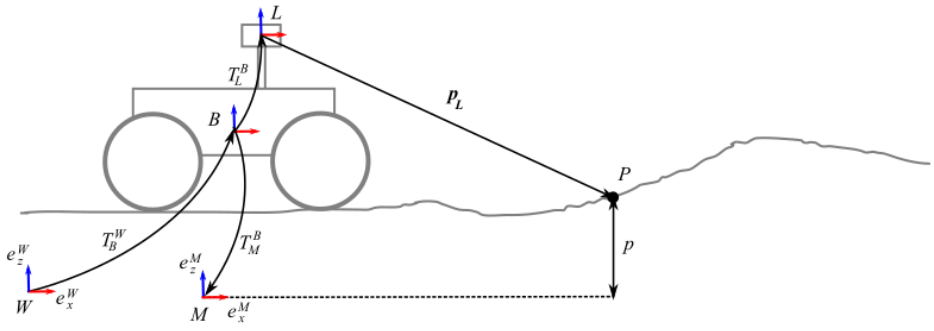


Figure 5.4: The illustration of frames used in the probabilistic mapping, where W is the world frame, B is the robot body frame, L is the lidar frame, and M is the local elevation map frame. The p indicates the projected height of a point P, obtained from the range measurement $\boldsymbol{p_L}$, in the M frame. Inspired by [49].

A newly-obtained point $\boldsymbol{p_L}$ is transformed into the M frame, resulting in a new height measurement p:

$$p = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \boldsymbol{T_L^M} \begin{pmatrix} \boldsymbol{p_L}^T & 1 \end{pmatrix}^T, \tag{5.15}$$

which is approximated by a Gaussian distribution $\tilde{p} \sim \mathcal{N}(p, \sigma_p^2)$. The variance σ_p^2 is obtained from:

$$\sigma_p^2 = \boldsymbol{J_L} \boldsymbol{\Sigma_L} \boldsymbol{J_L^T} + \boldsymbol{J_R} \boldsymbol{\Sigma}_{\boldsymbol{R_L^W}} \boldsymbol{J_R^T}, \tag{5.16}$$

where the $\boldsymbol{J_L}$ and $\boldsymbol{J_R}$ are Jacobians for the lidar measurement and the L frame rotation:

$$\boldsymbol{J_L} = \frac{\partial p}{\partial \boldsymbol{p_L}}, \quad \boldsymbol{J_R} = \frac{\partial p}{\partial \boldsymbol{R_M^L}}. \tag{5.17}$$

The $\boldsymbol{\Sigma_L}$ denotes the covariance matrix of the lidar model, and $\boldsymbol{\Sigma}_{\boldsymbol{R_L^W}}$ represents the covariance matrix of the sensor rotation.

The method employs a one-dimensional Kalman filter to fuse the new measurement $(\tilde{p}, \sigma_p^2)$ into the current estimate $(\hat{h}_{i-1}, \sigma_{\hat{h}_{i-1}}^2)$, resulting in a new estimate $(\hat{h}_i, \sigma_{\hat{h}_i}^2)$ [49]:

$$\hat{h}_i = \frac{\sigma_p^2 \hat{h}_{i-1} + \sigma_{\hat{h}_{i-1}}^2 \tilde{p}}{\sigma_p^2 + \sigma_{\hat{h}_{i-1}}^2}, \quad \sigma_{\hat{h}_i}^2 = \frac{\sigma_{\hat{h}_{i-1}}^2 \sigma_p^2}{\sigma_{\hat{h}_{i-1}}^2 + \sigma_p^2}. \tag{5.18}$$

Because the map is formulated in a robot-centric manner, it is necessary to update the map, i. e. $\hat{h}$ and $\sigma_{\hat{h}}^2$, upon a motion of the robot with respect to the W frame. The approach chosen in [49] involves extending each cell i with the spatial covariance matrix:

$$\boldsymbol{\Sigma_{P_i}} = \begin{pmatrix} \sigma_{x,min}^2 & 0 & 0 \\ 0 & \sigma_{y,min}^2 & 0 \\ 0 & 0 & \sigma_{\hat{h}_i}^2 \end{pmatrix}, \tag{5.19}$$

where $\sigma_{\hat{h}_i}^2$ is calculated using the aforementioned Kalman filter, and $\sigma_{x,min}^2$, $\sigma_{y,min}^2$ are initialized with $\left(\frac{d}{2}\right)^2$, where d denotes the side length of the cell, i. e. resolution.

For time $k+1$, the estimated position $\hat{\boldsymbol{r}}_{\boldsymbol{M_{k+1}P}}$ of a point P in the coordinate frame M_{k+1}, associated with the estimated robot frame $\tilde{B}_{k+1}$, can be written with respect to M_{k+1} as [49]:

$$\begin{aligned} \hat{\boldsymbol{r}}_{\boldsymbol{M_{k+1}P_i}} = -\boldsymbol{r}_{\tilde{\boldsymbol{B}}_{\boldsymbol{k+1}}\boldsymbol{M_{k+1}}} - \left(\boldsymbol{R}_{\boldsymbol{M_{k+1}}}^{\tilde{\boldsymbol{B}}_{\boldsymbol{k+1}}}\right)^{-1} \hat{\boldsymbol{r}}_{\tilde{\boldsymbol{B}}_{\boldsymbol{k}}\tilde{\boldsymbol{B}}_{\boldsymbol{k+1}}} \\ + \left(\hat{\boldsymbol{R}}_{\boldsymbol{M_{k+1}}}^{\boldsymbol{M_k}}\right)^{-1} \left(\boldsymbol{r}_{\tilde{\boldsymbol{B}}_{\boldsymbol{k}}\boldsymbol{M_k}} + \hat{\boldsymbol{r}}_{\boldsymbol{M_k P_i}}\right) \end{aligned} \tag{5.20}$$

The propagation of $\boldsymbol{\Sigma_{P_i,k}}$ for $\boldsymbol{r}_{\boldsymbol{M_k P_i}} \sim \mathcal{N}(\hat{\boldsymbol{r}}_{\boldsymbol{M_k P_i}}, \boldsymbol{\Sigma_{P_i}}, \boldsymbol{k})$ can be written as [49]

$$\boldsymbol{\Sigma_{P_i,k+1}} = \boldsymbol{\Sigma_{P_i,k}} + \boldsymbol{J_r} \boldsymbol{\Sigma_r} \boldsymbol{J_r^T} + \boldsymbol{J_R} \boldsymbol{\Sigma_R} \boldsymbol{J_R^T}, \tag{5.21}$$

where $\boldsymbol{\Sigma_r}$ and Σ_R represent the uncertainty of robot pose estimate between reference frames $\tilde{B}_k$ and $\tilde{B}_{k+1}$ with [49]:

$$\boldsymbol{r}_{\tilde{\boldsymbol{B}}_k,\tilde{\boldsymbol{B}}_{k+1}} \sim \mathcal{N}(\hat{\boldsymbol{r}}_{\tilde{\boldsymbol{B}}_k,\tilde{\boldsymbol{B}}_{k+1}}, \boldsymbol{\Sigma_r}), \quad \boldsymbol{R}^{\tilde{\boldsymbol{B}}_k}_{\tilde{\boldsymbol{B}}_{k+1}} \sim \mathcal{N}(\hat{\boldsymbol{R}}^{\tilde{\boldsymbol{B}}_k}_{\tilde{\boldsymbol{B}}_{k+1}}, \boldsymbol{\Sigma_R}). \tag{5.22}$$

The Jacobians $\boldsymbol{J_r}$, $\boldsymbol{J_R}$ from 5.21 are calculated as [49]:

$$\boldsymbol{J_r} = \frac{\partial \hat{\boldsymbol{r}}_{\boldsymbol{M_{k+1}P_i}}}{\partial \hat{\boldsymbol{r}}_{\tilde{\boldsymbol{B}}_k\tilde{\boldsymbol{B}}_{k+1}}}, \quad \boldsymbol{J_R} = \frac{\partial \hat{\boldsymbol{r}}_{\boldsymbol{M_{k+1}P_i}}}{\partial \hat{\boldsymbol{R}}^{\tilde{\boldsymbol{B}}_k}_{\tilde{\boldsymbol{B}}_{k+1}}} \tag{5.23}$$

The height estimates $\hat{h}_i$ for the cells (x_i, y_i) are stored in a matrix that corresponds to the size of the elevation map. This matrix is then passed on to the traversability estimation method, which is described in the next section.

5.5 Traversability estimation

The navigation method employs a geometric-based approach to estimate terrain traversability, which involves evaluating the geometric properties of a surface - specifically, slope and roughness - to determine its suitability for traversal. To accomplish this, the method retrieves an elevation map

$$\underset{m_E \times n_E}{\boldsymbol{E}}, \text{ where } m_E, n_E \in \mathbb{N},$$

with resolution $s \in \mathbb{R}_{>0}$, and applies a series of mathematical expressions (as depicted in Figure 5.5). The traversability of each cell, corresponding to an entry of $\boldsymbol{E}$, is calculated as a normalized weighted sum of slope and roughness. Additionally, cells classified as unknown, i. e. corresponding entries of $\boldsymbol{E}$ that contain a NaN (Not a Number) value, within a defined radius $r_o \in \mathbb{R}_{>0}$ around the robot are treated as obstacles. This prevents the robot from encountering potentially hazardous situations, such as the risk of falling off a cliff. The following paragraphs provide a more in-depth analysis of each aspect of the traversability estimation process.

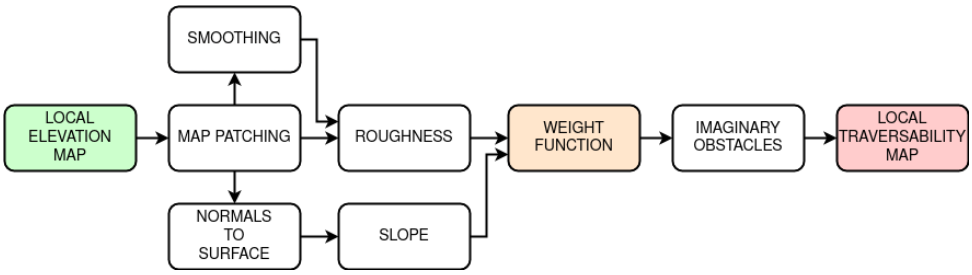


Figure 5.5: The process of traversability estimation, resulting in the local traversability map.

Let

$$S(\boldsymbol{E}) = \{e_{i,j} \in \mathbb{R} \,|\, i \in \{1, \ldots, m_E\} \wedge j \in \{1, \ldots, n_E\}\}$$

be the set of all entries of $\boldsymbol{E}$. Henceforth, we will be referring to $\boldsymbol{E}$ as elevation layer.

Map patching

An elevation layer that has been retrieved often includes isolated cells with unknown values, which can cause difficulties in path planning. However, if the resolution of the elevation layer is small, compared to the robot's wheel diameter, it is possible to approximate the values of these isolated cells using the surrounding elevation values without posing a risk to the robot. The neighboring cells, within the radius $r_m \in \mathbb{R}_{>0}$, of an unknown cell (i, j):

$$N_{i,j} = \left\{ n_{k,l} \,|\, n_{k,l} \in S(\boldsymbol{E}) \wedge (i-k)^2 + (j-l)^2 \leq \left(\frac{r_m}{s}\right)^2 \wedge (k \neq i \wedge l \neq j) \right\}$$

are utilized to approximate the elevation of the unknown cell with the median:

$$p_{i,j} = \frac{1}{2} \sum_{n_{k,l} \in N_{i,j}} n_{k,l}. \tag{5.24}$$

Let

$$\underset{m_E \times n_E}{\boldsymbol{P}}$$

containing $p_{i,j}$ on the (i, j)-th position, be a patch layer. The

$$S(\boldsymbol{P}) = \{ p_{i,j} \in \mathbb{R} \,|\, i \in \{1, \ldots, m_E\} \wedge j \in \{1, \ldots, n_E\} \}$$

represents the set of all entries of $\boldsymbol{P}$.

Map smoothing

The patch layer can be smoothed by calculating the mean of the cells surrounding cell (i, j) within a radius of $r_s \in \mathbb{R}_{>0}$, including cell (i, j) itself:

$$M_{i,j} = \left\{ m_{k,l} \,|\, m_{k,l} \in S(\boldsymbol{P}) \wedge (i-k)^2 + (j-l)^2 \leq \left(\frac{r_s}{s}\right)^2 \right\}. \tag{5.25}$$

This process is represented by the following equation, which calculates the mean $m_{i,j}$ for cell $p_{i,j}$:

$$m_{i,j} = \frac{1}{|M_{i,j}|} \sum_{m_{k,l} \in M_{i,j}} m_{k,l}. \tag{5.26}$$

We will refer to

$$\underset{m_E \times n_E}{\boldsymbol{M}}$$

that contains $m_{i,j}$ for (i, j)-th entry, as a smoothed layer.

Roughness calculation

The surface roughness for cell (i, j) is calculated as an absolute value of difference between the corresponding patched (5.24) and smoothed (5.26) layer values:

$$r_{i,j} = |p_{i,j} - m_{i,j}|. \tag{5.27}$$

Normals to surface

The method adopted for calculating a normal to the surface within a cell (i, j) involves the eigen decomposition of a correlation matrix of neighboring cells $M_{i,j}$ defined in (5.25). It is assumed that there are at least three neighbors. The cell (i, j) of a patch layer can be represented as a vector

$$\boldsymbol{v_{i,j}} = \begin{pmatrix} i & j & p_{i,j} \end{pmatrix}^T$$

Let $M_{\boldsymbol{v}_{i,j}}$ be a set of vectors and $\boldsymbol{v_{k,l}} \in M_{\boldsymbol{v}_{i,j}}$ be a vector corresponding to the element $m_{k,l} \in M_{i,j}$. The neighboring cells in the form of vectors are used to compute a correlation matrix

$$\boldsymbol{\Sigma_{i,j}} = \frac{1}{|M_{\boldsymbol{v}_{i,j}}|} \sum_{\boldsymbol{v_{k,l}} \in M_{\boldsymbol{v}_{i,j}}} \boldsymbol{v_{k,l}} \boldsymbol{v_{k,l}}^T - \frac{1}{|M_{\boldsymbol{v}_{i,j}}|^2} \sum_{\boldsymbol{v_{k,l}} \in M_{\boldsymbol{v}_{i,j}}} \boldsymbol{v_{k,l}} \left(\sum_{\boldsymbol{v_{k,l}} \in M_{\boldsymbol{v}_{i,j}}} \boldsymbol{v_{k,l}} \right)^T.$$

By performing the eigen decomposition of the correlation matrix using

$$\boldsymbol{\Sigma_{i,j}} = \boldsymbol{V \Lambda V}^T,$$

unitary eigenvectors represented as the columns of matrix $\boldsymbol{V}$ and a diagonal matrix $\boldsymbol{\Lambda}$ of eigenvalues are obtained. The eigenvector corresponding to the lowest eigenvalue is the resulting normal vector

$$\boldsymbol{n_{i,j}} = \begin{pmatrix} n_{x_{i,j}} & n_{y_{i,j}} & n_{z_{i,j}} \end{pmatrix}^T. \tag{5.28}$$

Slope calculation

The local slope of the surface within the cell (i, j) is calculated using the z-coordinate of (5.28) with the following equation:

$$s_{i,j} = \arccos(n_{z_{i,j}}) \tag{5.29}$$

that results in an interval of possible values:

$$s_{i,j} \in \left\langle 0, \frac{\pi}{2} \right\rangle,$$

where 0, $\frac{\pi}{2}$ correspond to a flat/vertical surface, respectively.

Weight function

The traversability is estimated using a normalized weighted sum of $s_{i,j}$ (5.29) and $r_{i,j}$ (5.27):

$$t_{i,j} = \alpha \left(1 - \frac{s_{i,j}}{s_{max}} \right) + \beta \left(1 - \frac{r_{i,j}}{r_{max}} \right), \tag{5.30}$$

where $\alpha, \beta \in \mathbb{R}$ are weights and $s_{max}, r_{max} \in \mathbb{R}_{>0}$ are parameters that correspond to the maximum allowed slope and roughness of the surface,

respectively. We consider $\alpha = \beta = 0.5$ to give a role of slope and roughness in the traversability estimation the same importance. Additionally, $t_{i,j} \leq 0$ is considered as non-traversable, therefore represents an obstacle.

Let

$$\underset{m_E \times n_E}{\boldsymbol{T}}$$

containing $t_{i,j}$ of the (i,j)-th position, be the traversability layer. All undefined $t_{i,j}$ in $\boldsymbol{T}$ corresponding to cells (i,j) within a pre-defined radius around the robot are considered as non-traversable. This results in the local traversability map.

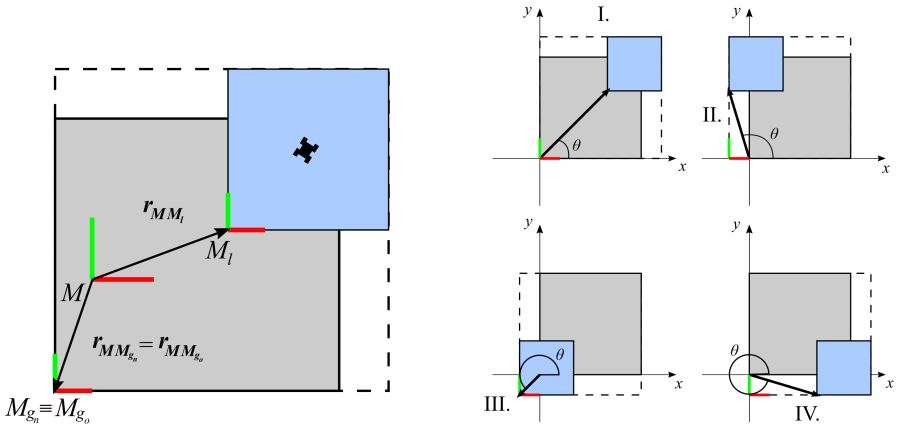
5.6 Global traversability map

The global traversability map serves as the basis for the global planner to plan a route to the desired destination. The method used to create this map involves the continuous stitching, or merging, of updates of the local traversability map M_l.

At the navigation system startup, the global traversability map M_g is initialized with the first available M_l. When a new update of M_l is received, a new global traversability map M_{g_n} is created by merging the old global traversability map M_{g_o} with the current M_l. This results in M_{g_n} with a size of the bounding box that encloses the overlapping M_{g_o} and M_l. This process is illustrated in Figure 5.6a, where the two-dimensional vectors $\boldsymbol{r_{MM_{g_n}}}$, $\boldsymbol{r_{MM_{g_o}}}$, and $\boldsymbol{r_{MM_l}}$ represent the relative translation of M_{g_n}, M_{g_o}, M_l frames with respect to the map frame M of the LIO-SAM. The relative position described with translational vectors is sufficient as the orientation of all mentioned frames is identical.

The translational vector $\boldsymbol{r_{MM_{g_n}}}$ depends on the angle θ between $\boldsymbol{r_{M_{g_o}M_l}}$ and the positive x-axis of the M frame (shown in Figure 5.6b):

$$\boldsymbol{r_{MM_{g_n}}} = \begin{cases} \boldsymbol{r_{MM_{g_o}}}, \text{ if } \theta \in \langle 0, \frac{\pi}{2}) \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} (\boldsymbol{r_{MM_l}} - \boldsymbol{r_{MM_{g_o}}}) + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \boldsymbol{r_{MM_{g_o}}}, \text{ if } \theta \in \langle \frac{\pi}{2}, \pi) \\ \boldsymbol{r_{MM_l}} - \boldsymbol{r_{MM_{g_o}}}, \text{ if } \theta \in \langle \pi, \frac{3}{2}\pi) \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \boldsymbol{r_{MM_{g_o}}} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} (\boldsymbol{r_{MM_l}} - \boldsymbol{r_{MM_{g_o}}}), \text{ if } \theta \in \langle \frac{3}{2}\pi, 2\pi) \end{cases} \tag{5.31}$$



(a) : A new global map M_{g_n} (dashed) as a result of merge of an old global map M_{g_o} (gray) and local map M_l (blue).

(b) : Positions of the new global map M_{g_n} frame (red, green) according to the magnitude of θ.

Figure 5.6: The process of updating the global traversability map (a) and locations of the updated global map's origin, depending on the orientation of the translational vector between the old global map and the local traversability map (b).

The dimensions of M_{g_n} corresponding to θ are:

$$\begin{pmatrix} m_{g_n} & n_{g_n} \end{pmatrix}^T = \begin{cases} \frac{1}{r}\boldsymbol{r_{M_{g_o}M_l}} + \begin{pmatrix} m_l & n_l \end{pmatrix}^T, \text{ if } \theta \in \langle 0, \frac{\pi}{2}) \\ \frac{1}{r}\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}\boldsymbol{r_{M_{g_o}M_l}} + \begin{pmatrix} m_{g_o} & n_l \end{pmatrix}^T, \text{ if } \theta \in \langle \frac{\pi}{2}, \pi) \\ \frac{1}{r}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\boldsymbol{r_{M_{g_o}M_l}} + \begin{pmatrix} m_{g_o} & n_{g_o} \end{pmatrix}^T, \text{ if } \theta \in \langle \pi, \frac{3}{2}\pi) \\ \frac{1}{r}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\boldsymbol{r_{M_{g_o}M_l}} + \begin{pmatrix} m_l & n_{g_o} \end{pmatrix}^T, \text{ if } \theta \in \langle \frac{3}{2}\pi, 2\pi) \end{cases} \tag{5.32}$$

, where r denotes the resolution of the local traversability map, $(m_l,\ n_l)$ are dimensions of M_l, and $(m_{g_o},\ n_{g_o})$ are dimensions of M_{g_o}.

When merging overlapping global and local traversability maps, the global map M_{g_o} is overwritten with values from the local map that have been classified as traversable or as obstacle. Cells classified as unknown are not considered in this process in order to prevent the global map from erasing mapped areas where the local map has incomplete information.

5.7 Kinematics of the robotic platform

The navigation method assumes a four-wheeled skid-steered mobile robotic platform. According to [52], a four-wheeled platform can be described using the two-wheel differential drive kinematics [53].

Authors of [52] assume the robot's body frame with its origin in the center

of an area defined by the ground contact points of the left and right wheels and with y axis aligned with the front of the robot. Velocities (v_l, v_r) represent the linear velocity of left and right wheels, respectively, with respect to the robot's frame. The vector $\boldsymbol{v} = (v_x, v_y)$ represents the robot's translational velocity in its body frame, while ω_z represents the angular velocity around the robot's z axis. The Instantaneous Center of Rotation (ICR) of the moving platform, denoted as $\boldsymbol{ICR_v} = (x_{ICR_v},\ y_{ICR_v})$, is defined in the robot's frame, along with the ICRs for left and right treads: $\boldsymbol{ICR_l} = (x_{ICR_l},\ y_{ICR_l})$ and $\boldsymbol{ICR_r} = (x_{ICR_r},\ y_{ICR_r})$. By analyzing Figure 5.7a, the following relations can be derived:

$$x_{ICR_v} = \frac{-v_y}{\omega_z}, \tag{5.33}$$

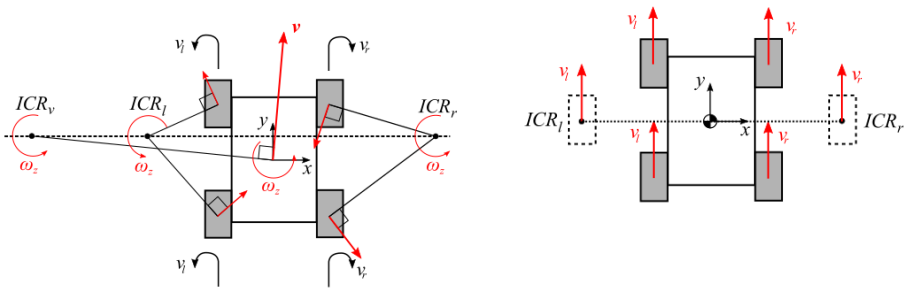
$$x_{ICR_l} = \frac{\alpha_l v_l - v_y}{\omega_z}, \tag{5.34}$$

$$x_{ICR_r} = \frac{\alpha_r v_r - v_y}{\omega_z}, \tag{5.35}$$

$$y_{ICR_v} = y_{ICR_l} = y_{ICR_r} = \frac{v_x}{\omega_z}. \tag{5.36}$$

where α_l, α_r are correction factors that affect (v_l, v_r). We assume an ideal symmetrical kinematic model [52]:

$$\alpha = \alpha_l = \alpha_r, \quad y_{ICR_v} = 0, \quad x_{ICR} = -x_{ICR_l} = x_{ICR_r}. \tag{5.37}$$



(a) : General description of a moving four-wheeled platform.

(b) : A four-wheeled platform described with an ideal differential drive (dashed).

Figure 5.7: Kinematics of a four-wheeled robotic platform in the motion plane. The left figure (inspired by [52]) illustrates a motion of a generic four-wheeled platform, where $\boldsymbol{ICR_v}$ is the Instantaneous Center of Rotation (ICR) of the moving platform, and $\boldsymbol{ICR_l}$, $\boldsymbol{ICR_r}$ are ICRs corresponding to left and right wheels, respectively. The $\boldsymbol{v}$ represents translational vector of a moving robot, ω_z the angular velocity, and (v_l, v_r) linear velocities of left and right wheels, respectively, all of the above with respect to the robot's body frame. The right figure illustrates the conversion of a four-wheeled platform to an ideal differential drive.

Under these assumptions, the four-wheeled platform can be described using an ideal differential drive model, as shown in Figure 5.7b. In this model, the

ground contact points of the ideal differential drive coincide with the ICRs of the four-wheeled platform and the center of mass is coincident with the robot's body frame. Equations (5.34), (5.35) and assumptions (5.37) can be used to calculate the velocities (v_l, v_r) yielding the following equations:

$$v_l = \frac{v_y - x_{ICR} \cdot \omega_z}{\alpha}, \quad v_r = \frac{v_y + x_{ICR} \cdot \omega_z}{\alpha}. \tag{5.38}$$

The parameters x_{ICR} and α can be obtained through a simple experimental procedure described in [54]. In this procedure, the authors set $v_r = -v_l$ to achieve in-place rotation of the robot around its z axis. The value of x_{ICR} is then calculated using

$$x_{ICR} \approx \frac{\int v_r \mathrm{d}t - \int v_l \mathrm{d}t}{2\phi}, \tag{5.39}$$

where ϕ represents the actual angle of rotation. The value of α is obtained from a straight motion of the robot using

$$\alpha \approx \frac{2d}{\int v_r \mathrm{d}t + \int v_l \mathrm{d}t}, \tag{5.40}$$

where d represents the actual distance travelled by the robot. In this thesis, we assume that $\alpha = 1$, and x_{ICR} to be provided by the manufacturer of the used robotic platform, introduced in section 7.2.1. Velocities v_l and v_r are then used to set instantaneous angular velocities of each wheel pair using ω_z and v_y from the local planner.

5.8 Planners

The navigation system employs a hybrid approach that combines both global and local planning strategies. The global planner retrieves an updated global traversability map in order to determine the shortest path from the robot's current pose to a target goal. The robot is represented as a two-dimensional point that coincides with the top-down projection of the origin of the robot's body frame. To prevent collisions, obstacles are inflated by a half of a diagonal of the robot's footprint. In addition to providing a path to an unmapped location, the system also allows for path planning in areas that are initially unknown. As a result, it is necessary to incorporate a local planner that can adjust the planned trajectory based on newly-mapped obstacles in order to ensure collision avoidance.

5.8.1 Global planner

The global planning employs the Dijkstra's algorithm [55] with the worst-case performance $\Theta(|E| + |V| \log |V|)$ (using a Fibonacci heap min-priority queue), where $|E|$ is the number of edges and $|V|$ is the number of vertices, in our case, cells . This algorithm is preferred over greedy algorithms, for example A*, due to its ability to provide a guaranteed shortest path at the cost of increased computational complexity in certain situations. The pseudo-code of the Dijkstra's algorithm is shown in Algorithm 1.

Algorithm 1 Dijkstra's algorithm

Require: map, initialPose, targetPose

for each cell c in map: // Initialization
  if c is not obstacle:
    distance of $c \leftarrow \infty$
    parent of $c \leftarrow$ UNDEFINED
    add c to *Unvisited* set
distance of initialPose $\leftarrow 0$

while *Unvisited* is not empty **do** // Main loop
  $d \leftarrow$ cell in *Unvisited* with minimum distance
  if $d =$ targetPose
    $S \leftarrow$ retrievePath()
    return S
  remove d from *Unvisited*
  for each neighbor n of d still in *Unvisited*:
    $tmp \leftarrow$ distance of d + edge(d, n)
    if $tmp <$ distance of n:
      distance of $n \leftarrow tmp$
      parent of $n \leftarrow d$
end while

fn retrievePath():
  $P \leftarrow$ empty path
  $d \leftarrow$ targetPose
  if parent of d is DEFINED **or** $d =$ targetPose:
    while d is DEFINED:
      insert d at the beginning of P
      $d \leftarrow$ parent of d

5.8.2 Local planner

The local planning employed in the navigation scheme utilizes the Dynamic Window Approach (DWA) [56], which is derived from the motion dynamics of a robot. The planner accesses an up-to-date local traversability map to provide information on surrounding obstacles, and a global path, which it follows. If the global path collides with obstacles in the local traversability map, the local planner notifies the global planner to re-plan the route.

The planner is restricted to circular trajectories, which are defined by the translational and rotational velocities of the robot, denoted as v and ω, respectively. Velocity control commands for the robot are searched directly within a two-dimensional space of v and ω. This search space is further reduced to include only admissible velocities V_a that ensure safe trajectories, meaning that the robot, moving with the given velocities, is able to stop before reaching the nearest obstacle on the corresponding circular trajectory. A dynamic window is then applied as a final restriction to the V_a, resulting

in velocities V_r that the robot is able to reach within a short time interval based on the robot's maximum acceleration.

A number of N velocity pairs (v_i, ω_i) are sampled from the V_r, and evaluated using an objective function [56]

$$G(v_i, \omega_i) = \sigma(\alpha \cdot heading(v_i, \omega_i) + \beta \cdot dist(v_i, \omega_i) + \gamma \cdot vel(v_i, \omega_i)),$$

yielding a cost for each curvature. The velocity pair with the highest cost is then used to control the robot.

The objective function consists of a weighted sum of the results of three functions. The *heading* function represents the difference between the heading of the target goal and the heading of the predicted robot pose. The *dist* function represents the distance to the nearest obstacle intersecting the circular trajectory. The *velocity* function evaluates the progress of the robot that follows the given trajectory. The weighted sum is smoothed with σ function. A pseudo-code of the DWA planner is provided in Algorithm 2.

Algorithm 2 Dynamic Window Approach

Require: robotPose, targetPose, v_{curr}, ω_{curr}, $\dot{v}_{max}$, $\dot{\omega}_{max}$, map

$v_f \leftarrow$ forwardVelocity(robotPose, $\dot{v}_{max}$, $\dot{\omega}_{max}$)
$V_r \leftarrow$ DynamicWindow(v_{curr}, ω_{curr}, $\dot{v}_{max}$, $\dot{\omega}_{max}$)

for each (v_i, ω_i) in V_r:
    $d_o \leftarrow$ distanceToObstacle(v_i, w_i, $\dot{v}_{max}$, $\dot{\omega}_{max}$, map)
    $d_b \leftarrow$ RobotBreakingDistance(v_i)
    if $d_o > d_b$: // Robot stops before reaching an obstacle
        $heading \leftarrow$ headingDifference(robotPose, targetPose, v_i, w_i)
        $dist \leftarrow$ (d_o - d_b) / (trajectoryDistance - d_b) // Normalize
        $vel \leftarrow |v_f - v_i|$
        $cost \leftarrow$ objectiveFunction($heading$, $dist$, vel, α, β, γ)
        if $cost > cost_{max}$
            $v_{best} \leftarrow v_i$
            $\omega_{best} \leftarrow \omega_i$
            $cost_{max} \leftarrow cost$

setRobotVelocity(v_{best}, ω_{best})

Chapter 6

Implementation

This chapter presents a detailed account of the implementation of the navigation method in the form of a ROS project, organized into packages. The initial section outlines the utilized software. The subsequent section provides a description of the used robotic platform and the last section delves into the implementation of each component of the navigation scheme. Apart from the software mentioned in section 6.1, the navigation system utilizes existing ROS packages that implement certain parts of the navigation system and are referenced in the following sections. The relevant parts of the software are attached to the thesis and are listed in Appendix D.

6.1 Used software

6.1.1 Robot Operating System (ROS)

ROS (Robot Operating System) [57] is a flexible framework for building and managing robotic systems, acting more as a hardware abstraction layer than an operating system. It provides a set of tools and libraries, referred to as packages, for developing robot software, including sensor drivers, communication, visualization, and debugging. ROS employs a publish-subscribe communication model, in which nodes (executable programs) can publish messages to a specific topic or subscribe to a topic. Nodes can also send and receive service requests and responses. This allows nodes to communicate with one another and exchange data, enabling the creation of complex robotic behaviors and control systems. Nodes are started by launching a ROS-specific XML .launch file. We will refer to this type of file as launch file. In addition, ROS utilizes a multi-variate dictionary known as the Parameter server, which allows for the runtime storage and retrieval of node parameters.

This thesis utilizes the ROS Noetic distribution, primarily targeted at the Ubuntu 20.04 release.

6.1.2 ROS Visualization (RViz)

Rviz [58] is a 3D visualization tool developed within the ROS framework. It enables users to visualize data from various sensors, including laser scanners,

IMUs, and cameras, as well as robot models, reference frames, built maps, and other elements of the robotic system. Rviz also provides functionality for interacting with and debugging the system, such as the ability to view and edit parameter settings and display diagnostic information.

6.1.3 Gazebo simulator

Gazebo [59] is a widely used 3D robotics simulator for the development and testing of robotic systems, particularly mobile robots. It allows users to design and simulate robots in a virtual environment, including their kinematics, dynamics, and sensor behavior. Gazebo includes a physics engine that can accurately model the interaction of objects in the simulated environment, including collisions, friction, and other physical phenomena. The simulator natively incorporates the Open Dynamics Engine (ODE)[1], but it also supports DART[2], Bullet[3], and Simbody [60] engines. The 3D graphics are rendered with the Ogre[4] (version 2.1) graphics rendering engine.

Gazebo also includes support for a variety of noise models and sensors, such as cameras, laser scanners, and IMUs. In most cases, Gaussian noise is used for the noise models, but custom noise properties can also be specified. Gazebo communicates with ROS using the `gazebo_ros_pkgs`[5] ROS built-in meta-package, which provides plugins that offer message and service publishers for interfacing.

For object and scene description, Gazebo uses the Simulation Description Format (SDF)[6] exclusively, while the robot description also supports the Unified Robot Description Format (URDF)[7].

This thesis utilizes Gazebo version 11.0.0.

6.2 Robotic platform

The robot Husky[61] from Clearpath Robotics Inc. serves as the testing platform for the navigation system. It is a medium-sized, four-wheeled development robotic platform with large lug-tread tires, making it suitable for use in rough terrain environments. Detailed specifications can be found in section 7.2.1.

The ROS packages forming the `husky` meta-package [62] provide a robot URDF description, `husky_description.urdf.xacro` (`husky_description` package), velocity control (`husky_control` package), and simulation tools (`husky_gazebo` package). The robot is controlled using the `diff_drive_controller` [63], which subscribes to topic `/cmd_vel` of the message type `geometry_msgs/Twist`, which provides twist velocities $(\boldsymbol{v}, \boldsymbol{\omega})$. The

[1] `https://www.ode.org/`
[2] `https://dartsim.github.io/`
[3] `https://pybullet.org/wordpress/`
[4] `https://www.ogre3d.org/`
[5] `https://github.com/ros-simulation/gazebo_ros_pkgs`
[6] `http://sdformat.org/`
[7] `https://github.com/ros/urdf`

velocity controller is started from `control.launch` file, which is included in `spawn_husky.launch` file that spawn the robot in Gazebo using `gazebo_ros` package [64]. The robot's body frame is referred to as `base_link`. Apart form the `base_link` frame, other unimportant static frames are located on the robot, to which sensors are mounted.

6.3 Sensors

This section provides only the implementation of the used sensor types. Specifications of each sensor can be found in section 7.2.2. Figure 6.1 depicts the ROS sensor topics.

Laser scanner

In this thesis, lidar is an essential sensor for continuous localization and mapping part of the navigation system. The simulated lidar model is implemented in the ROS meta-package `velodyne_simulator` [65] using the generic `gazebo_ros_laser_controller` sensor plugin, which employs raytracing to determine the range for each sampled point. The lidar URDF, `VLP-16.urdf.xacro`, is included in the Husky URDF description. The lidar scans are published to the `/husky_sensors/velodyne_points` topic of the message type `sensor_msgs/PointCloud2` with respect to the `velodyne` frame.

To incorporate the lidar scans into the navigation system, the scans are transformed to the `base_link` frame using the `pcl_transformer` node located in the `rough_terrain_navigation/husky_transforms` package. This node utilizes the PCL library [66] and the `pcl_ros` package [67] to transform the scans. Firstly, the *pcl*::*fromROSMsg()* function converts the ROS message to a PCL point cloud. Then, *pcl_ros*::*transformPointCloud()* is applied to transform the point cloud using *tf*::*TransformListener()* to provide a static tranformation between `velodyne` and `base_link` frames. Finally, the transformed point cloud is converted back to the ROS message using *pcl*::*toROSMsg()* and published. This node is started from the `pcl_transformer.launch` file.

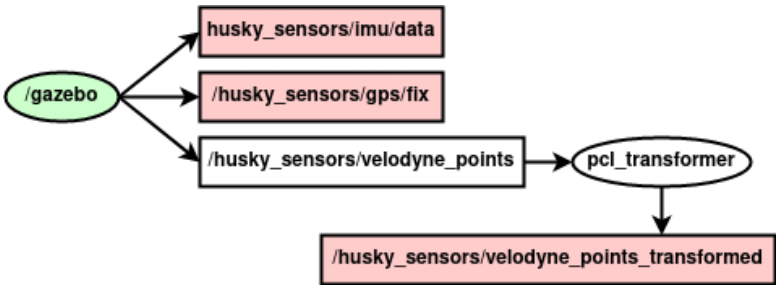


Figure 6.1: The sensor part of the navigation system. ROS nodes (ellipses) and topics (boxes) where inputs are marked in green and outputs in red color.

IMU

The IMU, which is also an essential sensor for the localization part of the navigation system, is modeled using the `hector_gazebo_plugins` ROS package [68], specifically the `GazeboRosImu` plugin. The sensor implementation is the part of the Husky URDF. The IMU data is published to the `/husky_sensors/imu/data` topic of the message type `sensor_msgs/Imu`. Although the specified IMU reference frame ,`imu_link`, is located in the IMU mounting on the robot, the Gazebo simulator implicitly samples IMU data from the `base_link` with respect to this link, therefore no transformation is needed.

GPS

The GPS receiver plays a supportive role in the localization aspect of the navigation system. It is modeled using the `hector_gazebo_plugins`, specifically the `GazeboRosGps` plugin. The sensor implementation is the part of the Husky URDF. The GPS data is published to the `/husky_sensors/gps/fix` topic of the message type `sensor_msgs/NavSatFix` and sampled from the `navsat_link` frame. The data is transformed to the `base_link` frame later in the 6.4.1 section.

6.4 Method implementation in ROS

6.4.1 LIO-SAM

The LIO-SAM method, described in section 5.3, is implemented in the `lio_sam` package [10], [18], which consists of four nodes.

The `/lio_sam_imuPreintegration` node subscribes to the `/husky_sensors/imu/data` topic to receive IMU measurements, as well as the `/lio_sam/mapping/odometry` and `/lio_sam/mapping/odometry_incremental` topics to receive lidar odometry messages of the `navsat_msgs/Odometry` type. This node is responsible for estimating IMU bias, performing graph optimization, and publishing odometry of the message `navsat_msgs/Odometry` type to the `/odometry/imu` topic.

The `/lio_sam_imageProjection` node subscribes to the `/husky_sensors/imu/data`, `/husky_sensors/velodyne_points_transformed`, and `/odometry/imu_incremental` topics to deskew incoming lidar point clouds, and publishes the resulting data to the `/lio_sam/deskew/cloud_info` topic of the `lio_sam/cloud_info` message type. Additionally, this node provides an initial guess for the feature scan-matching process in lidar odometry. The `/lio_sam_FeatureExtraction` node subscribes to the deskewed clouds and performs the extraction of edge and planar features, which are published to the `/lio_sam/feature/clouf_info` topic of the `lio_sam/cloud_info` message type .

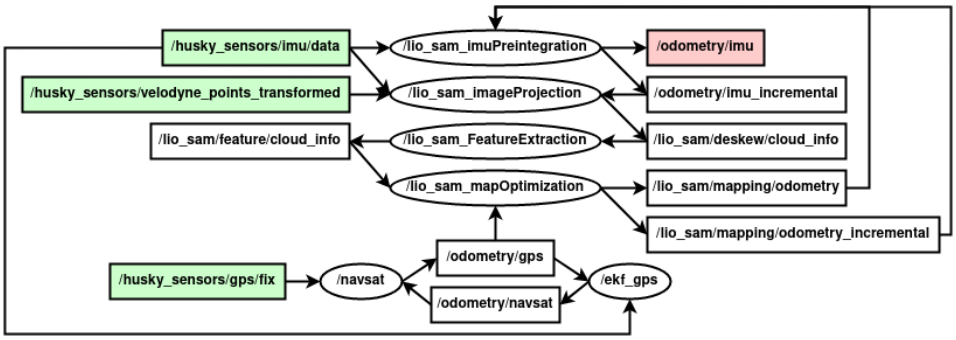


Figure 6.2: The odometry part of the navigation system. ROS nodes (ellipses) and topics (boxes) where inputs are marked in green and outputs in red color.

Finally, the `/lio_sam_mapOptimization` node subscribes to the extracted features and to the `/odometry/gps` topic of the `navsat_mags/Odometry` message type, which represents GPS measurements transformed into world Cartesian coordinates. This node performs feature matching, loop closure, and main factor graph optimization using the GTSAM library [69] and publishes the resulting lidar odometry messages to the `/lio_sam/mapping/odometry` and `/lio_sam/mapping/odometry_incremental` topics.

The aforementioned `/odometry/imu_incremental` and `/lio_sam/mapping/odometry_incremental` topics contain unoptimized odometry estimates that are used to calculate IMU bias, as jumps in the optimized odometry may disrupt this process.

The messages published to the `/odometry/gps` topic are provided by the `/navsat` node of the `navsat_transform_node` type from the `robot_localization` ROS package [70], which subscribes to GPS measurements in the WGS84 LLA format and transforms them into world Cartesian coordinates using Equirectangular projection. The `ekf_gps` node of the `ekf_estimation node` type from the `robot_localization` package is also integrated into the system and is used in cases where the first GPS measurement is obtained after the robot has moved from its initial position.

All of the aforementioned nodes are launched from the launch file structure with the root `lio_sam_nav.launch` file. The implementation diagram of the LIO-SAM method is shown in Figure 6.2.

6.4.2 Probabilistic terrain mapping

The probabilistic terrain mapping method, described in section 5.4, is provided by the `/elevation_mapping` node and implemented in the `elevation_mapping` ROS package [49].

This node subscribes to the topic `/odometry/imu_pose_with_cov`, which provides `geometry_msgs/PoseWithCovarianceStamped` messages that are converted from the LIO-SAM odometry `navsat_msgs/Odometry` messages using the `/odom_msg_converter` node. In addition, the `/elevation_mapping` node receives lidar measurements in order to fuse them into a 2.5D elevation

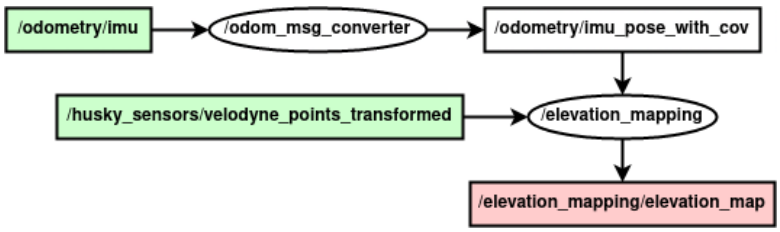


Figure 6.3: The mapping part of the navigation system. ROS nodes (ellipses) and topics (boxes) where inputs are marked in green and outputs in red color.

map.

The map generation is facilitated by the `grid_map` ROS package [71], which provides an implementation of a layered map in the *GridMap()* class along with supportive functions for manipulating data in this class.

The resulting elevation map provides an `elevation` layer, which contains height values assigned to each cell, and is published to the topic `/elevation_mapping/elevation_map` of the `grid_map_msgs/GridMap` message type. The aforementioned nodes are launched from the `elevation_mapping.launch` and `odom_msg_converter.launch` files, respectively. The implementation diagram is shown in Figure 6.3.

6.4.3 Traversability estimation

The traversability estimation described in section 5.5 is implemented in the `filters_demo` node, located in the `grid_map/grid_map_demos` package [71] and renamed to `traversability_estimation` in the navigation system. It is started from the `husky_traversability.launch` file. The node subscribes to the `/elevation_mapping/elevation_map` topic of the message type `grid_map_msgs/GridMap`, which provides a local map with an elevation layer. The traversability estimation is performed using chained filters, provided mainly by the `grid_map/grid_map_filters` package [71]. These filters represent classes that implement certain parts of the traversability estimation and are fetched by the `traversability_estimation` node. The filters are specified in a separate `filter_chain_husky.yaml` configuration file located in `rough_terrain_navigation/config/`. Each filter retrieves certain layers of the local map and modifies them or adds completely new layers with calculated data.

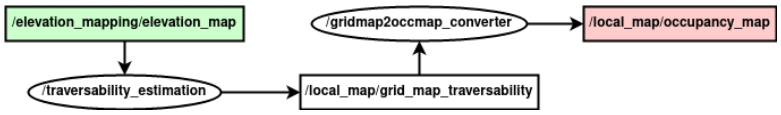


Figure 6.4: The diagram of the traversability estimation part of the navigation system. ROS nodes (ellipses) and topics (boxes) where inputs are marked in green and outputs in red color.

The map patching, described in section 5.5, is implemented in *Medianfilter.cpp*, which adds the `elevation_medianFill` layer to the local map.

This layer is retrieved by *MeanInRadiusFilter.cpp*, implementing section 5.5, which adds the `elevation_smooth` layer. Then, *MathExpressionfilter.cpp* accesses the `elevation_medianFill` and `elevation_smooth` layers to calculate roughness values in the form of the `roughness` layer, according to the section 5.5.

Normals to the surface, described in the section 5.5, are calculated from the `elevation` layer using *NormalVectorsFilter.cpp*, resulting in three additional layers with the prefix `normal_vectors_` for each coordinate of a normal vector. The filter uses the *EigenSolver()* class from the Eigen library to compute the eigenvectors. The `normal_vectors_z` layer is accessed by another *MathExpressionFilter.cpp* to calculate the slope using arccos() function, resulting in the `slope` layer.

The `roughness` and the `slope` layer are retrieved by a third *MathExpression.cpp*, which implements the weight function, according to the section 5.5, that calculates the traversability. The added `traversability` layer is then accessed by *ImaginaryObstaclesInRadiusFilter.cpp*, which changes map cells with NaN values within a specified radius around the robot to 0, resulting in the `traversability_with_imag_obstacles` layer. Lastly, lower and upper thresholds, implemented in *ThresholdFilter.cpp*, are applied to this layer, providing final traversability values in interval $\langle 0.0, 1.0 \rangle$, where 0.0 signifies a non-traversable cell and 1.0 signifies a completely traversable cell.

The thresholded `traversability_with_imag_obstacles` layer is then published to the `/local_map/grid_map_traversability` topic, to which the `/gridmap2occmap_converter` node subscribes and converts the map to the `nav_msgs/OccupancyGrid` message type using the *GridMapRosConverter*::*toOccupancyGrid()* function. This function maps the traversability values to the interval of integer values $\langle 0, 100 \rangle$, where 0 is completely traversable and 100 is an obstacle. The NaN values are mapped to -1. The new map is published to the `/local_map/occupancy_map` topic.

6.4.4 Global traversability map

The theory behind creating the global traversability map, described in section 5.6, is implemented in the `map_stitcher` node, located in the `rough_terrain_navigation/map_stitcher` package. The node is started from the `map_stitcher.launch` file. The node subscribes to `/local_map/occupancy_map` topic, which provides local traversability maps. Two consecutive local maps are combined using the *combineGrids()* function from the `occupancy_grid_utils` package [72], resulting in a global traversability map of the same message type, which is then published to the topic `global_map_stitched/occupancy_map`.

In addition, a ROS server is implemented, providing the *goalInMap* service, which is used when a new goal for the path planning is set from Mapviz [73]. The client implemented in the `move_base` plugin in Mapviz sends a request if the new goal is located within the currect global map. If not, the global map is resized accordingly and a response is sent to MapViz, which can then send the new goal to `move_base` node for path planning.

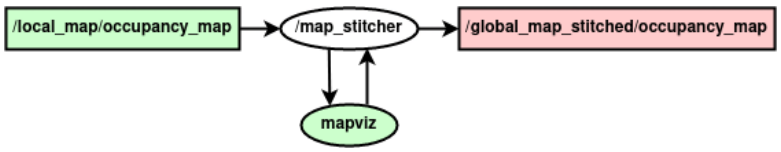


Figure 6.5: The diagram of the global map part of the navigation system. ROS nodes (ellipses) and topics (boxes) where inputs are marked in green and outputs in red color.

Without this service, the new goal would be sent directly to the `move_base` node without taking into account the current size of the global map, which could result in the goal being located outside of the map, and therefore invalid for the global planner.

A ROS diagram of the implemented global map creation is shown in Figure 6.5.

6.4.5 Path planning

The combination of the global and local planner based on Dijkstra's algorithm and DWA (see sections 5.8.1 and 5.8.2), respectively, utilized by the navigation system, is implemented in the `move_base` node from the `move_base` ROS package [74]. The node is launched from the `move_base.launch` file.

This node integrates the *nav_core::BaseGlobalPlanner* and *nav_core::BaseLocalPlanner* interfaces, provided by the `nav_core` ROS package [75] to load the *navfn::NavFn()* class from the `navfn` ROS package [76] as the global planner, and the *dwa_local_planner::DWAPlanner()* class from the `dwa_local_planner` ROS package [77] as the local planner.

In addition, the `move_base` node subscribes to the `/local_map/occupancy_map` and `/global_map_stitched/occupancy_map` topics to obtain global and local traversability maps, which are converted to costmaps of class *costmap_2d::Costmap2DROS()* from the `costmap_2d` ROS package [78]. These costmaps differentiate between costs: *costmap_2d::LETHAL_OBSTACLE*, *costmap_2d::NO_INFORMATION*, and *costmap_2d::FREE_SPACE*, corresponding to values of 100, -1, and $\langle 0, 99 \rangle$, respectively, in the occupancy maps.

The costmaps utilize two layers: the *costmap_2d::StaticLayer*, which represents the converted occupancy map, and the *costmap_2d::InflationLayer*, which represents extracted obstacles inflated arounda certain radius to prevent collisions. Both planners utilize the inflation layers for path planning.

In addition to the occupancy map topics, the `move_base` node subscribes to the `/odometry/imu` topic to obtain robot pose estimates, which are used by the local planner to estimate the robot's velocity. The node also subscribes to the `move_base/goal` topic of the `move_base_msgs/MoveBaseActionGoal` message type to receive a designated goal in world Cartesian coordinates and set from MapViz. In most cases, the goal cannot be reached exactly, therefore the `dwa_local_planner` has tolerance parameters `yaw_goal_tolerance` and `xy_goal_tolerance`. The `move\_base` node, specifically the local planner,

publishes motion commands to the `/cmd_vel` topic to steer the robot.

In the event that a path to the goal cannot be found, the robot executes a recovery behavior consisting of two consecutive full in-place turns to remap the terrain and potentially uncover a route to the goal. A ROS diagram of the integrated `move_base` node is shown in Figure 6.6.

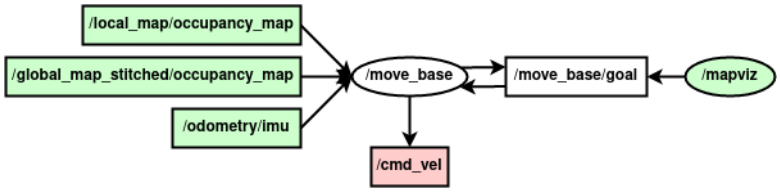


Figure 6.6: The path planning part of the navigation system. ROS nodes (ellipses) and topics (boxes) where inputs are marked in green and outputs in red color.

6.4.6 Setting goals using offline Google maps

In order to set a goal for the navigation system, specifically the global planner, the selected approach specifies the goal in WGS84 LLA coordinates, which can be picked from the exact location on a map of the environment and then transformed into a global Cartesian frame using equirectangular projection. To facilitate this process, the navigation system utilizes MapViz, a ROS tool for data and map visualization.

While several maps are included in MapViz by default, we opted to integrate offline Google Maps, as they provide more detailed and up-to-date information about the environment. The Google Maps satellite view is proxied using MapProxy to provide a Web Map Tile Service (WMTS) using the work of Daniel Snider [79]. Once this process is completed, the map stays cached and can be used offline in MapViz.

The goal can then be specified using a modified version of the `move_base` plugin, which includes an additional implementation of a ROS client that communicates with the `map_stitcher` node. The goal is set through the `move_base_msgs/MoveBaseActionGoal` message published to the `move_base/goal` topic. In addition to goal setting and visualization, the local traversability map and the robot's trajectory are also visualized to provide a more comprehensive view of the running navigation system. The `mapviz` node is launched from the `mapviz.launch` file.

6.5 Launch of the navigation system with simulator

ROS nodes are typically launched from launch files. These launch files may be nested, which simplifies the process of starting nodes significantly. However, launching multiple dependent nodes from a single launch file can cause their improper initialization. This problem can be solved with changes to node structure, but this reduces the versatility of the nodes. Furthermore, monitoring multiple nodes from a single terminal window can become complicated,

especially when node info messages are displayed frequently. Therefore, we use multiple launch files (depicted in Figure 6.7) launched in separate terminal windows.

To avoid the tedious process of setting up these windows every time a new simulation is executed, we use a Terminator [80] layout named `rough_terrain_nav` to source the ROS project and launch all launch files consecutively with a single command:

```
$ terminator -l rough_terrain_nav.
```

This approach works well for small simulated scenes. Large or complex scenes launched withing `bringup.launch` file can take several minutes to load, which can cause depending nodes to be improperly initialized. Therefore launch files containing complex scenes must be launched separately, rather than from `bringup.launch`.

Additionally, we use a simple bash script called `env_setup.sh`, sourced from the ROS project's `setup.bash`, to set specific parameters throughout all launch files using environment variables. This allows for quick setup of the navigation system. The optional Python script `teleop_twist_keyboard.py`, included in the layout, provides manual steering for the robot Husky via `\cmd_vel` topic, and is used when a robot needs to be repositioned in the scene between experiments.

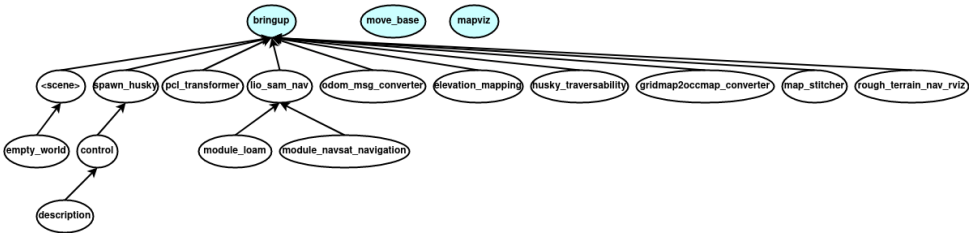


Figure 6.7: The launch file structure of the ROS project (.launch extesions are omitted). The blue-colored launch files are launched in separate terminal windows.

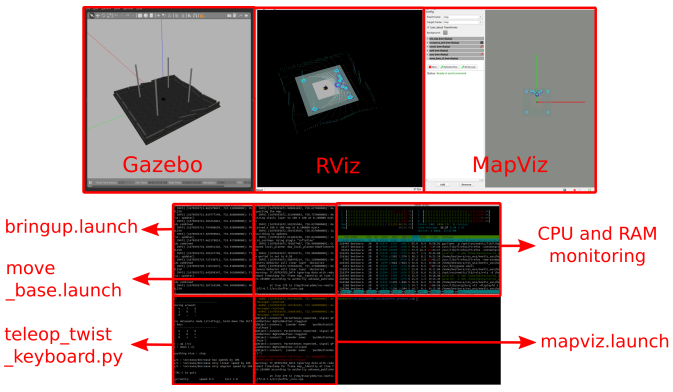


Figure 6.8: Overview of the windows that appear after executing the rough_terrain_nav layout.

Chapter 7

Experiments

In this chapter, the simulated robotic platform and its sensor suite are described (see section 7.2), including the discussion of positioning of the sensors, with a focus on the lidar. Subsequently, configuration of the navigation system and simulator are presented in sections 7.3 and 7.4.1, respectively. Prior to the experiments conducted, the process of creating simulation scenes (described in section 7.4) is outlined in detail. The experiments conducted, their results, and evaluations are presented in section 7.5. Finally, limitations of the proposed navigation method are addressed at the end of this chapter.

7.1 Hardware setup

All experiments were conducted on a laptop PC (Thinkpad T490) equipped with an Intel Core i5-8265U (8 x 1.60 GHz), Intel UHD Graphics 620, and 40 GB of RAM. The laptop was running the Ubuntu 20.04.5 LTS operating system.

7.2 Simulated experimental platform

In this section, we describe the simulated robotic platform and the specific sensors used in the navigation method.

7.2.1 Robot Husky

The robot Husky, introduced in section 6.2, is described in the `husky_description.urdf.xacro` URDF file with dimensions shown in 7.1. However, the base collision boxes in the provided model did not accurately represent the robot's shape and therefore were modified to a more accurate representation shown in Figure 7.2. The specifications of the robot from [61] that are important for the experiments, excluding dimensions, are listed in Table 7.1. For the purposes of simulation, the friction between a terrain and the robot's tires were set to 1.0, which simulates a rubber tire on dry concrete. The value of x_{ICR}, introduced in section 5.7

is derived from the `wheel_separation_multiplier` parameter, provided in `husky/husky_control` package and was determined as $x_{ICR} \approx 0.52$ m.

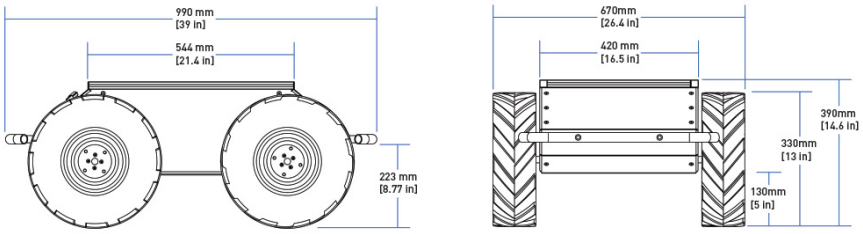


Figure 7.1: Dimensions of the robot Husky in the side and front view, respectively. Obtained from [61].

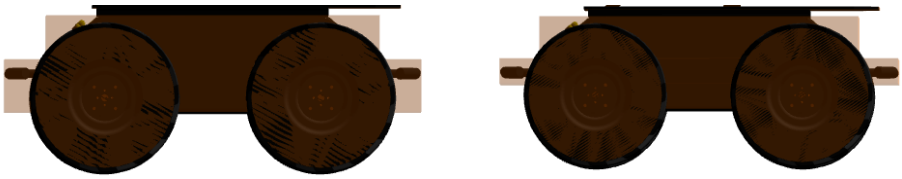


Figure 7.2: A model of Husky robot shown in the simulator with overlaid collision boxes (orange). Modified collision boxes of the front and rear bumpers are displayed on the right.

Selected specifications of the robot Husky	
Weight (kg)	50
Maximum speed (m· s^{-1})	1.0
Ground clearance (m)	0.13
Maximum climb grade (°)	45
Maximum travesal grade (°)	30
Track (m)	0.555

Table 7.1: Selected specifications of the robot Husky.

7.2.2 Sensor suite

All simulated sensors were designed to match the physical sensor equipment, compatible with the sensor mountings on the physical robot, and typically delivered with the robot Husky.

The Velodyne Puck (VLP-16)

Velodyne Lidar Inc. has become a significant contributor to the field of LiDAR technologies, offering high-performance product line. The Velodyne Puck lidar[81] is a suitable choice for use in the following experiments due to its specifications. The important specifications of this sensor from [81] are summarized in Table 7.2.

The selected specifications of the VLP-16 lidar	
Weight (kg)	0.59
Number of channels	16
Measurement range (m)	$\langle 0.9, 100 \rangle$
Range accuracy (m)	Up to $\pm$ 0.03
Horizontal field of view (°)	360
Horizontal angular resolution (°)	$\langle 0.1, 0.4 \rangle$
Vertical field of view (°)	$\langle -15.0, 15.0 \rangle$
Vertical angular resolution (°)	2.0
Rotation rate (Hz)	$\langle 5, 20 \rangle$

Table 7.2: Selected specifications of the Velodyne Puck lidar sensor obtained from [81].

For performance reasons, the simulation parameters for the sensor were limited to a rotation rate of 10 Hz and a horizontal angular resolution of 1.8°. Assuming that the range measurements are Gaussian distributed and the given range accuracy is guaranteed, the sensor accuracy actually represents a $\pm 3\sigma$ interval. Therefore, the measurement noise was modeled as Gaussian additive noise with a mean $\mu = 0$ m and a standard deviation $\sigma = 0.01$ m. The additive noise is internally sampled per each ray.

The CHR-UM7 Orientation Sensor

The IMU is modeled based on the accessible CHR-UM7 IMU produced by CH Robotics company. The specifications of the CHR-UM7 from[82] are listed in Table 7.3.

Specifications of the CHR-UM7 IMU	
Sampling rate (Hz)	500
Static pitch/roll accuracy (°)	± 1
Dynamic pitch/roll accuracy (°)	± 3
Static yaw accuracy (°)	± 3
Dynamic yaw accuracy (°)	± 5
Angle repeatability (°)	0.5
Angular resolution (°)	0.01

Table 7.3: Specifications of the CHR-UM7 orientation sensor obtained from [82].

Since the available sensor specifications[82] differentiate between static and dynamic roll/pitch/yaw accuracy and the robot remains static only during extensive planning, specifications for dynamic behaviour are preferred. However, the plugin only specifies noise for the yaw orientation. Following the same procedure as in the LiDAR modeling, the yaw Gaussian noise was set with a mean of $\mu = 0$ rad and a standard deviation of $\sigma = 0.029$ rad. The remaining noise parameters in the plugin specify linear acceleration and angular rate noise, and their derivation would require a detailed analysis of

the physical sensor. Therefore, example values were used instead (see Table 7.4). The simulated sampling rate was set to 500 Hz.

Simulation parameters of the CHR-UM7			
Sampling rate (Hz)	500		
Linear (x, y, z) acceleration drift ($\mathrm{m{\cdot}s^{-2}}$)	0.005	0.005	0.005
Linear (x, y, z) acceleration Gaussian noise σ ($\mathrm{m{\cdot}s^{-2}}$)	0.005	0.005	0.005
Angular (x, y, z) rate drift ($\mathrm{rad{\cdot}s^{-1}}$)	0.005	0.005	0.005
Angular (x, y, z) rate Gaussian noise σ ($\mathrm{rad{\cdot}s^{-1}}$)	0.005	0.005	0.005
Heading drift (rad)	0.005		
Heading Gaussian noise σ (rad)	0.029		
Yaw offset (rad)	0.5π		

Table 7.4: The list of the parameters used for the simulation of the CHR-UM7 IMU.

The Duro GNSS Receiver

The integration of a GPS receiver helps to reduce larger drift errors, leading to improved pose estimation. However, the navigation method design opted for setting target goals in WGS84 coordinates, making the use of this type of sensor necessary. The Duro GNSS Receiver from Swift Navigation Inc. is a high-performance military-grade receiver that is known for its ability to withstand strong vibrations and provide precise positioning, making it suitable for use in rough terrain. Selected parameters of the Duro GNSS Receiver, obtained from[83], are listed in Table 7.5.

Specifications of the Duro GNSS Receiver	
Weight (kg)	0.8
Horizontal position accuracy (CEP 50) (m)	0.75
Time accuracy (RMS) (ns)	60

Table 7.5: Selected specifications of the Duro GNSS Receiver obtained from [83].

The provided horizontal accuracy is in terms of CEP 50, which is a measure that indicates the radius of a circle centered on the mean of all measurements, where 50 % of the measurements fall within the perimeter of the circle. Since the package models both horizontal and vertical accuracy as additive Gaussian noise, conversion between CEP 50 and standard deviations σ_x, σ_y is necessary. The standard deviations can be isolated from a quantile function of the Rayleigh distribution that describes the general CEP (for derivation, see Appendix B). Substitution of the known CEP 50 horizontal accuracy gives:

$$\sigma_x = \sigma_y = \frac{0.75}{\sqrt{-2\ln(0.5)}}\ \mathrm{m} = 0.637\ \mathrm{m}.$$

The vertical accuracy can be roughly estimated from the given standard deviation of time accuracy, $\sigma_t = 60$ ns. Given that GPS satellites use atomic

clocks to measure time, ground truth of timestamps included in transmitted messages is assumed. The time difference between the timestamp and the current time measured by the receiver is then used for trilateration to estimate the pose of the receiver. Therefore, the σ_t multiplied by the speed of light yields a standard deviation for the vertical accuracy of $\sigma_z = 18$ m. However, this is a very rough estimate and the actual σ_z would be much smaller. The simulated frequency of received GPS messages was set to 40 Hz.

7.2.3 Sensor placement

The placement of sensors, particularly lidar, may improve the navigation system significantly. The position of the lidar sensor affects the density of the point cloud in front of the robot. The goal is to maximize the data density in order to create maps covering the entire area in the robot's frontal proximity.

Three different lidar configurations were tested. In the first configuration (shown in Figure 7.3a), the sensor was mounted on 10cm stands above the robot's `top_plate_link` frame. This resulted in a dense circularly symmetric point cloud (see Figure 7.3d) in front of the robot, but did not allow for mapping of terrain behind small obstacles (e.g. small rocks), leading to a short view range and sparse maps. The second configuration, depicted in Figure 7.3b, eliminates the drawback of the short view range by mounting the sensor with 10cm stands on a 51cm tall arch. However, obtained circularly symmetric point cloud (see Figure 7.3e) did not map surroundings in the robot's proximity. Therefore, the sensor was tilted by $-10°$ (depicted in Figure 7.3c), resulting in a dense planar-symmetric point cloud shown in Figure 7.3f. Further increases in the tilt angle resulted in a significant decrease in performance of the lidar odometry part of the localization method.

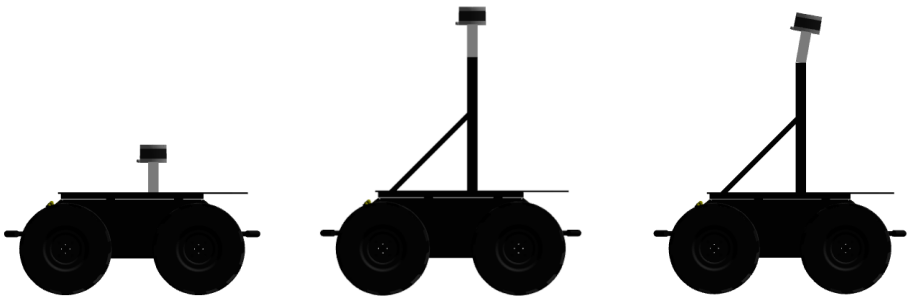
It is essential to know the sensor coordinate frames with respect to the robot's `base_link` frame in order to transform and fuse data properly. All sensor frames, including the robot's frame, are located in the robot-centered xz plane. Thus the planar description of their placement relative to the `base_link` frame is sufficient to describe relations between them (see Figure 7.4).

7.3 Navigation system settings

This section presents key parameter settings for the navigation system that were applied in all conducted experiments.

The key frame addition thresholds for the LIO-SAM were set to 0.5 m of translational motion and 0.2 rad of rotational motion of the robot. The Euclidean distance between the current keyframe and surrounding frames within that distance was set to 15 m in order to consider them for loop closure.

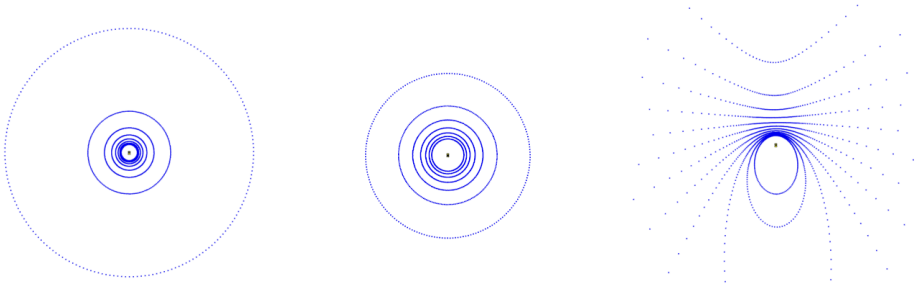
Given the chosen placement position of the lidar, the robot-centered square local elevation map was limited to a size of 10 by 10 m, with a resolution of 0.1 m to adequatly capture terrain details. The elevation map was initialized



(a) : Lidar mounted on 10cm stands right the top_plate_link frame.

(b) : Lidar mounted on 10cm stands on 51cm tall arch.

(c) : Lidar mounted on the arch, and tilted about $-10°$.



(d) : Point cloud, corresponding to the configuration in (a).

(e) : Point cloud, corresponding to the configuration in (b).

(f) : Point cloud, corresponding to the configuration in (c).

Figure 7.3: The lidar placements on the robot Husky (a), (b), (c) with the orthographic projections of the corresponding point clouds (d), (e), (f), taken from the 100m distance.

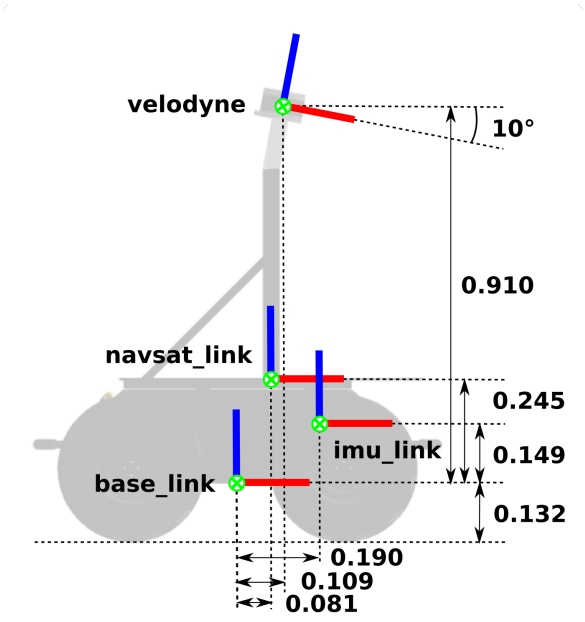


Figure 7.4: The relations between sensor coordinate frames and robot's body frame in the robot-centered xz plane.

with a robot-centered 5 by 5 m flat surface to provide an initial costmap for the planners. Additionally, the prior initialization of the elevation map prevents the traversability estimation component of the navigation system from identifying cells within a radius $r_o = 1.5$ m around the robot as obstacles, which would otherwise be classified as 'unknown'.

The radius used for patching of the elevation layer in the traversability estimation was set to $r_m = 0.15$ m and the radius for smoothing the patch layer and for calculating normals was set to $r_s = 0.35$ m.

For the DWA local planner's robot motion model, the maximum acceleration for translational and rotational velocities, $\dot{v}_{max}$ and $\dot{\omega}_{max}$, were limited to 2.5 $\mathrm{m} \cdot \mathrm{s}^{-2}$ and 3.2 $\mathrm{rad} \cdot \mathrm{s}^{-2}$, respectively. The maximum velocity of the robot was set to 1.0 $\mathrm{m} \cdot \mathrm{s}$. The number of samples was constrained to 6 and 20 for translational and rotational velocities, respectively, resulting in 120 sampled trajectories. The weights of the objective function were set to: $\alpha = 24.0$, $\beta = 0.01$, $\gamma = 32.0$.

For the local planner, the tolerance of the robot reaching a goal was set to 0.25 m and 0.1 rad.

7.4 Simulation preparation

This section focuses on preparations preceding the conducted experiments, including setting up the physics engine and modeling assets such as terrains for the simulated scenes. The created scenes, known as 'Gazebo worlds,' are presented at the end.

7.4.1 ODE settings

For the purpose of simulation, the natively supported Open Dynamics Engine (ODE) was selected due to its compatibility with the used ROS packages. The ODE offers a range of parameters, including the `real_time_update_rate`, which specifies the frequency at which the simulation time steps are advanced. In this study, the `real_time_update_rate` parameter was set to 0, indicating that the simulation runs as fast as possible based on the computing power. Another relevant parameter is the `max_step_size`, which specifies the time step for ODE's fixed-step solver. The default value in Gazebo is 1 ms, but this value was found to result in a significant decrease in real-time factor due to the limited computing power. Therefore, the `max_step_size` parameter was set to 1.6 ms to improve the simulation run-time performance and enable the simulation of the used 500Hz IMU sensor.

7.4.2 Assets for the simulated scenes

A simulated scene consists of SDF models that contain two separate 3D meshes: one for surface rendering, which is typically textured, and the other for collision detection. These meshes, in general, can be represented in various formats, such as STL, COLLADA, PLY, and FBX. The SDFormat, however,

only supports the COLLADA and STL formats. Therefore, the COLLADA format with the `.dae` extension is used for the created assets. It is worth noting that the higher the resolution of the collision mesh, the greater the negative impact on the run-time performance of the simulation. Consequently, low-resolution meshes that reasonably represent real scenes are preferred.

A special group of SDF models are terrains that represent ground surfaces in the form of a warped plane mesh. For the simulation, terrains with a 10cm resolution were created. However, such terrains cannot accurately represent a real location, which is necessary for navigation using offline Google maps. To address this, a reasonably accurate terrain that maps the real location was created from a topographic point cloud. The process of creating such terrain is described in subsection 7.4.3.

When a terrain is defined, it is populated with objects to give authenticity to the simulated scene. For this purpose, both publicly available[1][2][3] and custom-made meshes were used. The created ROS package, `gazebo_worlds`, comprises all generated SDF models, including both publicly available and custom-made meshes. In addition to the SDF models, the package contains Gazebo `.world` files, which form entire scenes, and the corresponding ROS `.launch` files, which are used for launching a given `.world` file through ROS in Gazebo.

The popular open-source 3D creation suite, Blender[84], was used for creating the assets, along with the mesh2sdf tool (see Appendix A), which allows for the conversion of textured `.dae` files to SDF models.

7.4.3 Process of generating a georeferenced SDF terrain from a topographic LAZ point cloud

Complex scenes can be assembled using assets created for example in Blender. However these scenes do not accurately imitate an actual terrain, thence the built-in offline Google Maps in MapViz cannot be used to set a specific target goal in the proposed navigation method.

To obtain information about an actual terrain, lidar mapping or photogrammetry methods [85] could be used. However, creating one's own mapping of a terrain is outside the scope of this thesis. Instead, publicly available terrain data in the form of ready-to-use point cloud was used. One such source of terrain data is OpenTopography[4], which provides topographic point clouds of locations around the world. For the purposes of simulating a ground mobile robot, dense point clouds are preferred for surface reconstruction because they result in a detailed mesh. Additionally, the mapped terrain should be suitable for a wheeled robot to operate in, including roads and 2D manifolds. After searching through OpenTopography's database, the only dataset that met these requirements was the topographic point cloud of Skyline Drive

[1] `https://www.cgtrader.com/free-3d-models/exterior/landscape/low-poly-forest-nature-set-free-trial`

[2] `https://www.cgtrader.com/items/3903752/download-page`

[3] `http://models.gazebosim.org/`

[4] `https://www.opentopography.org/`

Road Area, Cañon City, Colorado[86]. This dataset maps an area of 0.42 km^2 with an average point density of 243.29 points/m^2, and includes both asphalt and rough terrain roads.

The point cloud is in the form of a LAZ file, which is a compressed version of the flexible LAS format. The points, in this case, are encoded in the [p_x, p_y, p_z, R, G, B] format, with spatial values given relative to the encoded local coordinate frame. The following steps of generating georeferenced SDF terrain from a topographic point cloud are summarized in Figure 7.5.

The dataset processing begins in CloudCompare[87], an open-source software for 3D point cloud and mesh processing. The first step is to translate the point cloud's local coordinate frame to the origin of CloudCompare's coordinate system. This results in smaller coordinate values, which helps to prevent rounding errors because both CloudCompare and MeshLab[88], which is used later in the process, employ the OpenGL library, which does not handle large coordinate values well. The second step is to use the ***Segment tool*** to select only the desired area of the point cloud, if necessary. This step is optional. The third step is to pick a "distinct" point [p_{d_x}, p_{d_y}] in the point cloud, which will be used for georeferencing the terrain later. In this case, a point representing a sharp peak of a rock was chosen. Finally, the point cloud is exported in the colored PLY format.

For the textured surface reconstruction, the open-source software for processing and editing 3D triangular meshes MeshLab is used. The first step is to calculate a normal for each point in the point cloud based on its neighboring points, serving as a prerequisite for the next step. This is done using tool ***Filters → Normals, Curvature and Orientation → Compute Normals for Point Sets***. In this case, 8 nearest neighbors were used. The second step is to reconstruct the surface using the Screened Poisson algorithm [89] with ***Filters → Remeshing, Simplification and Reconstruction → Surface Reconstruction: Screened Poisson***. The `Reconstruction depth` and `Interpolation weight` parameters were set to 12 and 4, respectively, to produce a detailed mesh. The Screened Poisson implementation in MeshLab uses Neumann boundary conditions [89], which means that the reconstructed surface will have square boundaries, including the desired reconstructed area. To remove the unwanted square boundary the ***Filters → Selection → Select faces with edge longer than*** tool is used to crop the surface to the shape of the point cloud. The reconstruction process may also create isolated meshes that are not connected to the main mesh, which can be removed using the ***Filters → Cleaning and Repairing → Remove isolated pieces*** tool. To create a texture for the mesh, the tool ***Filters → Texture → Parametrization per Triangle*** is used. In this case, the texture was initialized to 16384x16384 pixels. The color information from the point cloud is then transferred to the texture using the ***Filters → Texture → Transfer Vertex Attributes to Texture*** tool. Finally, the reconstructed mesh is exported as an OBJ file along with a PNG texture file and an MTL file that provides the texture definition to the surface object.

The model was georeferenced using boundary GPS coordinates lat_0, lat_1,

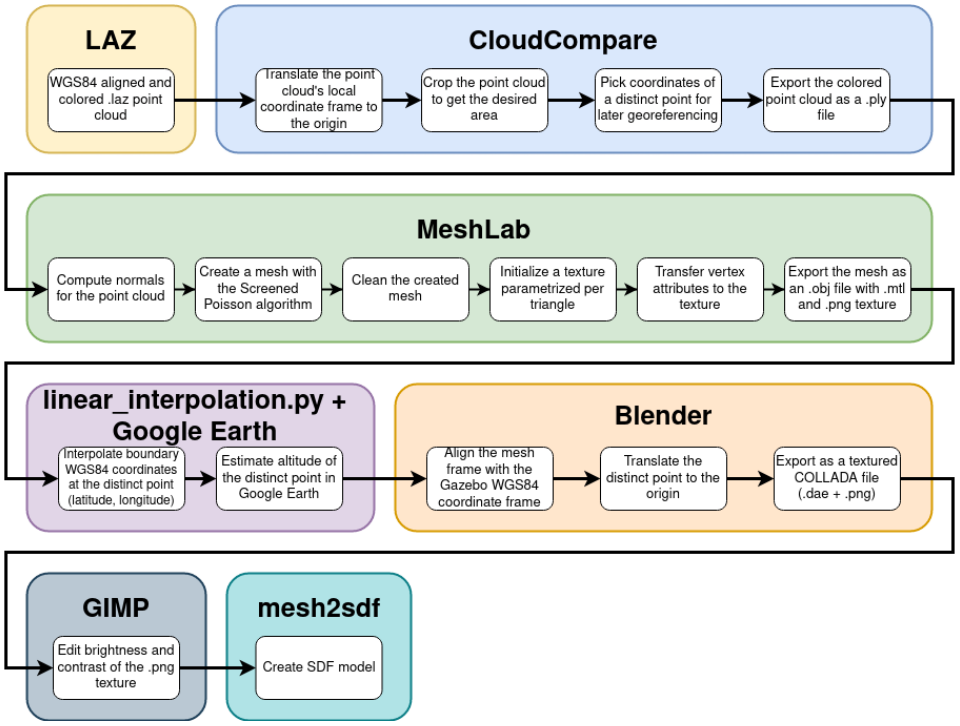


Figure 7.5: The process of generating an SDF terrain from a LAZ topographic point cloud.

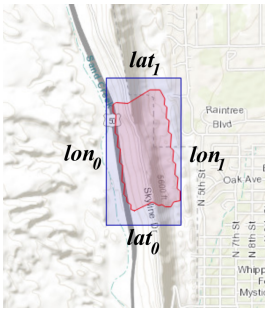
lon_0, lon_1 that define the mapped area and were provided with the dataset. The latitude and longitude of the "distinct" point p_{lat}, p_{lon} were calculated by interpolating the boundary GPS coordinates using linear interpolation, given by the formulas:

$$p_{lat} = lat_0 + (p_{d_y} - p_{y_0})\frac{lat_1 - lat_0}{p_{y_1} - p_{y_0}}, \qquad p_{lon} = lon_0 + (p_{d_x} - p_{x_0})\frac{lon_1 - lon_0}{p_{x_1} - p_{x_0}},$$

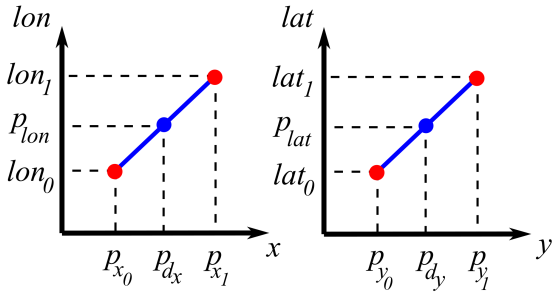
where p_{y_0}, p_{y_1}, p_{x_0}, p_{x_1} are coordinates given in the local coordinate frame of the dataset, corresponding to the lat_0, lat_1, lon_0, lon_1 boundary GPS coordinates. The altitude of the "distinct" point was then estimated in Google Earth[90] using the calculated horizontal coordinates p_{lat}, p_{lon}.

The next step is to create an asset from the OBJ model in Blender. Firstly, the imported mesh is aligned with the Gazebo WGS84 coordinate frame. This means that the positive direction of x-axis in Gazebo corresponds to positive direction of latitude and the positive direction of y-axis in Gazebo corresponds to negative direction of longitude. Secondly, the “distinct” point on the aligned mesh is translated to the origin in Blender, so that when it is later imported into Gazebo, the “distinct” point will be identical to the origin. Lastly, the mesh is exported as a textured COLLADA file, which is suitable for generating an SDF model.

The texture created using MeshLab has dimmer colors than the topographic point cloud. Therefore, the texture is imported into the open-source image editor GIMP [91] in order to increase its brightness and contrast. However,



(a) : The range of the Skyline Drive Road Area dataset (red) and given GPS boundaries (blue). Obtained from [86].



(b) : The linear interpolation of GPS boundaries lat_0, lat_1, lon_0, lon_1 to calculate the GPS coordinates p_{lat}, p_{lon} of a "distinct" point $[p_{d_x}, p_{d_y}]$ given in the local coordinate frame of the Skyline Drive Road Area dataset.

Figure 7.6: The provided GPS boundaries of the Skyline Drive Road Area dataset (a) and the linear interpolation of the GPS boundaries to calculate the GPS coordinates of the "distinct" point.

this step does not affect the results of the experiments.

The final step is to use the mesh2sdf tool (see Appendix A) to create an SDF model. Once this step is completed, the asset is ready to be used when composing a scene.

7.5 Performed experiments

This chapter presents the experiments that were conducted. In order to provide a more thorough understanding of the performance of the proposed navigation method, a video summarizing all of the experiments was created and can be accessed through the link provided in Appendix C.

7.5.1 Traversability estimation reflecting robot specifications

The aim of this experiment was to determine appropriate values for the s_{max} and r_{max} parameters of the traversability estimation function so that the resulting traversability value reflects the robot's specifications, namely maximum climb grade, maximum traversal grade, and the height of the front bumper (as described in section 7.2.1). Since the maximum traversal grade value of 30 degrees is smaller than the maximum climb grade, we limit the maximum climb grade to this value. According to the robot dimensions, the front bumper is located in height of 22 cm above the ground, meaning that the robot would be able to safely traverse a height difference of up to 20 cm. However, taking into account terrain imperfections, the maximum traversable height difference was limited to 15 cm.

The procedure of estimating ideal values of s_{max} and r_{max} is straightforward. Given a map resolution of 10 cm, the two-dimensional space of s_{max} and r_{max} is evenly sampled, resulting in 24 combinations. Each combination

(s_{max_i}, r_{max_i}) is tested in a simulation in which the traversability estimation component of the navigation system assesses the traversability of terrain in the form of ramps and kerbs. A combination (s_{max_i}, r_{max_i}) that meets the requirements for maximum obstacle height difference and maximum traversal grade will be selected for the following experiments.

Scene

For this experiment, we have utilized two scenes. The first scene named `platform_playground.world` represents a flat surface measuring 20x20 m with obstacles in the form of concrete barriers surrounding the area. In addition, the scene contains platforms of heights 0.05, 0.1, 0.15, and 0.2 m, simulating kerbs commonly found in urban environments (shown in Figure 7.7).

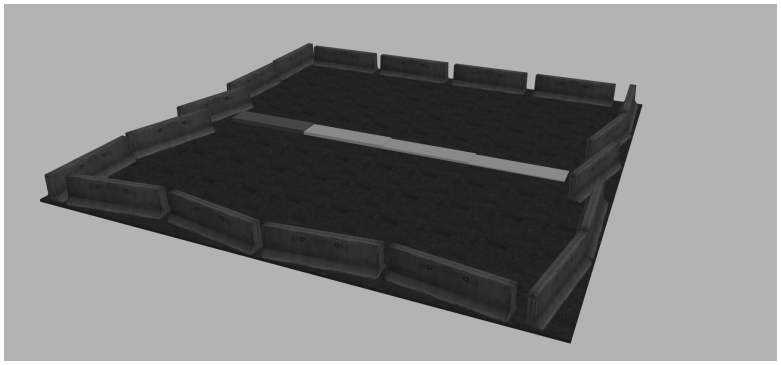


Figure 7.7: The platform_playground.world scene.

The second scene, `ramp_playground.world`, represents a flat surface measuring 100x20 m that is fenced with concrete barriers. It contains two sets of ramps with an elevation angle ranging from 10 to 45 ° in increments of 5 ° (depicted in Figure 7.8). Additional assets in the form of poles were added to the scene to provide more of detectable features for the lidar odometry.

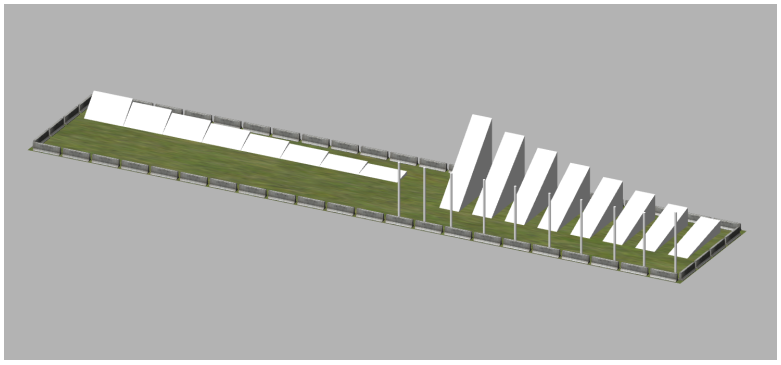


Figure 7.8: The ramp_playground.world scene.

Results and evaluation

All combinations resulting from a Cartesian product of sets

$$S = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\} \ni s_{max_i} \text{ [rad]},$$

$$R = \{5, 10, 15, 20\} \ni r_{max_i} \text{ [cm]}$$

were tested in both scenes with results depicted in Figure 7.9. The traversability weight function classified 15cm kerb and 30° ramp as non-traversable using orange-colored combinations. However, these combinations can be used in more restrictive traversability assessments. Using the green-colored combinations, the weight function evaluated all kerbs and ramps up to 15 cm and 30 °, respectively, as traversable. When using the red-colored combinations, the non-traversable obstacles were classified as traversable, therefore cannot be used. From the two graphs, we obtained 3 suitable candidates by taking the overlapping green combinations and chose to use the average of these values, i. e. $s_{max} = 0.3$ rad and $r_{max} = 0.15$ m, for further experiments.

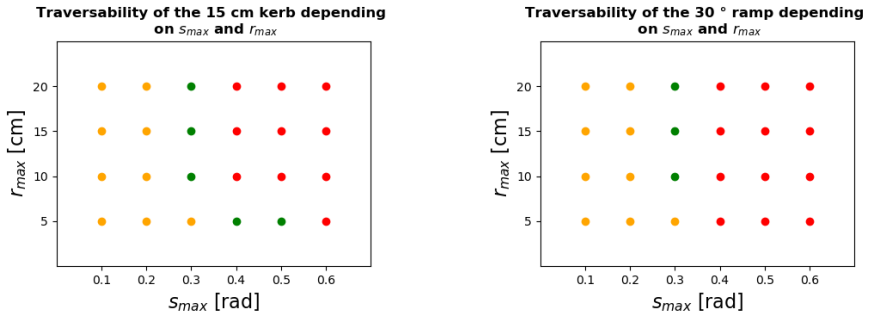


Figure 7.9: The traversability of a 15cm kerb and a 30° ramp that represent maximum allowed height difference and slope, reflecting robot specifications. Depending on a combination of values assessed to parameters s_{max} and r_{max}, the traversability weight function evaluated the kerb and ramp as non-traversable (orange), traversable (green), traversable, but the kerbs and ramps above the maximum allowed height difference and slope also appeared traversable (red).

Figure 7.10 depicts the grayscale traversability maps of kerbs and ramps, respectively, where black indicates completely non-traversable cells. These figures also show the local costmap from `move_base`, in which pink cells represent obstacles, blue represents an inflation around them.

When the robot traversed 30 ° and 25 ° ramps, it experienced significant slippage despite the high simulated friction between its tires and the surface, likely due to simulation inaccuracies. The effect of adding imaginary obstacles in the traversability estimation component of the navigation system can be seen in Figure 7.11, where the robot traversed a 20 ° ramp and got near the edge of the ramp. The unmapped cells in the defined radius around the robot were changed to non-traversable, i. e. obstacles, to prevent the robot from potential fall during navigation.

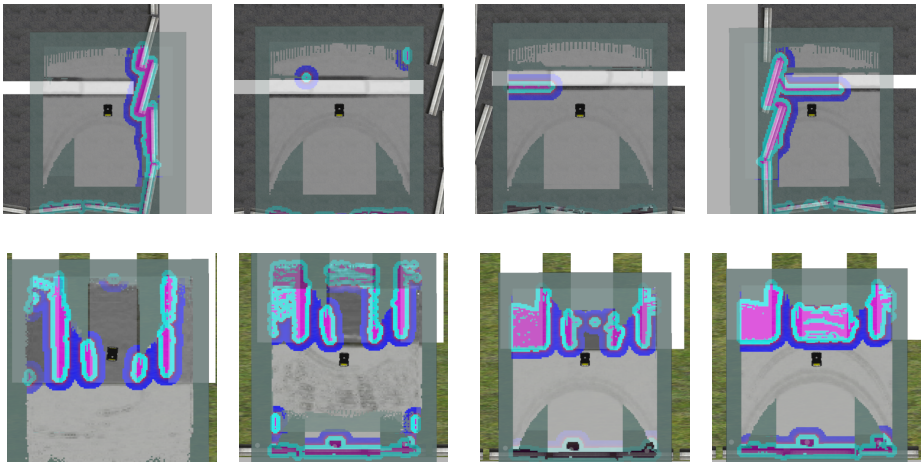


Figure 7.10: The traversability of a 5, 10, 15, and 20cm kerbs (top), and 20, 25, 30, and 35 ° ramps (bottom), respectively, using $(s_{max}, r_{max}) = (0.3, 0.15)$ [rad, m]]. The scene is overlaid with the grayscale traversability map, where black indicates non-traversable cells, along with the local costmap from move_base showing obstacles (pink), corresponding to the non-traversable cells, with an inflation around them (blue).

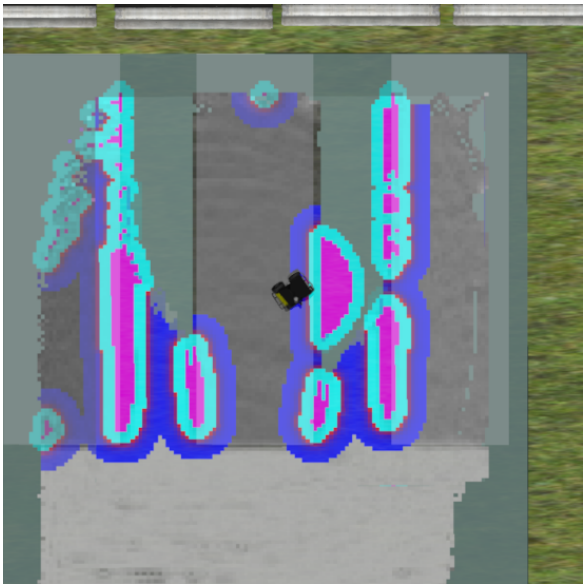


Figure 7.11: The robot traversing a 20° ramp. Imaginary obstacles (pink) were added to the map (right side of the ramp), when the robot got near the edge of the ramp, to prevent it from potential fall during navigation.

7.5.2 Navigating from an enclosed area

The purpose of this experiment was to evaluate the capability of the navigation system to determine that a designated goal location is unreachable. The robot was required to map the environment and continuously replan its path to the goal, while also avoiding collisions with obstacles, based on the terrain conditions. If the global planner is unable to find a path to the goal, the robot is expected to execute a recovery behavior in order to remap its surroundings and identify any potential changes in the environment that might allow for a route to the specified location.

Scene

For the purpose of this experiment, the scene `empty_playground.world` represents a flat surface measuring 20x20 m with obstacles in the form of concrete barriers surrounding the area (see Figure 7.12). Additional assets in the form of poles were added to the scene to provide more of detectable features for the lidar odometry.

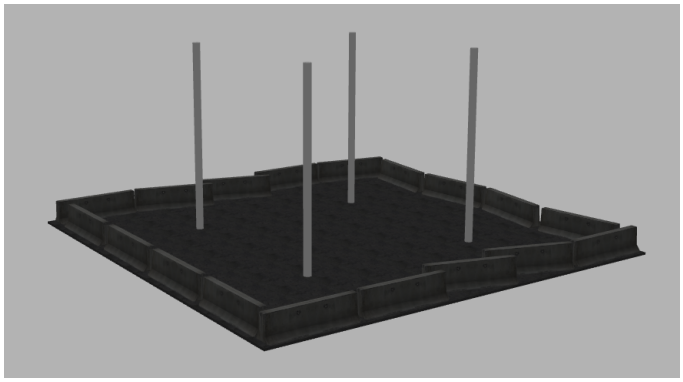
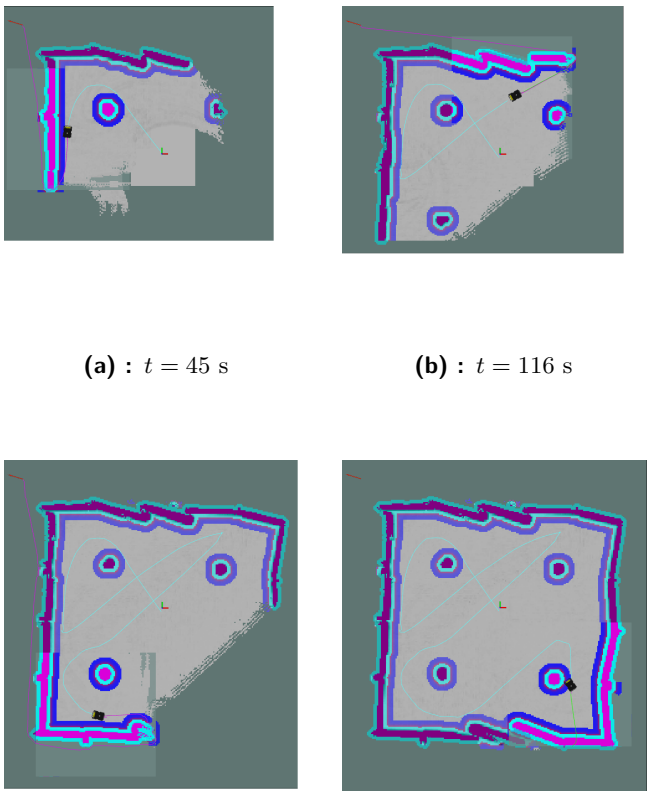


Figure 7.12: The empty_playground.world scene.

Results and evaluation

The robot started at global coordinates $(x, y) = (10, 11)$ m. The designated goal was set outside the enclosed area. During the navigation process, the robot mapped the entire enclosed area until the global planner could not find a path to the goal, while avoiding present obstacles. Upon executing the recovery behavior, no changes in the environment were observed, and the navigation to the goal was aborted by `move_base`. The entire run took 5 minutes and 32 seconds, with an average real-time performance of 0.27x.

Figure 7.13 illustrates the global and local traversability maps, the corresponding `move_base` costmaps, the robot, and the current global plan at various time stamps. It can be observed that the global map is slightly rotated in relation to the scene, a result of the map optimization process of the LIO-SAM. Figure 7.14 offers a comparison of the ground truth and the LIO-SAM positions for reference.



(a) : $t = 45$ s **(b) :** $t = 116$ s

(c) : $t = 216$ s **(d) :** $t = 332$ s

Figure 7.13: The process of navigation attempting to navigate from an enclosed area. The pictures taken at various time stamps show the goal, indicated by the red arrow, the current global path in purple, the trajectory from LIO-SAM in light blue, the grayscale traversability map, where black indicates non-traversable cells, along with the local costmap from move_base showing obstacles (pink), corresponding to the non-traversable cells, with an inflation around them (blue).

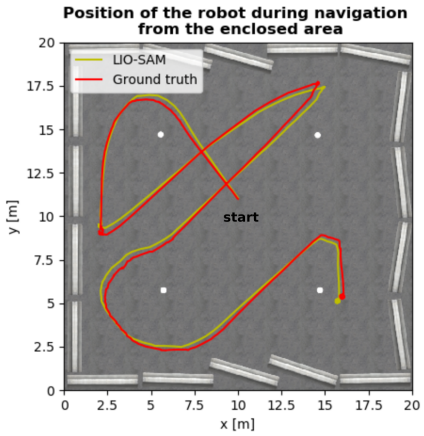


Figure 7.14: Comparison of the ground truth positions (red) and the estimated poses of the robot by the LIO-SAM (yellow) during navigation from the enclosed area.

7.5.3 Navigating through a complex environment

The aim of this experiment was to evaluate the capability of the navigation system to navigate safely to a predetermined destination through a complex and uneven terrain with potentially hazardous situations for the robot, such as falling into a trench or sliding down a hill. Additionally, navigation with the same starting point and destination was repeated in order to determine the success rate of the navigation method.

Scene

This experiment utilizes the `forest_terrain.world` scene that represents an uneven terrain of a forest environment, measuring 40x40 m, with various types of trees, bushes, stumps, and rocks. A bridged trench is also present along with an uphill zigzag road leading to the hilltop (visualized in Figure 7.15).

The elevation of the ground surface is illustrated in Figure 7.16. The traversability map of the entire terrain, also depicted in the Figure 7.16, which was generated offline using the traversability estimation component of the navigation system, displays the yellow flat surface and the green traversable road, which the robot should be able to follow when navigating to the hilltop.

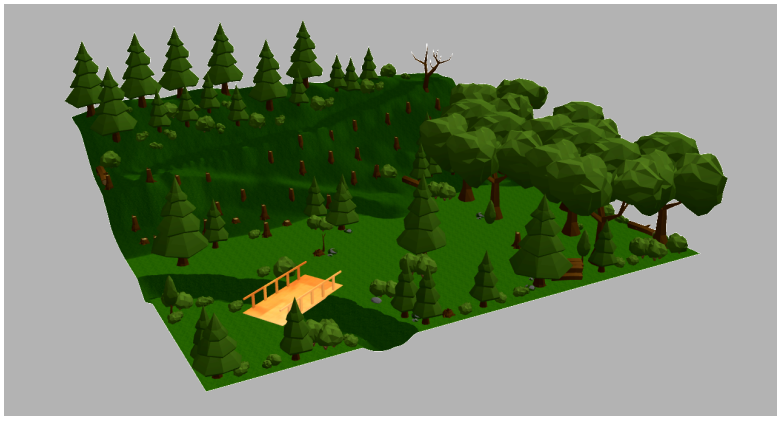


Figure 7.15: The forest_terrain.world scene.

Results and evaluation

The robot's starting position was at global coordinates $(x_s, y_s) = (4.5, 4.5)$ m and the designated goal was set to $(x_g, y_g) = (38, 38)$ m. Initially, the robot navigated directly towards the goal until it encountered a non-traversable hill. In response, the robot turned right and began exploring the rightmost area of the map, as it was determined to be the shortest path to the goal. Hovewer, after mapping this area, the path was revised and the robot was directed to take the uphill road, which it followed to reach the goal. The ground truth and the estimated positions provided by the LIO-SAM was almost identical

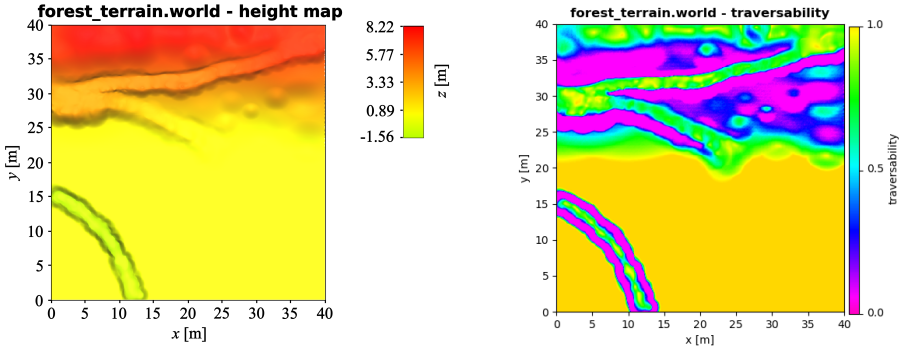


Figure 7.16: The height and the traversability maps of the ground surface of the forest_terrain.world, respectively. The pink areas in the traversability map represent non-traversable cells.

(shown in Figure 7.17), indicating that there were no deformations of the global map relative to the scene. Figure 7.18a presents a topographic view of the scene with the overlaid global traversability map and the estimated robot poses. In addition, Figure 7.18b shows the global costmap from `move_base`.

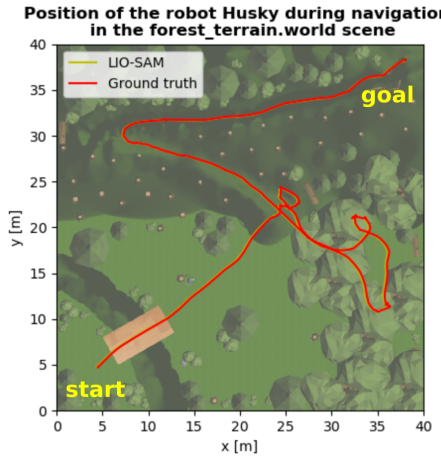
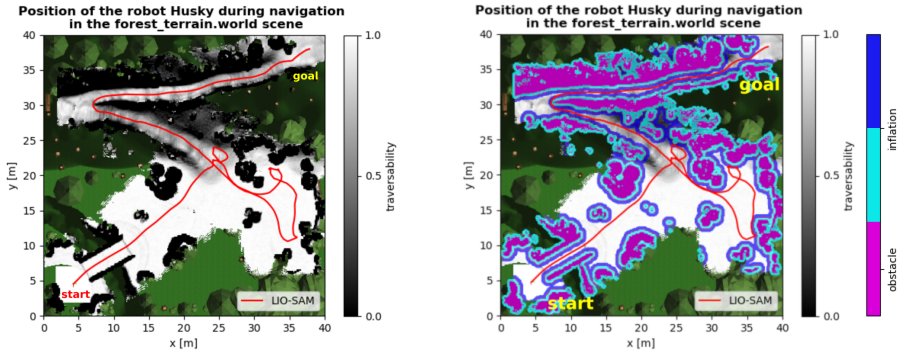


Figure 7.17: The comparison of the ground truth positions (red) and the estimated poses of the robot by the LIO-SAM (yellow) during navigation through the forest_terrain.world.

To evaluate the success rate of the navigation method, this experiment was repeated nine more times. Figure 7.19 illustrates the ground truth poses of the robot in all ten runs of the navigation method, during which the 2^{nd} and the 6^{th} runs ended midway through the route. The average simulation runtime of the successful navigation runs was approximately 27 minutes, with an average real-time performance 0.25x.

As can be seen in the Figure 7.17, the robot struggled on the right hillside at the beginning of the uphill road due to the increasing slope of the terrain, which had initially appeared traversable. However, as the robot further



(a) : The global traversability map. **(b) :** The global costmap from move_base.

Figure 7.18: The topographic view of the forest_terrain.world scene overlaid with the global traversability map and the estimated robot poses (left), the added global costmap from move_base (right).

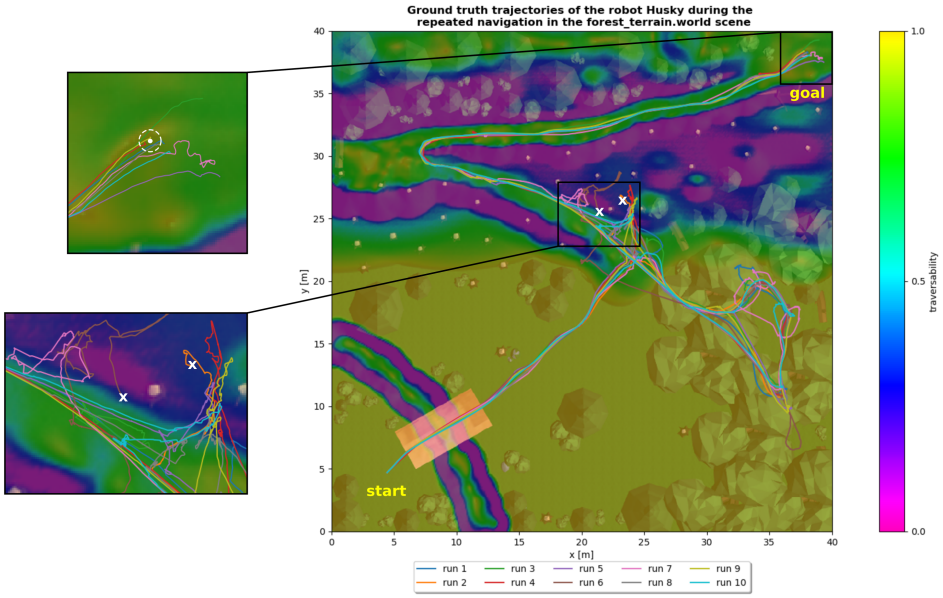


Figure 7.19: The ground truth estimated poses of the robot in ten conducted runs. The background represents the topographic view of the scene overlaid with the offline calculated traversability map of the ground surface. The top zoom shows the goal with the 0.25m tolerance radius (white) and positions, where the navigation runs ended up. The bottom zoom vizualize the hillside with locations (crosses), where the navigation method failed.

mapped the area towards the hillside, it became surrounded non-traversable terrain on three sides. As a result, the global path was replanned in a way that made in difficult for the local planner to turn the robot in place on the steep hill. This caused the local planner to fail to produce a trajectory in the 2^{nd} run, causing the robot to stop moving.

During the 6^{th} run, the robot took a slightly different path, which led it to pass the area where the 2^{nd} run had failed. However, the later estimated terrain traversability caused the global planner to plan a path towards the hillside. The robot successfully turned in place on the steep hill, but on the way back the robot slided towards the road, causing the LIO-SAM to incorrectly register lidar scans and the odometry to fail.

At the end of the eight successful runs, odometry began to slowly deviate in half of the runs as the number of detected features decreased as the robot approached the edge of the scene. This resulted in an inconsistent global map, in which the planners were still able to navigate to the goal, which deviated from the designated location in the scene. This odometry deviation was solely due to the limited size of the scene.

This experiment posed rather an extreme test of the proposed navigation method. In a real application, where an estimate of a traversable route is available, a series of waypoints leading to the designated goal would be provided to the navigation system, rather than simply the goal itself. This would prevent unnecessary exploration of the environment (as shown in Figure 7.19).

7.5.4 Navigating through a reconstructed real environment

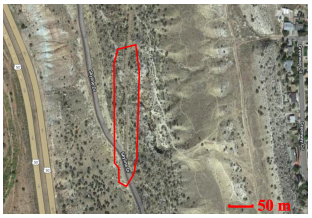
The purpose of this experiment was to evaluate the capability of the navigation method to navigate through an environment that closely resembles real terrain. Additionally, the designated goal was set in WGS84 LLA coordinates using MapViz.

Scene

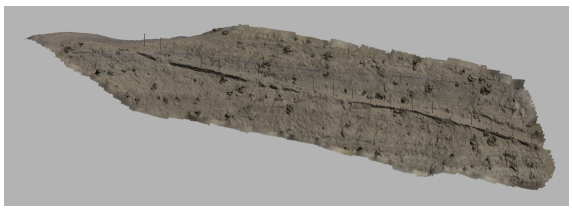
For the purpose of this experiment, we utilized the `skyline_drive_road.world` scene. The scene, measuring approximately 300x60 m, is a detailed replica of part of the Skyline Drive Area, Cañon City, Colorado, shown in Figure 7.20a. The terrain was derived from a topographic point cloud [86], publicly available from OpenTopography[5]. The process used to create the scene is described in a subsection 7.4.3. The resulting terrain is shown in Figure 7.20b.

The height map of the terrain is illustrated in Figure 7.21. The traversability map of the entire terrain, depicted in the Figure 7.22, was generated with the same procedure as in the previous experiment. The traversability map shows a yellow-green traversable road that leads across the entire scene, which the robot should follow.

[5] `https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.062021.32613.1`



(a) : The reconstructed area shown in Google maps [92].



(b) : The skyline_drive_road.world scene.

Figure 7.20: The area of interest (a) and the reconstructed terrain (b) of the Skyline Drive Road area.

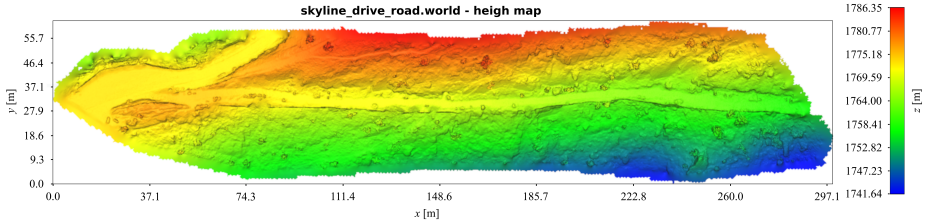


Figure 7.21: Height map of the skyline_drive_road.world scene.

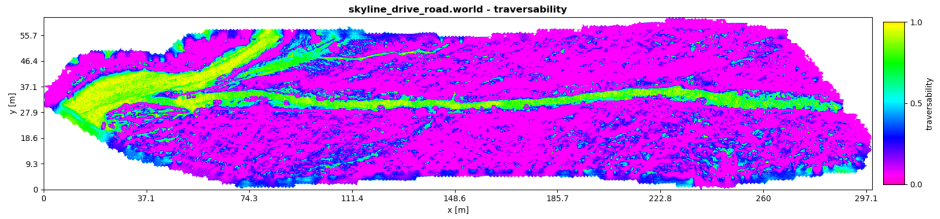


Figure 7.22: Traversability map of the skyline_drive_road.world scene. The pink areas in the traversability map represent non-traversable cells.

Results and evaluation

The robot's starting position was at global coordinates $(x_s, y_s) = (27, 32)$ m and the designated goal was set to $(lat, lon) = (38.46419, -105.25164)$, corresponding to the global cartesian coordinates $(x_g, y_g) = (275, 31)$ m.

However, after the navigation system was launched, odometry provided by the LIO-SAM failed immediately due to the small ammount of detected features in the environment. Therefore, additional assets in the form of the telegraphic poles were added to the scene along the road to provide detectable features. Despite these efforts, the LIO-SAM's odometry still failed to accurately match detected features.

As a solution, the odometry provided by the LIO-SAM was replaced with ground truth poses of the robot for the purpose of simulating an incomplete navigation system. This ultimately resulted in a successful navigation to the designated goal, which took approximately 42 minutes of the simulation

time with an average real-time performance of 0.25x. Figure 7.23 show a topographic view of the scene with the overlaid global traversability map and the ground truth robot poses. In addition, Figure 7.24 shows the global costmap from `move_base`.

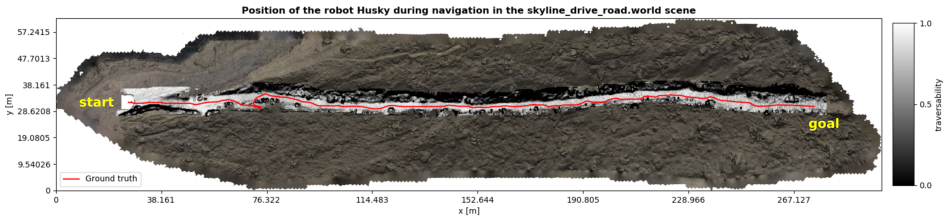


Figure 7.23: Topographic view of the skyline_drive_road.world scene overlaid with the global traversability map and the ground truth robot poses (red).

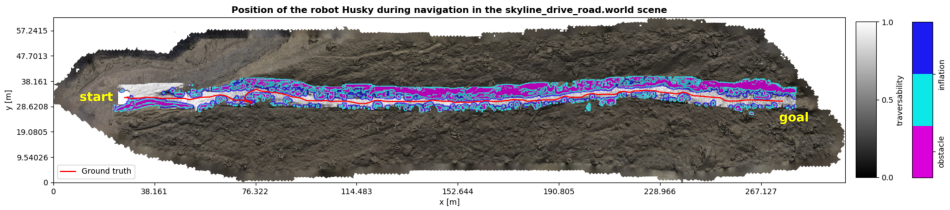


Figure 7.24: Topographic view of the skyline_drive_road.world scene overlaid with the global traversability map, the ground truth robot poses (red), and with the global costmap from move_base.

7.6 Limitations of the proposed method

7.6.1 Expected limitations

The first limitation of the navigation method arises from the selected map 2.5D representation, which is not capable of mapping multi-level environments. In an attempt of mapping multi-level environment, the global map gets overwritten with the robot mapping a new level. This could lead to misleading the global planner in cases when a newly-mapped level's traversable area connects the traversable areas of the global map, which is being overwritten, resulting in invalid global paths. However, this limitation was not confirmed by an experiment.

Another limitation related to mapping is the lack of optimization of the global map created from stitching together local traversability maps, which can result in inconsistency with the internal map of the LIO-SAM when a significant graph optimization is performed. However, this limitation was not fully confirmed in simulation due to the limited size of the scenes, which did not allow for major factor graph adjustments. A minor impact of the graph optimization on the global map inconsistencies was observed at the

beginning of experiment 7.5.2, where the internal map of the LIO-SAM was slighty rotated after first seconds of running the navigation system.

Additionaly, the navigation method is incapable of operating in dynamic environments due to the selected odometry method. The LIO-SAM matches extracted features from consecutive lidar scans and features extracted from dynamic obstacles will likely break the odometry [28]. Therefore, the proposed method is limited to static environments only.

Finally, the method was expected to run with a low real-time performance due to extensive and frequent memory allocation in the process of traversability estimation and the incorporation of incoming local traversability maps into the global map. This limitation was confirmed in navigation through a complex environment 7.5.3, where the method ran with an average real-time performance of 0.25x.

7.6.2 Observed limitations

When exploring possible placements of the lidar sensor, described in section 7.2.3, tilting the sensor more than –15 ° around its y axis shown a significant impair in the accuracy of odometry provided by the LIO-SAM. This was due to the smaller ammount of points in lidar scans, concentrated almost only in the near proximity in front of the robot, therefore decreasing the number of detectable features during the process of feature extraction.

The effect of insuficient ammount of features was observed in experiment 7.5.4, where the detected features were matched incorrectly, resulting in incorrect localization of the robot.

In experiment 7.5.3, it was observed that the DWA local planner struggled to find a valid trajectory when the robot was suddenly surrounded on three sides by non-traversable areas and had to move backwards. This resulted in a ‘no-path-found’ scenario and the navigation was terminated.

Chapter 8

Future work

The next phase will focus on optimizing the proposed method in terms of simplicity and memory efficiency.

Currently, the implementation of the traversability estimation involves a pipeline of mathematical operations, each of which requires its own data layer. Some of these operations can be combined, for example in the calculation of terrain slope, which involves calculating the normal to the surface and subsequent calculation of the slope of that vector. Additionally, certain computations evaluating the mathematical expressions can be parallelized, resulting in a shorter processing time for a single elevaion map.

The implemented global map update allocates a new empty map with a new local map, and is overwritten with the current global and the local map, resulting in an updated global map. To reduce the extensive memory allocation and data copying, a new empty map could be allocated less frequently and expanded in the direction of the moving robot to the extent that multiple consecutive local maps will fit within its area. This would allow for direct overwriting of the global map with incoming local maps and a decrease in memory allocation, leading to increase in real-time performance.

A major modification to the proposed method could involve integrating a more advanced version of the LIO-SAM. Improving the odometry, specifically the lidar odometry factor, could be achieved by incorporating the IA-LIO-SAM method, which utilizes intensity and ambient values of lidar scans to provide more stable features for scan matching. Another possibility that extends the current method to dynamic environments is to use the LIO-CSI method, which performs semantic-based feature extraction and loop-closure, filtering out dynamic objects in the data, therefore providing more robust map-based localization.

Additionally, it is necessary to verify the capabilities and limitations of the proposed navigation method using a real robotic platform, such as the Husky lab robot. However, it should be noted that the current laboratory equipment runs ROS2, which would require the implementation of the proposed method to be migrated to this version of ROS.

Chapter 9

Conclusions

In order to navigate a robot through a rough terrain environment, the robot needs to be able to map its surroundings and localize within the mapped environment, using its sensor suite.

For this purpose, various sensors commonly used in mobile robotics were evaluated for their suitability in navigation, resulting in the selection of lidar as the primary sensor. Subsequently, lidar-based Simultaneous Localization and Mapping (SLAM) techniques, with a focus on graph-based methods, and map representations, suitable for the specified task, were reviewed.

The proposed navigation method integrates the Lidar-Inertial Odometry via Smoothing and Mapping (LIO-SAM) method to estimate the robot's pose using an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS) receiver in addition to lidar. The navigation method utilizes a probabilistic mapping approach that incorporates pose estimates and raw lidar measurements, resulting in a 2.5D robot-centric elevation map. A geometric-based approach that uses a weighted sum of the slope and roughness of the terrain, is applied to the elevation map to classify the local terrain as either traversable or non-traversable (obstacles). The global map is created by merging consecutive local traversability maps. The path planning component of the navigation system combines a global and local planner. The global planner, based on Dijkstra's algorithm, receives a designated goal in GPS coordinates, set through a user interface that incorporates offline maps, and generates a suitable path in the global map, while the local planner, using the Dynamic Window Approach (DWA), plans on the local traversability map, smooths the global path, informs the global planner to replan the route in the event of collision of the current global path with obstacles, and produces velocity control commands for the robot in the form of instantaneous twist velocities.

The proposed navigation system was implemented as a ROS project that integrates various pre-existing packages to address specific aspects of the navigation method. In order to test the system, assets were created and assembled into scenes in Gazebo simulator. Furthermore, the utilized sensors were modeled after actual products and the placement of the lidar was considered to provide optimal coverage of the terrain in front of the robot.

In the first conducted experiment, an optimal combination of parameters in

the traversability estimation weight function was identified by evaluating the traversability of a ramp and kerb that represented the maximum traversable slope and height difference for the robot. Subsequently, navigation from an enclosed area demonstrated the behavior of the method, where the robot executed a recovery behavior after exploring the entire area and failing to find a valid path. Further experiments were conducted in a forest terrain and on a straight trail based on an actual location, and the method was found to be quite effective in estimating traversable terrain and generating corresponding routes. In the final experiment, the robot's pose estimation failed due to incorrectly matched features in lidar odometry. In order to test the remaining parts of the navigation system, the LIO-SAM was replaced with ground truth poses.

In conclusion, the proposed method addresses the navigation problem in an unknown environment by planning safe paths according to the estimated traversability of the mapped terrain and the designated goal, and generating appropriate control commands to guide the robot along the planned route to the destination. However, the method is limited to single-level environments due to the chosen map representation and is coinstrained to static environments by the selected odometry approach. Furthermore, the success rate of the method strongly depends on the environment in terms of number of detectable features for lidar odometry and occurrence of narrow dead ends, which lead to local planner failures.

The further steps will focus on optimizing the real-time performance, alternative methods of feature extraction in lidar odometry, and confirmation of the capabilities and limitations of the proposed navigation method on a real robotic system.

Bibliography

[1] N. Nourani-Vatani, J. Roberts and M. V. Srinivasan, "Practical visual odometry for car-like vehicles," 2009 IEEE International Conference on Robotics and Automation, 2009, pp. 3551-3557, doi: 10.1109/ROBOT.2009.5152403.

[2] O. Maklouf, A. Adwaib, "Performance evaluation of GPS INS main integration approach," World Academy of Science, Engineering and Technology, International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering, 2014, vol. 8, no. 2, pp. 476–484, doi: 10.5281/zenodo.1092603.

[3] T. Yang, Y. Li, C. Zhao, D. Yao, G. Chen, L. Sun, T. Krajník, and Z. Yan, "3D ToF LiDAR in Mobile Robotics: A Review," ArXiv, vol. abs/2202.11025, 2022, doi: 10.48550/ARXIV.2202.11025.

[4] R. Mázl, "Lokalizace pro autonomní systémy," Ph.D. dissertation, Czech Technical University in Prague, 2007. [Online]. Available: `https://www.yumpu.com/xx/document/view/28149323/lokalizace-pro-autonomni-systemy`

[5] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: Types, approaches, challenges, and applications," SpringerPlus, vol. 5, no. 1, 2016.

[6] A. El-Rabbany, Introduction to GPS: The global positioning system. Boston, MA: Artech House, 2006.

[7] B. Siciliano and O. Khatib, Springer Handbook of Robotics. Berlin: Springer, 2016.

[8] F. Dellaert and M. Kaess, "Factor graphs for robot perception," Foundations and Trends in Robotics, vol. 6, no. 1-2, pp. 1–139, 2017, doi: 10.1561/2300000043.

[9] Yang J., Li Y., Cao L., Jiang Y., Sun L., Xie Q, "A Survey of SLAM Research based on LiDAR Sensors," International Journal of Sensors, 2019, vo. 1, no. 1, pp. 1003.

[10] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti and D. Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 5135-5142, doi: 10.1109/IROS45743.2020.9341176.

[11] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," Sensors, vol. 20, no. 7, p. 2068, 2020, doi: 10.3390/s20072068.

[12] J. Zhang and S. Singh, "Loam: Lidar Odometry and mapping in real-time," Robotics: Science and Systems X, 2014, doi: 10.15607/rss.2014.x.007.

[13] K. Chen, K. Zhan, F. Pang, X. Yang, and D. Zhang, "R-LIO: Rotating lidar inertial odometry and mapping," Sustainability, vol. 14, no. 17, p. 10833, 2022, doi: 10.3390/su141710833.

[14] Q. Tong, C. Shaozu. 2019. *A-LOAM*, [Source code]. Available: `https://github.com/HKUST-Aerial-Robotics/A-LOAM`

[15] K. Giseop. 2021. *SC-A-LOAM*, [Source code]. Available: `https://github.com/gisbi-kim/SC-A-LOAM`

[16] G. Kim and A. Kim, "Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 4802-4809, doi: 10.1109/IROS.2018.8593953.

[17] H. Wang, C. Wang, C. -L. Chen and L. Xie, "F-LOAM : Fast LiDAR Odometry and Mapping," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 4390-4396, doi: 10.1109/IROS51168.2021.9636655.

[18] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 4758-4765, doi: 10.1109/IROS.2018.8594299.

[19] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard and F. Dellaert, "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 3281-3288, doi: 10.1109/ICRA.2011.5979641.

[20] D. Faconti. 2018. *LeGO-LOAM-BOR*, [Source code]. Available: `https://github.com/facontidavide/LeGO-LOAM-BOR`

[21] K. Giseop. 2020. *SC-LeGO-LOAM*, [Source code]. Available: `https://github.com/irapkaist/SC-LeGO-LOAM`

[22] Z. Zhang, “Iterative Closest Point (ICP),” Computer Vision, pp. 433–434, 2014, doi: 10.1007/978-0-387-31439-6_179.

[23] K. Koide, J. Miura, and E. Menegatti, “A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement,” International Journal of Advanced Robotic Systems, vol. 16, no. 2, p. 172988141984153, 2019, doi: 10.1177/1729881419841532.

[24] M. A. Fischler and R. C. Bolles, “Random sample consensus,” Communications of the ACM, vol. 24, no. 6, pp. 381–395, Jun. 1981, doi: 10.1145/358669.358692.

[25] H. Ye, Y. Chen and M. Liu, "Tightly Coupled 3D Lidar Inertial Odometry and Mapping," 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 3144-3150, doi: 10.1109/ICRA.2019.8793511.

[26] K. Giseop. 2021. *SC-LIO-SAM*, [Source code]. Available: `https://github.com/gisbi-kim/SC-LIO-SAM`

[27] M. Jung, S. Jung, H. Jang, and A. Kim, “Intensity and ambient enhanced LIDAR-inertial slam for unstructured construction environment,” Journal of Korea Robotics Society, vol. 16, no. 3, pp. 179–188, 2021, doi: 10.7746/jkros.2021.16.3.179.

[28] G. Wang, S. Gao, H. Ding, H. Zhang, and H. Cai, “LIO-CSI: LIDAR inertial odometry with loop closure combined with Semantic Information,” PLOS ONE, vol. 16, no. 12, 2021, doi: 10.1371/journal.pone.0261053.

[29] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han, “Searching efficient 3D architectures with sparse point-voxel convolution,” Computer Vision – ECCV 2020, pp. 685–702, 2020, doi: 10.1007/978-3-030-58604-1_41.

[30] J. Jiao, H. Ye, Y. Zhu and M. Liu, "Robust Odometry and Mapping for Multi-LiDAR Systems With Online Extrinsic Calibration," in IEEE Transactions on Robotics, vol. 38, no. 1, pp. 351-371, Feb. 2022, doi: 10.1109/TRO.2021.3078287.

[31] T. -M. Nguyen, S. Yuan, M. Cao, L. Yang, T. H. Nguyen and L. Xie, "MILIOM: Tightly Coupled Multi-Input Lidar-Inertia Odometry and Mapping," in IEEE Robotics and Automation Letters, vol. 6, no. 3, pp. 5573-5580, July 2021, doi: 10.1109/LRA.2021.3080633.

[32] Z. Wang, L. Zhang, Y. Shen and Y. Zhou, "D-LIOM: Tightly-coupled Direct LiDAR-Inertial Odometry and Mapping," in IEEE Transactions on Multimedia, doi: 10.1109/TMM.2022.3168423.

[33] Y. Roth-Tabak and R. Jain, “Building an environment model using depth information,” Computer, vol. 22, no. 6, pp. 85–90, 1989, doi: 10.1109/2.30724.

[34] D. M. Cole and P. M. Newman, "Using laser range data for 3D SLAM in outdoor environments," Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., 2006, pp. 1556-1563, doi: 10.1109/ROBOT.2006.1641929.

[35] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6d SLAM-3D mapping outdoor environments," Journal of Field Robotics, vol. 24, no. 8-9, pp. 699–722, 2007, doi: 10.1002/rob.20209.

[36] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, "Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments," Journal of Field Robotics, vol. 34, no. 5, pp. 940–984, 2016, doi: 10.1002/rob.21700.

[37] M. Herbert, C. Caillas, E. Krotkov, I. S. Kweon and T. Kanade, "Terrain mapping for a roving planetary explorer," Proceedings, 1989 International Conference on Robotics and Automation, 1989, pp. 997-1002 vol.2, doi: 10.1109/ROBOT.1989.100111.

[38] R. Hadsell, J. A. Bagnell, D. Huber, and M. Hebert, "Accurate rough terrain estimation with space-carving kernels," Robotics: Science and Systems V, 2009, doi: 10.15607/rss.2009.v.019.

[39] R. Triebel, P. Pfaff and W. Burgard, "Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing," 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 2276-2282, doi: 10.1109/IROS.2006.282632.

[40] D. Meagher, "Geometric modeling using Octree encoding," Computer Graphics and Image Processing, vol. 19, no. 1, p. 85, 1982, doi: 10.1016/0146-664x(82)90128-9.

[41] J. Wilhelms and A. Van Gelder, "Octrees for faster Isosurface Generation," ACM Transactions on Graphics, vol. 11, no. 3, pp. 201–227, 1992, doi: 10.1145/130881.130882.

[42] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping Framework based on octrees," Autonomous Robots, vol. 34, no. 3, pp. 189–206, 2013, doi: 10.1007/s10514-012-9321-0.

[43] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3D-NDT," Journal of Field Robotics, vol. 24, no. 10, pp. 803–827, 2007, doi: 10.1002/rob.20204.

[44] T. Stoyanov, M. Magnusson, H. Andreasson and A. J. Lilienthal, "Path planning in 3D environments using the Normal Distributions Transform," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 3263-3268, doi: 10.1109/IROS.2010.5650789.

[45] S. Garrido, M. Malfaz, and D. Blanco, “Application of the fast marching method for outdoor motion planning in Robotics,” Robotics and Autonomous Systems, vol. 61, no. 2, pp. 106–114, 2013, doi: 10.1016/j.robot.2012.10.012.

[46] R. Bogdan Rusu, A. Sundaresan, B. Morisset, K. Hauser, M. Agrawal, J.-C. Latombe, and M. Beetz, “Leaving flatland: Efficient real-time three-dimensional perception and motion planning,” Journal of Field Robotics, vol. 26, no. 10, pp. 841–862, 2009, doi: 10.1002/rob.20313.

[47] S. Pütz, T. Wiemann, M. K. Piening and J. Hertzberg, "Continuous Shortest Path Vector Field Navigation on 3D Triangular Meshes for Mobile Robots," 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 2256-2263, doi: 10.1109/ICRA48506.2021.9560981.

[48] B. Garigipati, N. Strokina and R. Ghabcheloo, "Evaluation and comparison of eight popular Lidar and Visual SLAM algorithms," 2022 25th International Conference on Information Fusion (FUSION), 2022, pp. 1-8, doi: 10.23919/FUSION49751.2022.9841323.

[49] P. Fankhauser, M. Bloesch and M. Hutter, "Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization," in IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 3019-3026, Oct. 2018, doi: 10.1109/LRA.2018.2849506.

[50] J. P. Snyder, Flattening the earth two thousand years of map projections. Chicago u.a.: The University of Chicago Press, 2002.

[51] C. Forster, L. Carlone, F. Dellaert and D. Scaramuzza, "On-Manifold Preintegration for Real-Time Visual–Inertial Odometry," in IEEE Transactions on Robotics, vol. 33, no. 1, pp. 1-21, Feb. 2017, doi: 10.1109/TRO.2016.2597321.

[52] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo and J. Gonzalez, "Experimental kinematics for wheeled skid-steer mobile robots," 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007, pp. 1222-1227, doi: 10.1109/IROS.2007.4399139.

[53] P. Corke, Robotics, vision and Control: Fundamental Algorithms in MATLAB. Australia: Springer, 2011.

[54] J. L. Martínez, A. Mandow, J. Morales, S. Pedraza, and A. García-Cerezo, “Approximating kinematics for tracked mobile robots,” The International Journal of Robotics Research, vol. 24, no. 10, pp. 867–878, 2005. doi: 10.1177/0278364905058239.

[55] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, no. 1, pp. 269–271, 1959, doi: 10.1007/bf01386390.

[56] D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997, doi: 10.1109/100.580977.

[57] Stanford Artificial Intelligence Laboratory et al. 2018. *Robotic Operating System*, ver.Noetic. [Software]. Available: `https://www.ros.org`

[58] D. Herschberger, D. Gossow, J. Faust, W. Woodall. 2012. *ROS Visualization (RViz)*, [Software]. Available: `http://wiki.ros.org/rviz`

[59] Open Source Robotics Foundation. 2014. *Gazebo Simulator*, ver.11.0.0, [Online]. Available: `https://classic.gazebosim.org/blog/gazebo11`

[60] M. A. Sherman, A. Seth, and S. L. Delp, "Simbody: Multibody Dynamics for Biomedical Research," Procedia IUTAM, vol. 2, pp. 241–261, 2011, doi: 10.1016/j.piutam.2011.04.023.

[61] "Husky UGV - outdoor field research robot by clearpath," Clearpath Robotics, 20-Dec-2022. [Online]. Available: `https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`. [Accessed: 06-Jan-2023].

[62] Clearpath Robotics. 2016. *husky ROS meta-package*, [Source code]. Available: `https://github.com/husky/husky`

[63] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, and E. Fernandez Perdomo, "Ros_control: A generic and simple control framework for Ros," The Journal of Open Source Software, vol. 2, no. 20, p. 456, 2017, doi: 10.21105/joss.00456.

[64] J. Hsu, J. Koenig, D. Coleman. 2013. *gazebo_ros ROS package*, [Source code]. Available: `https://github.com/ros-simulation/gazebo_ros_pkgs/tree/noetic-devel/gazebo_ros`

[65] K. Hallenbeck. 2018. *velodyne_simulator ROS package*, [Source code]. Available: `https://bitbucket.org/DataspeedInc/velodyne_simulator/src/master/`

[66] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 1-4, doi: 10.1109/ICRA.2011.5980567.

[67] Open Perception, J. Kammerl, W. Woodall. 2018. *pcl_ros ROS package*, [Source code]. Available: `https://github.com/ros-perception/perception_pcl`

[68] S. Kohlbrecher, J. Meyer. 2016. *hector_gazebo_plugins ROS package*, [Source code]. Available: `https://github.com/tu-darmstadt-ros-pkg/hector_gazebo/tree/melodic-devel/hector_gazebo_plugins`

[69] F. Dellaert, GTSAM Contributors. 2022. *borglab/gtsam*, ver.4.2a8, [Software]. Georgia Tech Borg Lab, doi: 10.5281/zenodo.5794541.

[70] T. Moore. 2015. *robot_localization ROS package*, [Source code]. Available: `https://github.com/cra-ros-pkg/robot_localization`

[71] P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and use case for Rough Terrain Navigation," Studies in Computational Intelligence, pp. 99–120, 2016. doi: 10.1007/978-3-319-26054-9_5.

[72] B. Marthi. 2014. *occupancy_grid_utils ROS package*, [Source code]. Available: `https://github.com/clearpathrobotics/occupancy_grid_utils`

[73] M. Alban. 2015. *MapViz: Modular ROS visualization tool*, [Source code]. Available: `https://github.com/swri-robotics/mapviz`

[74] E. Marder-Eppstein. 2020. *move_base ROS package*, [Source code]. Available: `https://github.com/ros-planning/navigation/tree/noetic-devel/move_base`

[75] E. Marder-Eppstein. 2020. *nav_core ROS package*, [Source code]. Available: `https://github.com/ros-planning/navigation/tree/noetic-devel/nav_core`

[76] K. Konolige, E. Marder-Eppstein. 2014. *navfn ROS package*, [Source code]. Available: `https://github.com/ros-planning/navigation/tree/noetic-devel/navfn`

[77] E. Marder-Eppstein. 2020. *dwa_local_planner ROS package*, [Source code]. Available: `https://github.com/ros-planning/navigation/tree/noetic-devel/dwa_local_planner`

[78] E. Marder-Eppstein, D. V. Lu, D. Herschberger. 2018. *costmap_2d ROS package*, [Source code]. Available: `https://github.com/ros-planning/navigation/tree/noetic-devel/costmap_2d`

[79] D. Snider. 2017. *ROS Offline Google Maps for MapViz*, [Source code]. Available: `https://github.com/danielsnider/MapViz-Tile-Map-Google-Maps-Satellite`

[80] J. Chris et al. 2007. *Terminator Terminal Emulator*, [Software]. Available: `https://gnome-terminator.org/`

[81] "Puck lidar sensor, high-value surround Lidar," Velodyne Lidar, 02-Dec-2022. [Online]. Available: `https://velodynelidar.com/products/puck/`. [Accessed: 06-Jan-2023].

[82] "Pololu - UM7 Orientation Sensor," Pololu Robotics & Electronics. [Online]. Available: `https://www.pololu.com/product/2741`. [Accessed: 06-Jan-2023].

[83] "Duro product summary - swift nav." [Online]. Available: https://www.swiftnav.com/sites/default/files/duro_product_summary.pdf. [Accessed: 06-Jan-2023].

[84] Blender Foundation. 2002. *Blender*, ver.2.82.7., [Software]. Available: https://www.blender.org/

[85] O. Hellwich, "Photogrammetric methods," Encyclopedia of GIS, pp. 860–864, 2008.

[86] C. Wethington, J. Knapp, J. Puckette, C. Barnes, L. Weilert, and D. Laó-Dávila, "The Skyline Drive Road Area," OpenTopography Dataspace - survey of Skyline Drive area, Cañon City, Colorado, April 2021, 24-Jun-2021. [Online]. Available: https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.062021.32613.1. [Accessed: 07-Jan-2023].

[87] CloudCompare. 2003. *CloudCompare: 3D point cloud and mesh processing software*, ver.2.11.1, [Software]. Available:https://www.danielgm.net/cc/

[88] P. Cignoni, A. Muntoni. 2022. *MeshLab: software for processing and editing 3D triangular meshes*, ver.2022.02, [Software]. Available:https://www.meshlab.net/

[89] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," ACM Transactions on Graphics, vol. 32, no. 3, pp. 1–13, 2013, doi: 10.1145/2487228.2487237.

[90] "Google Earth: Skyline Drive Road Area," Google Earth. [Online]. Available: https://www.google.com/intl/cs/earth/. [Accessed: 07-Jan-2023].

[91] The GIMP Team. 2022. *GIMP: Open source image editor*, ver.2.10, [Software]. Available: https://www.gimp.org/

[92] "Google Maps: Skyline Drive Road Area," Google maps. [Online]. Available: https://www.google.com/maps/. [Accessed: 07-Jan-2023].

Appendix A

SDF model generation with mesh2sdf tool

The purpose of this C++ project is to automatically generate SDF models from textured `.stl`/`.dae` meshes using the TinyXML2[1] library. The generated `model.sdf` and `model.config` files are placed in a created folder structure as illustrated in Figure A.1.

The code assumes that all input files have the same name and that at least the source mesh file without a texture is provided using the `--src` parameter. A list of valid parameters can be found in Table A.1.

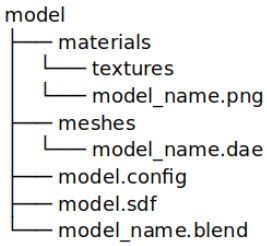


Figure A.1: Folder structure of a generated SDF model.

Parameter	Description	Example
`--help`	display help message	—
`--src`	name of the source model	foo
`--out`	name of the output model	fooo
`--a`	author's name	Vratislav Besta
`--e`	author's email	bestavra@gmail.com
`--mv`	set model version	1.0
`--sv`	set SDF version	1.5
`--d`	model description	Generated with mesh2sdf

Table A.1: The list of valid parameters for the mesh2sdf.

[1] https://github.com/leethomason/tinyxml2

Appendix B

Derivation of the quantile function for CEP (Circular Error Probable)

The CEP measure assumes two orthogonal and independent Gaussian random variables

$$U \sim \mathcal{N}(0, \sigma^2), \qquad V \sim \mathcal{N}(0, \sigma^2)$$

which put together a two-dimensional vector

$$Y = (U, V).$$

With U and V having density functions

$$f_U(u, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{u^2}{2\sigma^2}\right), \qquad f_V(v, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{v^2}{2\sigma^2}\right),$$

and R as the length of Y, i.e. $R = \sqrt{U^2 + V^2}$, the R has cumulative distribution function

$$F_R(r, \sigma) = \iint_{D_r} f_U(u, \sigma) f_V(v, \sigma) \mathrm{d}A, \tag{B.1}$$

where D_r is the disk

$$D_r = \left\{u, v \mid \sqrt{u^2 + v^2} \leq r\right\}.$$

Rewriting the (B.1) in polar coordinates

$$u = \rho\cos\theta, \quad v = \rho\sin\theta$$

with the Jacobian

$$|\boldsymbol{J}| = \begin{vmatrix} \frac{\partial u}{\partial \theta} & \frac{\partial v}{\partial \theta} \\ \frac{\partial u}{\partial \rho} & \frac{\partial v}{\partial \rho} \end{vmatrix} = \begin{vmatrix} \cos\theta & \sin\theta \\ -\rho\sin\theta & \rho\cos\theta \end{vmatrix} = \rho\cos^2\theta + \rho\sin^2\theta = \rho$$

yields the following:

$$F_R(r, \sigma) = \frac{1}{2\pi\sigma^2} \int_0^{2\pi} \int_0^r \rho \exp\left(\frac{-\rho^2}{2\sigma^2}\right) \mathrm{d}\rho\mathrm{d}\theta. \tag{B.2}$$

Integrating (B.2) with respect to θ, it becomes

$$F_R(r,\sigma) = \frac{1}{\sigma^2}\int_0^{\rho} \rho \exp\left(\frac{-\rho^2}{2\sigma^2}\right) \mathrm{d}\rho. \tag{B.3}$$

Further integration with respect to ρ

$$F_R(r,\sigma) = \frac{1}{\sigma^2}\int_0^{\rho} \rho \exp\left(\frac{-\rho^2}{2\sigma^2}\right) \mathrm{d}\rho \; = \left| \begin{matrix} \text{subst.} \\ s = -\frac{\rho^2}{2\sigma^2} & \mathrm{d}u = -\frac{\rho}{\sigma^2}\mathrm{d}r \\ l.b.\ u = -\frac{0^2}{2\sigma^2} = 0 & u.b.\ u = -\frac{r^2}{2\sigma^2} \end{matrix} \right|$$

$$= \int_0^{-\frac{r^2}{2\sigma^2}} \exp^s \mathrm{d}s$$

yields cumulative distribution function of the Rayleigh distribution:

$$F_R(r,\sigma) = 1 - \exp\left(-\frac{r^2}{2\sigma^2}\right). \tag{B.4}$$

From substitution of (B.4) into quantile function

$$Q_R(p) = F_R^{-1}(r),$$

the $Q_R(p)$ becomes

$$Q_R(p,\sigma) = \sigma\sqrt{-2\ln(1-p)}, \tag{B.5}$$

where p is the probability. Since CEP specifies a radius of the mean-centered circle that includes 50 % of all measurements, the standard deviation σ can be estimated using the following equation:

$$Q_R(p,\sigma) = \sigma\sqrt{-2\ln(0.5)}. \tag{B.6}$$

Appendix C

Navigation experiments - video

The following video (https://www.youtube.com/watch?v=gA9GhKZYBec) presents highlights from a series of navigation experiments. Each section of the video corresponds to one of the experiments and shows visualizations of the navigation data in RViz and the running simulation in Gazebo. In some cases, MapViz is also used to set goals.

In RViz, matched scans from the LIO-SAM are shown in a height-colored format. Traversable cells are represented in grayscale, while obstacles are marked in pink and surrounded by light and dark blue cells indicating the inflation around them. The planned path is shown in violet, and the robot's trajectory is shown in light blue.

Appendix D

Attachments

The attachments include the following files:

1. `gazebo_world.zip`
2. `mapviz.zip`
3. `occupancy_grid_utils.zip`
4. `rough_terrain_navigation.zip`
5. `utils.zip`

The first four attachments contain the contents of ROS packages. The files within these packages that have the added `modified_` prefix are components of the existing software that is cited in the thesis. These files have been expanded with additional content or used with different values for the corresponding parameters. The fifth attachment comprises created Terminator layout, mesh2sdf tool, and a Python script used for interpolating GPS coordinates.

In addition, well-documented private Git repositories have been created to provide a more complete record of the work done and to allow for later reference or reuse. These repositories include additional material such as terrains used in the experiments.

Both the GitHub and GitLab repositories contain the same content and are available upon request from the official GitHub website[1] or the faculty's GitLab[2].

[1] https: //github.com/
[2] https: //gitlab.fel.cvut.cz/users/sign_in