Politecnico di Torino

Master of Science in Mechatronic Engineering

Master's Degree Thesis

Localization and Mapping for Legged Robots

Supervisor:
Prof. Giovanni Gerardo Muscolo

Co-supervisor:
Dr. Geoff Fink

Co-supervisor:
Dr. Claudio Semini

Student:
Giovanni Rosato
ID. 254947

Accademic Year
2019/2020

Abstract

This thesis has been conducted at Ististuto Italiano di Tecnologia (IIT). IIT is an Italian scientific research centre based in Genoa (Italy, EU). In the work, we present an experimental analysis of two methods in the Simultaneous Localization And Mapping (SLAM) literature and two autonomous Localization methods. The SLAM and localization methods have been simulated and tested on a legged robot. Legged robots have a more complicated design with respect to wheeled or tracked vehicles. They are used for important challenges, such as dynamic locomotion over rough terrain, extremely rough surfaces and structural robustness to falls. Legged robots with the capability to recognize the surrounding environment, rather than wheeled vehicles, are able to avoid obstacles by adjusting their height. They are conceived for reaching high versatility and mobility.

We performed simulations as well real experiments using the quadruped robot HyQReal. It is approximately 0.9 m tall, weighs 130 kg, and has onboard hydraulics and battery. In addition to the default sensors, such as an IMU and cameras, our robot is equipped with a Velodyne VLP-16 LiDAR. Light Detection And Ranging (LiDAR) sensors work without any artificial or natural light because they send and receive their own laser impulses and does not require ambient light to operate. The 3D LiDAR is installed with an angle of 15° on the y-axis on the back of the robot providing a full 360 degrees view. We provided an insight study of solutions available in Robotic Operating System (ROS) in the context of autonomous exploration of unknown indoor and outdoor environments on a legged robot. We are interested in verifying which SLAM packages works in the best way out-of-the box for such system.

We discussed about the development of mapping and localization processes in 3D with the algorithm Lightweight and Ground-Optimized Lidar Odometry and Mapping (LeGO-LOAM). We examined about 2D mapping processes with methods such as Gmapping and Hector SLAM. In addition, the Monte Carlo localization (MCL) algorithm was implemented. We tested two versions of the MCL: the first one, works with a 2D pre built map and it produces a position estimate in three dimensions (x, y, yaw). The second one uses a 3D pre built map and it generates a position estimate in six dimensions (x, y, z, roll, pitch, yaw). This algorithms allow to obtain an estimated position as precise as the generated map is. We present simulated as well as real-world experiments with the quadruped robot and thoroughly evaluate the best approach while the robot moves.

We adopted a metric for the accuracy analysis. For each algorithm we took into account the: mapping accuracy, path accuracy and computational load.

We tested them in a simulation environment. We divided the map in three sections with different sizes. For each section we looked for the accuracy parameters in the mapping and localization processes. The simulations show that LeGO-LOAM requires an high computational load, higher then Hector SLAM. In the experimental section, the algorithms on the HyQReal robot are verified. The experiments were carried out in two different environments: the first was a laboratory, which is an indoor room, instead the second one was larger and partly outdoors. For the first experiment, we had a motion capture system which allowed us to obtain a real time position ground truth. We created a 3D model of the laboratory starting from the floor plans. This information was used to compare the accuracy of the map and the estimated position. During both experiments, we had an interference with the robot's safety structure, which prevented it from accidentally falling. To overcome this problem, we implemented a 2D filter named *LaserScan filter* and, a 3D filter named *CropBox filter*, both of them are part of the pointcloud library. For the second experiment the robot travelled more than 100 m with the aim of verifying the accuracy of the algorithms over long distances. We made the robot go back and forth 10 times. We didn't have a motion capture system, hence we calculated the drift in body position for both the real and the estimated ones on the basis of some markers placed on the ground. Quantity and qualitative results demonstrate that a SLAM algorithm requires efficiency with respect to runtime and memory usage. Moreover, to have a versatile and fast legged robot, it has to estimate a position and to build a map as accurate as possible.

Contents

List of Figures

List of Tables

Dedicated to my family, my parents Domenico and Annalisa, my sisters Irma and Fracesca, my friends and my love Gabriella...

Acknowledgements

I would like to express my gratitude to my primary supervisor, Prof. Giovanni Gerardo Muscolo, who allowed me to get in touch with the Italian Institute of Technology. He followed me step by step along the process and was very helpful and professional. I would also like to show my deep appreciation to my supervisor, Dr. Geoff Fink, who guided me throughout this project. Thanks to him I was able to develop professionalism and responsibility in the workplace. His help and knowledge were of fundamental importance to me. Last but not least, Dr Claudio Semini, PI of the Dynamic Legged System Lab, who gave me the opportunity to become part of the DLS family. The months spent in the laboratory are for me a priceless. The international team and professional workplace allowed me to grow and to motivate myself during the project. I would also like to thank my friends and family who supported me and offered deep insight into the study. I wish to acknowledge the help provided by the technical and support staff in the DLS lab.

"We can live a wonderful life in the world if we know how to work and love, work for those we love and love what we work for."."

Lev Tolstoj

Introduction

The Company

This thesis has been conducted at Ististuto Italiano di Tecnologia (IIT). IIT is an italian scientific research centre based in Genoa (Italy,EU) with eleven centers throughout Italy and two outstations in the USA in collaboration with MIT and Harvard. IIT as a company focuses heavily on technology areas such robotics, drug discovery, neuroscience, nanotechnology, computer vision, optical microscopy. The thesis work was conducted at IIT's Dynamic Legged System Lab on the quadruped robot HyQReal.

Background

Mobile robotics is the branch of robotics focused on mobile robots and the methods of achieving navigation and interaction tasks with the environment. Mobile robotics poses different problems than industrial robotics, focused on manipulators. For example, the scene framed by a sensor mounted on a mobile base tends to change suddenly with the movements of the robot. Instead, an industrial robot mounted on a fixed base, can move its joints, and has a limited workspace. In mobile robotics, however, workspace is virtually infinite and, therefore, the space of configurations is introduced, defined as the number of parameters needed to express a robot pose. It follows that the perception of the environment is a strong element compared to industrial robotics.

The traditional sensors used in mobile robotics are typically proximity ones, such as sonar, laser scanner and bumper. Generally, these sensors are active and continuously measure the distance between the sensor and the first obstacle encountered along the path of propagation of the signal itself, measuring the flight time, i.e. the time it takes the signal to go from the laser to the object and go back after being reflected.

One environment sensing technology, commonly used for obstacle detection, is Light Detection and Ranging (LIDAR).

By moving the ranging sensor during the scan a point cloud representing the environment is generated. LIDAR sensors work well also during the night because a LIDAR unit sends out and receives its own laser impulses and does not require ambient light to operate. The actual calculation for measuring how far a returning light photon has travelled to and from an object is quite simple:

$$\text{Distance} = (\text{Speed of light x Time of flight})/2$$

Although its resolution is not considered dense it allows us to get consistent results. In Fig. 1 we can see a legged robot in an unknown environment, the pointcloud shows different colours based on the distance from LIDAR. These point clouds can be used for 3D map construction and motion estimation in order to solve the Simultaneous Localization and Mapping (SLAM) problem.

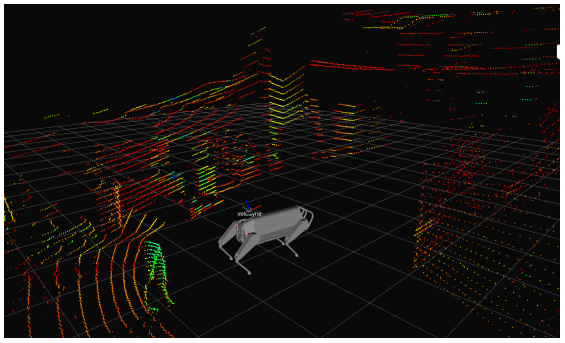


FIGURE 1: Pointcloud aroud the robot, the Velodyne is on the back

A dilemma in robotics is that in order to construct a map, accurate robot position estimation is necessary, but to know the robot's position a map is required. SLAM is the process of updating a map of an unknown environment while keeping track of the position of the vehicle. SLAM has been applied in self-driving cars, unmanned aerial [2] and underwater vehicles, planetary rovers, and even within humanoid robot [3].

A typical approach in the SLAM problem is to divide it in two tasks: how to create an accurate map and how to estimates the pose (position and orientation) of the robot.

Thesis Scope

In this thesis we analyse two algorithms in the SLAM literature and a localization method. For 2D SLAM we present Hector SLAM, instead for a 3D SLAM we analyse LeGO-LOAM, as 2D and 3D localization method, we analysed the behaviour of the Adaptive Monte Carlo Localization method (AMCL). The mapping and localization

methods have been simulated and tested on a quadruped robot, HyQReal [4], running the Robot Operating System (ROS) [5]. HyQReal is an improved version of HyQ and HyQ2Max. IIT's robots that demonstrated a wide repertoire of indoor/outdoor motions ranging from running and jumping to careful walking over rough terrain.

FIGURE 2: HyQReal

The building of a 3D perception system also gives the basis for future works and other skills to the robot. The choice of the sensor fell into the Velodyne Puck Lite 16 in Fig. **??**.

Only LIDAR based sensors will be examined, other vision sensors such as mono and stereo cameras will not be examined. Which SLAM algorithm to be chosen will be supported by a theoretical investigation.

Fig. 3 illustrates the workflow of this thesis. The first phase is a prestudy, where the basic theory of SLAM algorithms will be studied and ROS/Gazebo/Rviz tutorials will be conducted. This is just to get a good initial understanding of the subject. The next phase of the thesis is to doing a theoretical investigation on the SLAM algorithms chosen for investigation in this thesis. Once the base is done, some ROS implementation is needed to be able to perform simulations. When the simulations are done, the same procedure has been conducted on the real robot in order to observe the results.

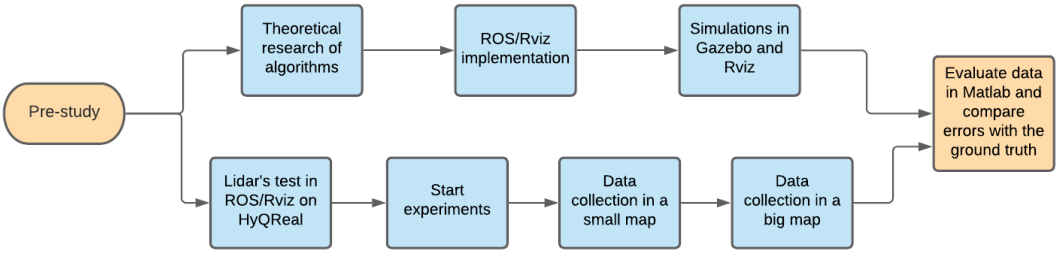


FIGURE 3: Thesis workflow

Outline

The organization of the thesis takes into account the work done and the obtained results, comparing them with similar international researches. The outline is the following:

- In Chapter 1 we will analyse the theoretical works in literature in terms of SLAM works and algorithms that have been developed robotics, general models and the choice of SLAM algorithms.
- In Chapter 2 we discuss about the ROS implementation and how each algorithm has been implemented on the model in Gazebo and Rviz.
- In Chapter 3 we will simulate each algorithm in three scenarios, and we will analize them in term of mapping accuracy, path accuracy and computational effort.
- In Chapter 4 we will discuss about the real experiments. We collected data in two scenarios: a small map in an indoor environment, and a big map in an outdoor environment.
- In Chapter 5 we conclude the thesis with a discussion about all the presented algorithms and results.

Chapter 1

Lidar Odometry, Localization and Mapping

1.1 Introduction to SLAM

In the last few years, with the expansion of several robotics applications, there have been proposed many approaches in autonomous navigation. The SLAM problem is generally divided in two tasks: how to create an accurate map with all its characteristics in 3D and 2D; how to estimates the pose (position and orientation) of the robot. These arguments are treated a lot in the scientific community and many key papers have been proposed.

Methods that are using LIDAR sensors, work well also during the night because a LIDAR unit sends out and receives its own laser impulses and does not require ambient light to operate. Although its resolution is not considered dense it allows us to get consistent results. A generic overview on the SLAM algorithm and its development is contained in [6]. It is based on the work contained in [7], where it is established a high degree of correlation between estimates of the location of different landmarks in a map and how these correlations would grow with successive observations.

We can see that the source of error between estimated and true landmark locations is the error in knowledge of where the robot is when landmark observations are made. As the robot moves through the environment and takes observations of landmarks it builds a network linked by relative location whose precision increases whenever an observation is made.

As a solution method the Extended Kalman filter (EKF) was applied to estimate the pose of the robot, but the performance was not ideal. For some strong non-linear systems, this method will bring more truncation errors, which may lead to inaccurate positioning

and mapping. Particle filter approaches were introduced because they can effectively avoid the non-linear problem, but it also leads to the problem of increasing the amount of calculation with the increase of particle number.

In recent years, based on the works of LiDAR-SLAM, some researchers have started to carry out the research of integrating new methods in order to achieve better results in real-time and accuracy.

The typical approach for finding the transformation between two LIDAR scans is iterative closest point (ICP) [8]. By finding correspondences at a point-wise level, ICP aligns two sets of points iteratively until stopping criteria are satisfied. When the scans include large quantities of points, ICP may suffer from prohibitive computational cost.

1.1.1 General model

Robot process model is a dynamic differential equation to describe the movement of a robot in a given environment and system input. It is related to the robot pose. The robot pose can be determined by its position and orientation.
According to [9]: "In a global coordinate system OXYZ, a robot position (p_v) is expressed by $(x, y, z)^T$, and its orientation can be expressed by Euler angles, rotation matrix, axis and angle, or quaternion. From any one of the orientation representations, it is possible to compute the other representations. For simplicity, Euler angles are selected as a robot orientation state vector. Therefore, the state vector of the robot X_v can be expressed as

$$X_v = \begin{bmatrix} p_v^T \\ \theta_v^T \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \theta_x \\ \theta_y \\ \theta_z \end{bmatrix} \tag{1.1}$$

where T is the transpose of a matrix and assuming that the robot moves relative to its current pose with speed v and changes direction with Euler angles $(\delta\theta_x, \delta\theta_y, \delta\theta_z)$, the input to the robot can be expressed by

$$U = \begin{bmatrix} v \\ \delta\theta_x \\ \delta\theta_y \\ \delta\theta_z \end{bmatrix} \tag{1.2}$$

where v is the robot speed in scalar, and the direction of the speed is always in the robot's forward pointing axis of its body. In order to simplify its implementation, the Euler angles need to be expressed in the form of a rotation matrix M_v

$$M_v = R_z(\theta_z)R_y(\theta_y)R_x(\theta_x) \tag{1.3}$$

where Rz, Ry, and Rx are the rotation matrices which are the rotation around the z, y, x-axis, respectively, in right hand coordinate system with positive angle $\theta_x, \theta_y, \theta_z$, the positive angle is at counter-clockwise direction. Then, the robot process model can be expressed as

$$\theta_u(k+1) = \begin{bmatrix} \theta_x(k+1) \\ \theta_y(k+1) \\ \theta_z(k+1) \end{bmatrix} = \begin{bmatrix} f_1(\theta_x(k), \delta\theta_x, \delta\theta_y, \delta\theta_z) \\ f_2(\theta_y(k), \delta\theta_x, \delta\theta_y, \delta\theta_z) \\ f_3(\theta_z(k), \delta\theta_x, \delta\theta_y, \delta\theta_z) \end{bmatrix} \tag{1.4}$$

and

$$P_u(k+1) = \begin{bmatrix} x(k+1) \\ y(k+1) \\ z(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ z(k) \end{bmatrix} + M_u(k) \begin{bmatrix} \cos(\alpha) \\ \cos(\beta) \\ \cos(\gamma) \end{bmatrix} v\delta t \tag{1.5}$$

where δt t is the sampling time, $M_v(k)$ is the rotation matrix, which corresponds to the Euler angles $(\delta\theta_x, \delta\theta_y, \delta\theta_z)$ at time k. In Equation 1.4, the angle corresponds to the matrix Mv(k + 1), which has following equation

$$M_v(k+1) = M_v(\delta\theta) \cdot M_v(k) \tag{1.6}$$

where $M_v(\delta\theta)$ is a matrix which corresponds to the Euler angle $\delta\theta$, and in Equation 1.5 the α, β, γ are direction angles corresponding to the Euler angles, $\theta_x, \theta_y, \theta_z$,

By combining the Equation 1.4 and 1.5, the process model can be written as a non-linear equation

$$X_u(k+1) = F(X_u(k), U(k) + \mu(k) + \omega(k)) \tag{1.7}$$

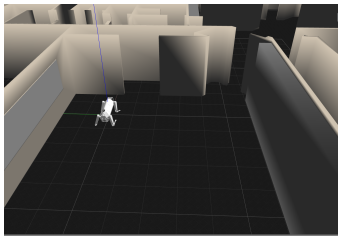
where $\mu(k)$ the input is noise, and $\omega(k)$ is the process noise, at the sample time k. The noise is assumed to be independent for different k, white, and with zero mean and covariance $Q_v(k)$."

1.1.2 EKF SLAM

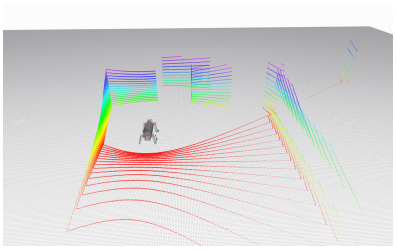
The EKF (Extended Kalman Filter) [10] SLAM approach keeps track of where the robot is likely positioned within a map, as well as keeping track of specific landmarks observed. Hence, when the robot is turned on, the LIDAR will gather information of the positioning of the robot. Moreover, landmarks from the environment are also extracted based on new observations from the LIDAR which is mounted on the robot. These new observations are associated with previous observations and updated in the EKF algorithm. However, if an observation of a landmark cannot be associated to a previous observation the observation itself is presented to the EKF algorithm as a new observation. First and foremost, the robot localizes itself within the map being created using observed landmarks and information from the sensors. The landmarks should preferably be observable from multiple angles, as well as not being separated from other landmarks. These prerequisites gives the EKF algorithm a good possibility to distinguish between landmarks at a later time. Moreover, the landmarks being used should also be stationary.

1.1.3 ICP

Iterative Closest point(ICP) [8] is an algorithm, which minimizes the difference between two point clouds by iteratively finding correspondences between the two sets of points. In the algorithm, one cloud, the Target, is fixed while the other cloud, the Source, is transformed. In each iteration the closest neighbour of each point in the source is found by using a search algorithm. The entire target point cloud is then transformed using the rigid body transformation estimation and a new closest neighbour search is performed. This process is iterated until convergence, thus the name Iterative Closest Point.



(A) HyQReal in Gazebo



(B) Generated pointcloud

FIGURE 1.1: Fig. 1.1a shows HyQReal in Gazebo. Fig. 1.1b shows the generated pointcloud in Rviz

1.1.4 3D LIDAR data representation

A first basic data structure, it's a point cloud intended as a vector of points, characterized by three values indicating position (x, y, z) and three values indicating, where present, the colour of the pixel (r, g, b), see Fig. 1.1b. One of the first data structure introduced specifically to represent three-dimensional data was the voxel grid [11], a grid of cubic cells of equal size that allows to discretize the acquired image, reducing the amount of stored data depending on the dimension of the cells (resolution). However, this is not enough, as a low resolution (large cells) does not allow to faithfully represent the scene, while a high resolution (small cells) increases exponentially the memory occupation, especially in the three-dimensional. In fact, the grid should still be initialized to the size of the bounding box which encloses the whole area to be acquired.

1.1.5 Octomap

One implementation of an approach to storage is Octomap [12], it is based on octree, hierarchical data structures in which each node represents the space contained in a cubic cell, named voxel [11], but with the addition of the tree hierarchy.

Due to the tree structure, octrees are well suited to model Boolean properties such as the occupation of a voxel. If a voxel is found to be occupied, the corresponding node of the tree is initialized with a precise value; if instead it is detected as free, the corresponding node is initialized to the other boolean value. Uninitialized nodes represent the unknown space. The main advantage of a tree structure concerns the possibility of cutting the nodes whose children are all occupied or all free, for reduce the amount of data to be stored.

1.1.6 2D LIDAR data representation

The data representation greatly influences the management of the robot's navigation behaviour. In this sense, in the three-dimensional world, the literature provides various types of data structures for information management perception deriving from sensors such as laser scanners. In the planar world, the most used data structure is the 2D gridmap, that is a two-dimensional matrix in which each cell contains a value and it represents the free, occupied or unknown state of that particular position [13].

1.2 Choice of SLAM algorithm

1.2.1 RBPF

For 2D SLAM approaches a milestone of LIDAR method is Rao-Blackwellized particle filter (RBPF) which is implemented in the ROS Gmapping package [14]. EKF and other Kalman filter based algorithms are efficient for representing linearised distribution, but the RBPF is a better way to represent non-gaussian distribution, because in real world measurements are not linearly distributed, which is suitable for presented test environments. Basic principle of RBPF is to set state of hypotheses, where each particle keeps state, with measurements obtained by laser sensor. Each landmark is associated to the corresponding particle. With given weights the strongest hypotheses is kept, after resampling the weak one is been omitted. Each state represents the posterior and each particle can redefined as a potential pose of the robot.

1.2.2 Hector Mapping

As an effective alternative to Gmapping, Hector SLAM [15] makes use of high scanning rates of rangefinders, which heavily rely on consecutive scan matching of sensor data and combines with multi-resolution occupancy grid maps. Those two algorithms differ from the information sources used for localization and mapping: GMapping uses odometry and laser scan; Hector Slam, instead, uses the laser scan only. Theoretically GMapping should perform better than Hector Slam especially on environments that cause laser scan estimated pose to be ambiguous (large space or long hallway without features): in those scenario GMapping can rely on odometry for robot localization. By other hand Hector Slam does not require odometry (so its a forced choice if robot does not provide it); another big advantage is that Hector Slam can work with laser mounted not planar to ground (as required by GMapping).

However, these solutions work best in planar environments, rely on available, sufficiently accurate odometry and do not leverage the high update rate provided by modern LIDAR systems. An overview of the proposed framework is shown in Fig. 1.2.

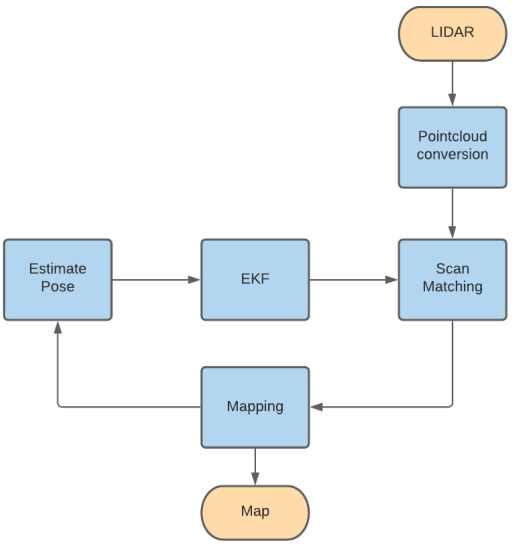


FIGURE 1.2: Overview on the pointcloud conversion and mapping.

1.2.3 LOAM

LIDAR Odometry and Mapping in Real-time (LOAM) is an algorithm for odometry and mapping using range measurements from a 2-axis LIDAR moving in 6-DOF. It achieves both low-drift and low-computational complexity without the need for high accuracy ranging or inertial measurements [16], [17]. The method decomposes the problem by two algorithms running in parallel. An odometry algorithm estimates velocity of the LIDAR and corrects distortion in the point cloud, then, a mapping algorithm matches and registers the point cloud to create a map. Combination of the two algorithms ensures feasibility of the problem to be solved in real-time. Additionally, a two-step method is proposed to remove the distortion [18]: an ICP based velocity estimation step is followed by a distortion compensation step, using the computed velocity.

1.2.4 LeGO-LOAM

The Lightweight and Ground-Optimized LIDAR Odometry and Mapping (LeGO-LOAM) method for real-time 6-DOF pose estimation leverages the presence of ground plane in its segmentation and optimization steps [1]. Real-time performance is achieved by novelly dividing the estimation problem across two individual algorithms. One algorithm runs

at high frequency and estimates sensor velocity at low accuracy. The other algorithm runs at low frequency but returns high accuracy motion estimation. The two estimates are fused together to produce a single motion estimate at both high frequency and high accuracy.

The system receives input from a 3D LIDAR and outputs 6 DOF pose estimation. The overall system is divided into five modules. The details of these modules are introduced in Fig. 1.3.

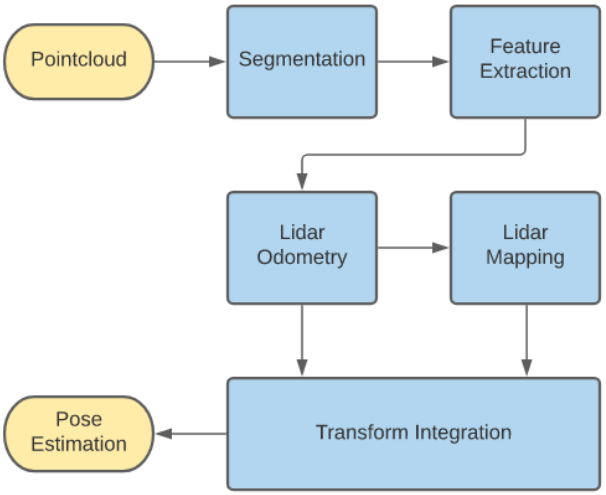


FIGURE 1.3: LeGO-LOAM's system overview

The first, segmentation, takes a single scan's point cloud and projects it onto a range image for segmentation. The segmented point cloud is then sent to the feature extraction module. Then, LIDAR odometry uses features extracted from the previous module to find the transformation relating consecutive scans. The features are further processed in LIDAR mapping, which registers them to global point cloud map. At last, the transform integration module fuses the pose estimation results from LIDAR odometry and LIDAR mapping and outputs the final pose estimate. The proposed system seeks improved efficiency and accuracy for ground vehicles, with respect to the original, generalized LOAM framework of [16] and [17].

We can further eliminate drift for this module by performing loop closure detection. In this case, new constraints are added if a match is found between the current feature set and a previous feature set using ICP. The estimated pose of the sensor is then updated by sending the pose graph to an optimization system such as iSAM2 [19].

The map selection technique is similar to the method used in [16]. However, a full 3D occupancy grid map is necessary since the map needs to encode both occupied and free volumes. In our system, we employ the octree-based mapping framework that models

occupied as well as free and unknown areas in the environment in a probabilistic and memory-efficient way. This enables our robot to use map resolutions as small as 2 cm for a complete 3D indoor map. The map representation is available as an open-source library [12].

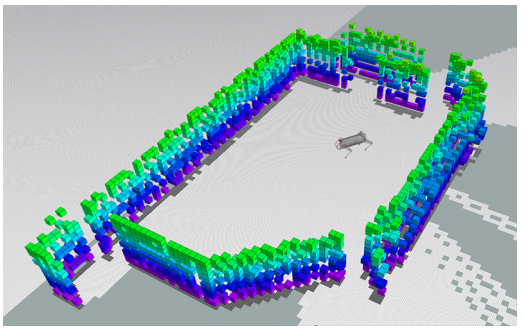


FIGURE 1.4: Example of the constructed map made by LeGO-LOAM

In Fig. 1.4 there is the estimated pointcloud elaborated with Octomap in an occupancy 3D grid map. Octomap also has the functionality to down-project the 3D map on the ground in an occupancy grid map in two dimensions.

1.3 Monte Carlo Localization

The Monte Carlo localization (MCL) algorithm is a probabilistic localization algorithm applied to a two-dimensional occupation grid map [20]. A particle swarm is used to describe and track the current possible pose of mobile robots in known maps [14]. It can globally estimate the pose with a small amount of computations and a low memory footprint. The estimated pose is very smooth during locomotion and is suitable for navigation control of mobile robots [21]. However, it is always affected by the strong non convexity of the sensor model and complex unstructured features of the environment.

MCL is a Bayes filtering technique which recursively estimates the posterior about the robot's pose x_t at time:

$$p(x_t \mid o_{1:t}, u_{1:t}) = \eta \cdot \overbrace{p(o_t \mid x_t)}^{\text{sensor model}} \cdot \int_{x_{t-1}} \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\text{motion model}} \cdot \underbrace{p(x_{t-1} \mid o_{1:t-1}, u_{1:t-1})}_{\text{motion model}} \, dx_{t-1} \tag{1.8}$$

In the formula, η is a normalization constant resulting from Bayes rule, $u_{1:t}$ denotes the sequence of all motion commands executed by the robot up to time t, and $o_{1:t}$ is the sequence of all observations. The term $p(x_t|x_{t-1}, u_t)$ is called the motion model and denotes the probability that the robot ends up in state x_t given it executes the motion command u_t in state x_{t-1}. The sensor model $p(o_t|x_t)$ denotes the likelihood of obtaining observation o_t given the robot's current pose is x_t.

In MCL, the belief distribution over the robot's current state is approximated by a set of n weighted samples or pose hypotheses $X_t = \{\langle x_t^{(1)}, w_t^{(1)}\rangle, \ldots, \langle x_t^{(n)}, w_t^{(n)}\rangle\}$. Here, each $x_t^{(i)}$ is one pose hypothesis and $w_t^{(i)}$is the corresponding weight, which is proportional to the likelihood that the robot is in the corresponding state. The update of the belief, also called particle filtering, consists of prediction, correction, and resampling.

When a laser sensor is used to locate robot on a 2D grid map, if the robot pose is given, it is very easy to calculate the agreement between the laser beams and the occupied grid. Therefore, the MCL algorithm can be used, which represents the pose of the robot with many particles, as shown in Fig. 1.5. Calculate the weight of the particle according

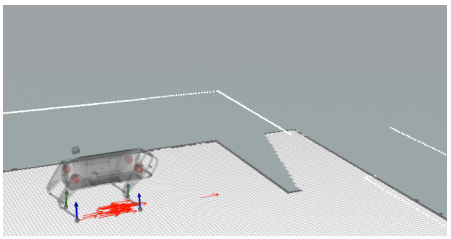


FIGURE 1.5: Particle distribution in the simulation environment

to the agreement with the map; then, determine the estimated pose and locate the robot. However, there are some problems in the MCL algorithm: it cannot solve the robot kidnapping problem. Once the pose changes discontinuous, the localization will fail. To improve the localization accuracy, many particles need to be added and result in slow localization convergence rate. The Adaptive Monte Carlo Localization (AMCL) algorithm is adapted from the MCL algorithm to solve above problems. The AMCL algorithm randomly adds free particles during resampling.

Chapter 2

Ros Configuration

2.1 System Overview

The transform tree is a critical aspect of any ROS implementation and requires much care. It describes how each coordinate frame relates to each other. The tree can have both parent and child elements. Each child can only have one parent frame but each parent can have multiple children. This means that the way a tree can be built is limited. Frames, coordinate systems, in ROS are 3D and right handed (X forward, Y left and Z up). Therefore, if a component publishes its data in a different orientation, it must be adjusted. Ros REP-0105 specifies the conventions that should be taken when building a transform tree. Most frames are published in a chain form, earth to map to odom to baselink. Here, baselink represents a point rigidly attached to the mobile robot base.

HyQReal contains different coordinate frames that change over time. The world frame is the only fixed at $z = 0$ in the simulator. All the frames that are connected to the world are considered mobile frames.

We can see how the tf tree shows the complexity of HyQReal in Fig. A.2 in Appendix A, there is a transformation for each joint, sensor, camera and baselink.

2.1.1 Hector SLAM

Hector SLAM can be used without odometry as well as on platforms that exhibit roll pitch motion. It leverages the high update rate of the LIDAR systems and provides 2D

pose estimates at scan rate of the sensors [22]. To use Hector we need a source of sensor_msg/LaserScan data or bagfiles. The LIDAR provides its own LaserScan but it is angled, and Hector doesn't work properly. This configuration led to the adoption of a package with the aim to convert the PointCloud into LaserScan. Furthermore, with the aid of this package it possible to choose the orientation of the laserscan. The proposed package is a ROS package that provides components to convert sensor_msgs/msg/PointCloud2 messages to sensor_msgs/msg/LaserScan messages and back, see Fig. 2.1.

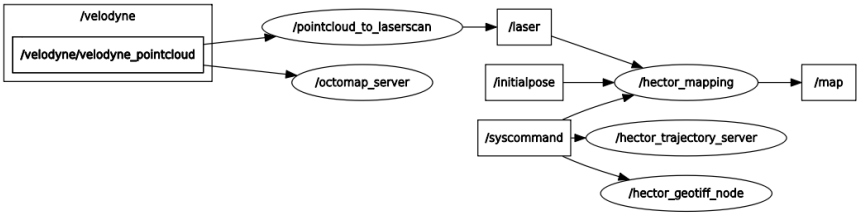
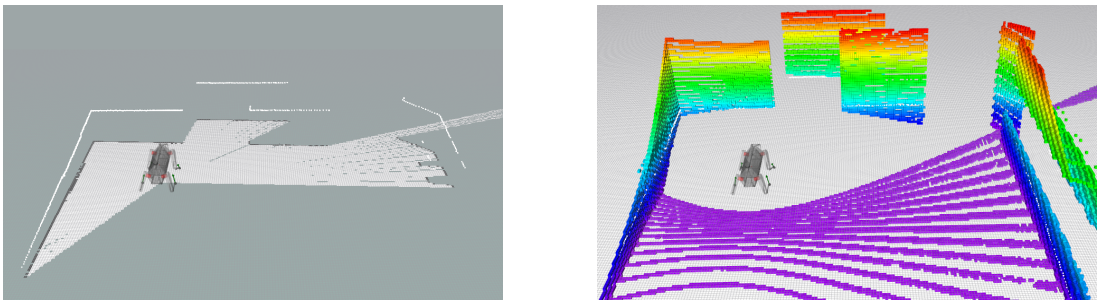


FIGURE 2.1: The ROS computation graph shows the LaserScan converted from the PointCloud and the required Hector SLAM's nodes

As well simulations as experiments Hector SLAM provides a tf transformation between the estimated pose and the real one. We used the */laser* topic as input for Hector SLAM and we also provided a 3D map from Octomap based on the estimated odometry, see Fig. 2.2.

2.1.2 LeGO-LOAM

In Fig. 2.3 we can see the interconnections graph of LeGO-LOAM, the main node that takes the pointcloud as input, and the Octomap as output, the topic */aft_mapped_to_init* is the estimated trajectory and the */laser_cloud_surround* is the pointcloud generated from the algorithm. LeGO-LOAM contains its own prebuilt transform tree. In Fig. 2.4



(A) LaserScan and gridmap

(B) 3D map made by Octomap

FIGURE 2.2: In Fig. 2.2a the white line, parallel to the ground, is the laserscan. The white map on the ground is the gridmap provided by Hector SLAM. In Fig. 2.2b the map built with Octomap using the estimated odometry from Hector SLAM

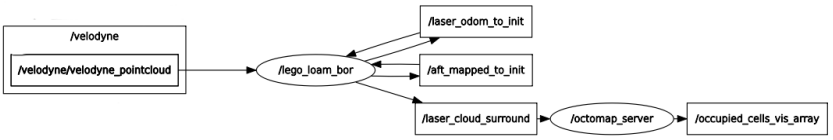


FIGURE 2.3: Hector SLAM's computation graph

LeGO-LOAM calls the LIDAR frame *camera*, thus the initialisation of the LIDAR position is called *camera_init*.

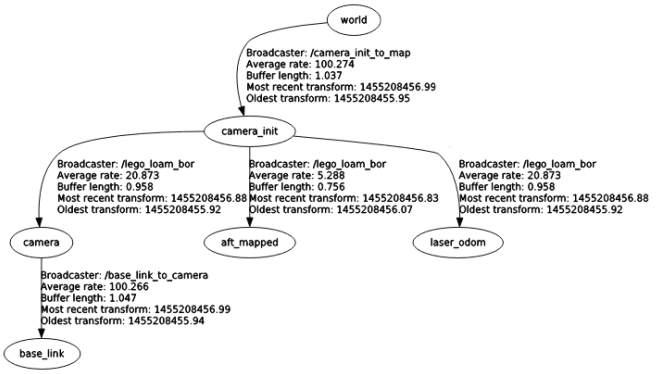


FIGURE 2.4: LeGO-LOAM's computation graph

The bottom most child of the LeGO-LOAM's tree is baselink. This baselink is different from the baselink provided by the tf transformation of the robot but it is computed by LeGO-LOAM and it is fixed to the robot model.

Chapter 3

Simulations

3.1 Overview

In order to determine which algorithm was suitable to use on the robot, simulations were performed on the simulation environment. Three maps were created, each of different size and each map containing different amount of features in the environment. This is to check the performance of each algorithm during different circumstances. All required the topics were recorded for each map, so that an identical path could be used for each simulation. The SLAM algorithms simulated are Hector SLAM for 2D and LeGO-LOAM for the 3D implementation. Thorough descriptions of all these nodes can be found in Chapter 2.

Different types of tests were performed on each algorithm, namely:

- Mapping accuracy
- Path accuracy
- Average center of mass's position error
- Average CPU load

Unlike what we have seen previously, now we testing both of them together. In Fig. 3.1 we can see the interconnection's nodes.

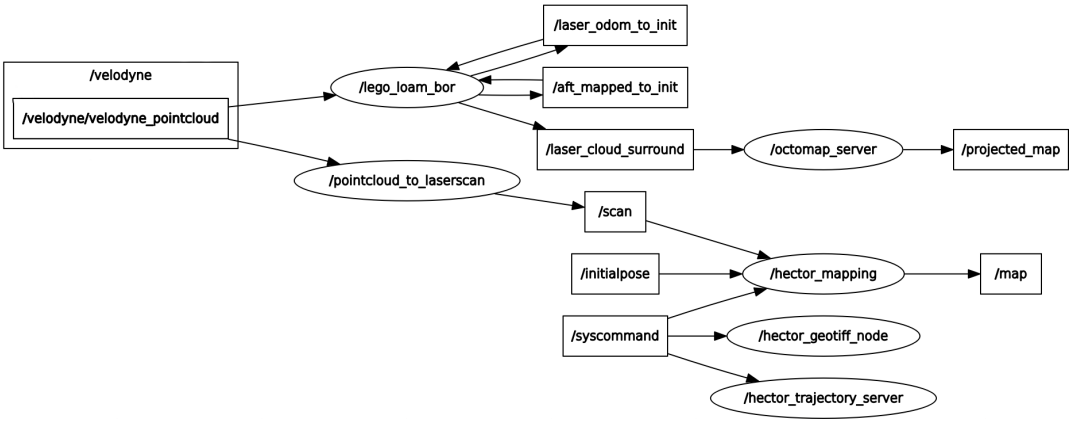
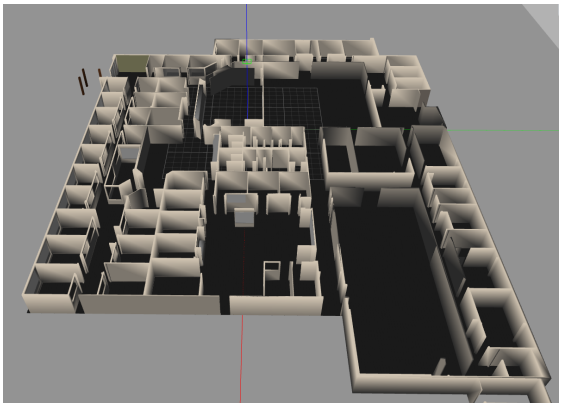
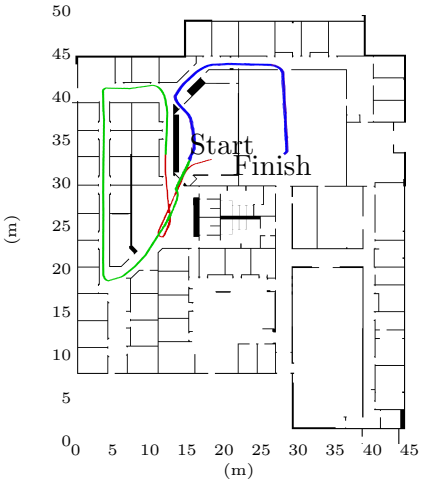


FIGURE 3.1: Graph of the interconnections for Hector SLAM and LeGO-LOAM

Each SLAM algorithm is working on a specific simulated environment, see Fig. 3.2. In Fig. 3.2a we have the model implemented in gazebo, it is the *willowgarage* model. Fig. 3.2b is the downprojection of the map made by a plugin in gazebo named *gazebo model plugin*. From the start position we can see 3 different colours (red, green and blue). They show the travelled path during the small, medium and large simulation, respectively.

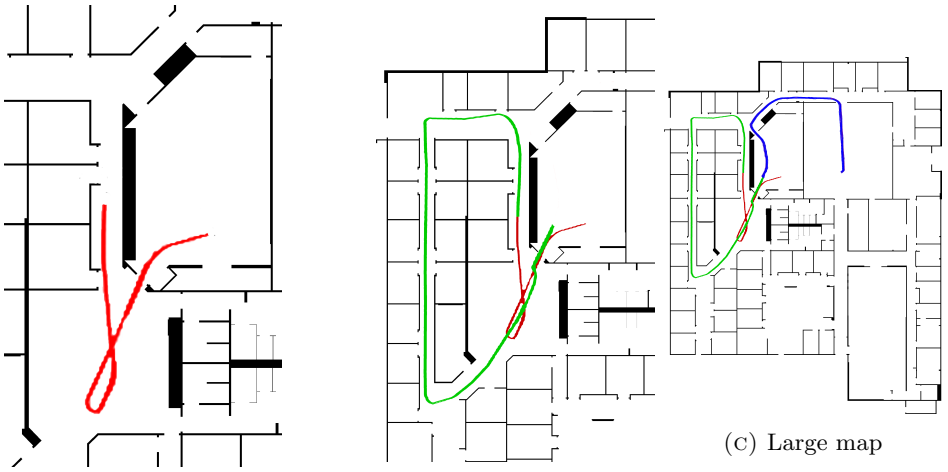


(A) 3D Model mplemented in Gazebo



(B) 2D representation

FIGURE 3.2: Fig. 3.2a is the simulated environment. In Fig. 3.2b we can see the overall path travelled by the robot, from the start to finish.



(A) Small map (B) Medium map (C) Large map

FIGURE 3.3: Each map shows the path travelled by the robot

3.2 Mapping accuracy

Three maps were designed for the simulation. The main idea was to have three maps which grow in size, as all SLAM algorithms scale with the map size, and this might give an idea of how well a particular algorithm scales. The maps are shown in Fig. 3.3. From left to right the map is growing in size.

In the mapping accuracy test the maps are compared in Matlab with specific functions for the OccupancyGridMap both in 2D and 3D. We have prepared tables that show the accuracy for each map.

Furthermore, in experiment we do not know this transformation. Hence we took the estimated trajectory from Hector SLAM and we obtained a 3D map with Octomap. By doing so we used its estimated odometry for building a 3D representation with a 2D SLAM algorithm. We also tested the accuracy of Hector SLAM's map in 2D with the 2D down projected map made by LeGO-LOAM.

In conclusion, we tested them with the self localization methods, to see which map produces the best result.

3.2.1 Small map

Fig. 3.4 below shows the results for the mapping of the small map. The results of each algorithm model appear to be fairly similar.

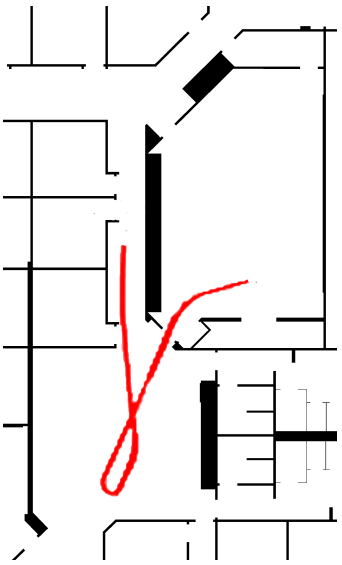
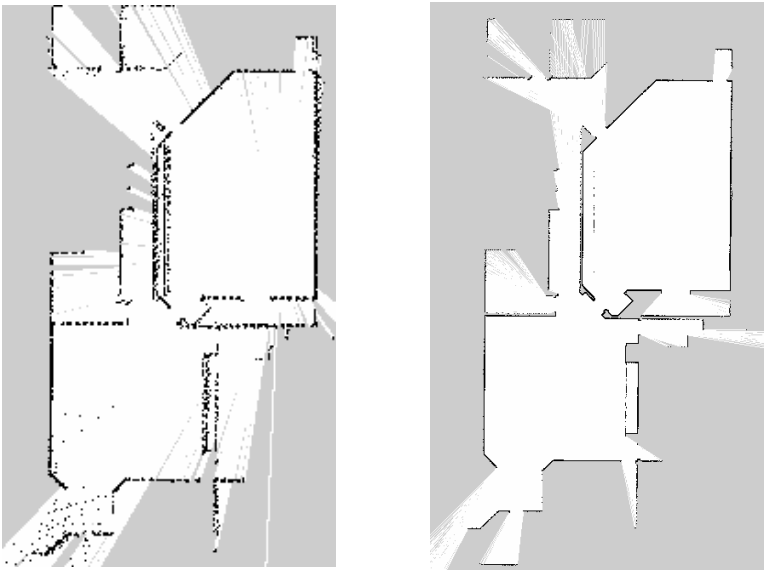


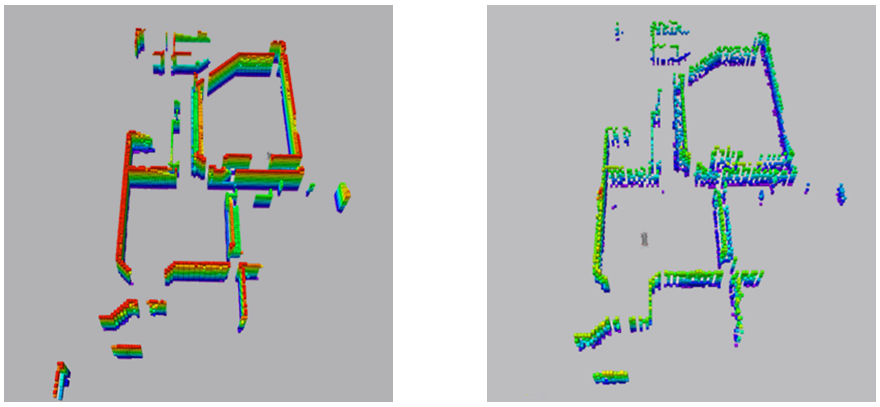
FIGURE 3.4: Small map's ground truth

In Fig. 3.5 we can see the results from the 2D SLAMs. We said before that Octomap has the capability to down project the 3D map, in this way we can compare them also in 2D. We can see in Fig. 3.5a that the generated map from LeGO-LOAM produces a less dense pointcloud than the real one.



(A) LeGO-LOAM (B) Hector SLAM

FIGURE 3.5: 2D maps built by the Hector SLAM and LeGO-LOAM in a small map



(A) Hector SLAM with Octomap (B) LeGO-LOAM

FIGURE 3.6: 3D maps built by Hector SLAM with Octomap and LeGO-LOAM

In Table 3.1 we see how the number of occupied cells of LeGO-LOAM is higher than Hector SLAM. However, LeGO-LOAM produces an high percentage of occupied cells that should be empty. This behaviour could be corrected with a loop closure.

Number of	Hector-SLAM		LeGO-LOAM	
cells that should be occupied	13308			
occupied cells	3278	(24.63%)	5205	(39.11%)
occupied cells that should be occupied	1568	(11.63%)	1740	(13.11%)
occupied cells that should be empty	1324	(9.94%)	2908	(21%)
empty cells that should be occupied	11740	(88.21%)	9089	(68%)
unknown cells that should be occupied	0	0%	2479	(18%)

TABLE 3.1: 2D Accuracy parameters in a small map

Now we are comparing the 3D maps. In this situation we can see the big difference in accuracy between them, see Table 3.2. The 3D pointcloud made by LeGO-LOAM has a lot empty points. Moreover, the Octomap built by Hector SLAM can be considered as a ground truth. It is the best approximation of the environment, it is as good as the estimated pose. In order to say which behaviour has higher performance we have to analyse the path accuracy. From the table we can just see the low density of LeGO-LOAM, it is of 0.2% of the real one, with an error of 2.04%.

Number of	LeGO-LOAM	
voxels that should be occupied	104289	
occupied voxels	3999	3.83%
occupied voxels that should be occupied	276	0.2%
occupied voxels that should be empty	2133	2.04%
empty voxels that should be occupied	14590	13.99%
unknown voxels that should be occupied	89423	85.74%

TABLE 3.2: 3D Accuracy parameters in a small map

3.2.2 Medium map

Starting from Fig. 3.4 we can see that the difference in Fig. 3.7 is that the map contains also a corridor on the left. Here we can see a first difference between them. Due to the fact that Hector SLAM creates the map from a single laserscan at same height of the velodyne, it detects only obstacles at that height. Instead LeGO-LOAM builds the map by downprojecting everything that is detected from the velodyne in three dimensions. This difference increases the accuracy for LeGO-LOAM.

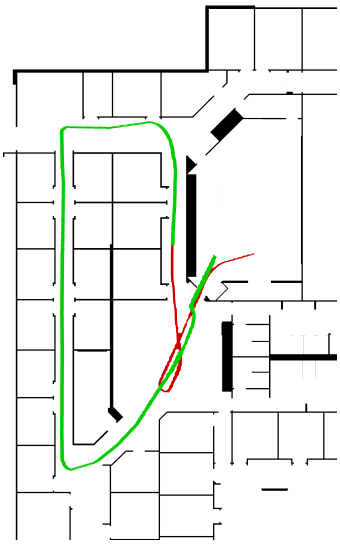


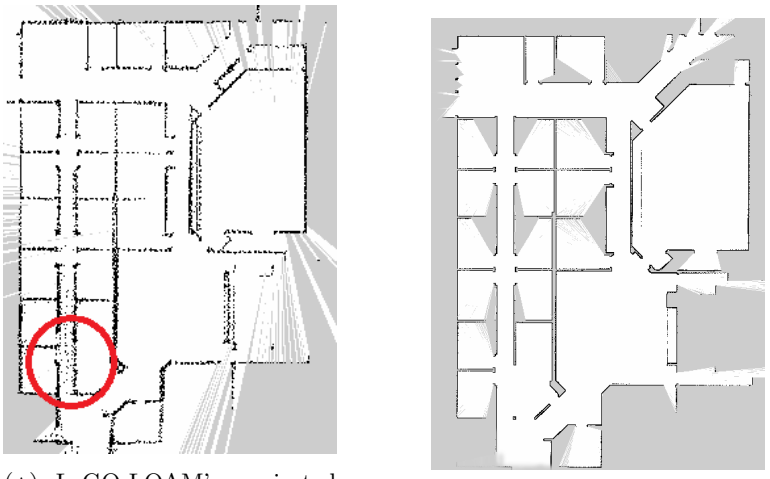
FIGURE 3.7: Medium map's ground truth

In Fig. 3.8a at the bottom side we can see what happens when there is a window and a long corridor during the path. It is marked with a red circle. LeGO-LOAM starts to estimate a slope at the end of the corridor. This wrong estimate produces an high percentage of occupied cells that should be empty, as we can see, it is 26.66%.

Number of	Hector-SLAM		LeGO-LOAM	
cells that should be occupied	22802			
occupied cells	7603	33.34%	13154	57.68%
occupied cells that should be occupied	3415	14.97%	5659	24.81%
occupied cells that should be empty	3795	16.64%	6081	26.66%
empty cells that should be occupied	17365	76.15%	11004	48.25%
unknown cells that should be occupied	2022	8.86%	6139	26.92%

TABLE 3.3: 2D Accuracy parameters in a medium map

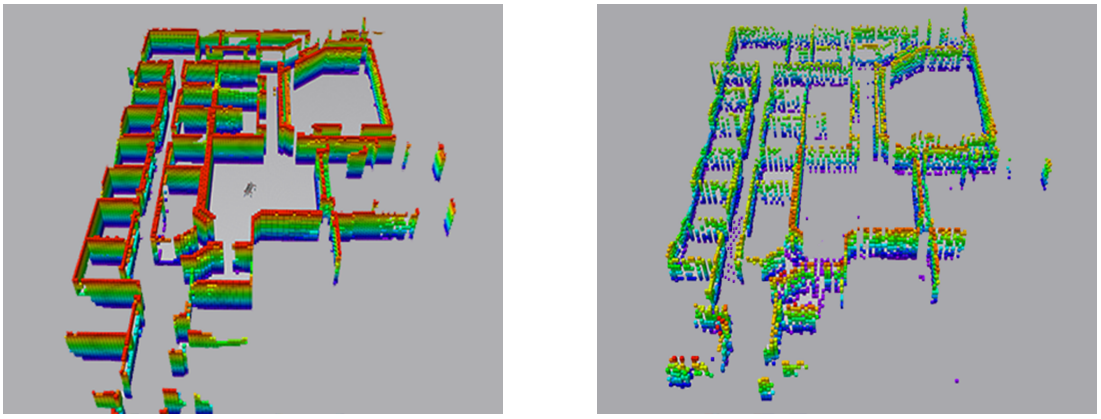
In Fig. 3.9 we can see the produced map in 3D. An important issue can be observed in Fig. 3.9b where the different composition of colors indicates that the estimated odometry of the robot has a big error in the z position. More details are explained in Section 3.4. For the accuracy in Table 3.4, LeGO-LOAM prints some wrong points on the ground, for this reason the occupied voxel's percentage is higher than 100%. As we said, this



(A) LeGO-LOAM's projected map

(B) Hector SLAM

FIGURE 3.8: 2D maps built by the Hector SLAM and LeGO-LOAM in a medium map. LeGO-LOAM detects the window and a slope, marked by a red circle, instead Hector SLAM produces empty cells in that position.



(A) Hector SLAM with Octomap

(B) LeGO-LOAM

FIGURE 3.9: 3D maps built by the Hector SLAM and LeGO-LOAM

behaviour is caused to an error on the pose estimation. For this reason the cells that should be empty are higher than 56%.

Number of	LeGO-LOAM	
voxels that should be occupied	50804	
occupied voxels	59998	118.09%
occupied voxels that should be occupied	6685	13.15%
occupied voxels that should be empty	28668	56.42%
empty voxels that should be occupied	28743	56.57%
unknown voxels that should be occupied	15376	30.26%

TABLE 3.4: 3D Accuracy parameters in a medium map

3.2.3 Large map

The large section test below, contains a big part of the real map. We can observe the 2D ground truth in Fig. 3.10 and the real path made by the robot.

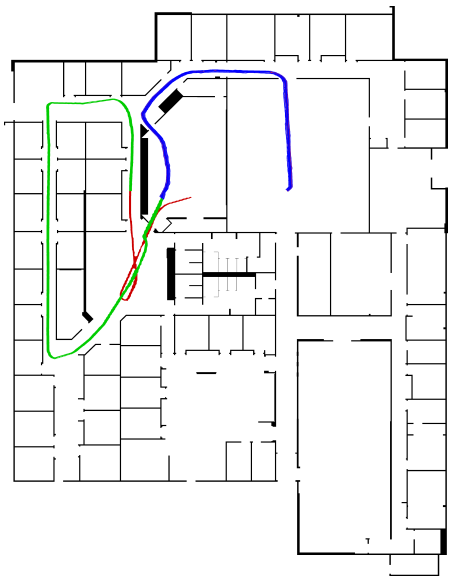
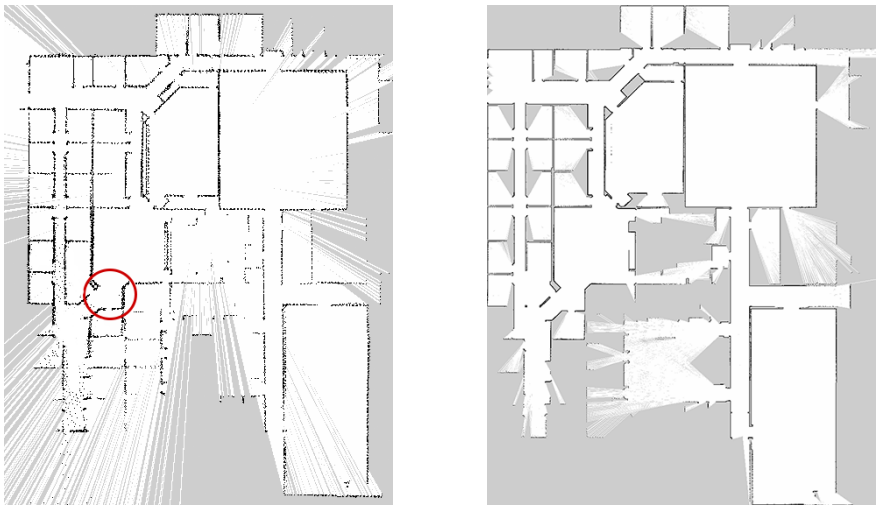


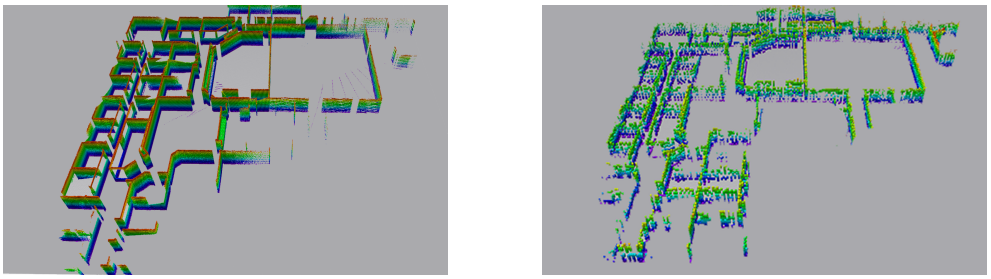
FIGURE 3.10: Large map's ground truth

We can notice a loop closure interaction it means that the algorithm is able to identify a recent room by comparing the actual pointcloud with an history in the memory. By doing so, LeGO-LOAM detects the room that it travelled during the first simulation and it corrects the map and the trajectory, see the circle in Fig. 3.11a. In the room the it travelled during the first simulation it deletes the points on the ground the z-axis estimate is corrected.



(A) LeGO-LOAM's projected map (B) Hector SLAM

FIGURE 3.11: 2D maps built by the Hector SLAM and LeGO-LOAM in a large map



(A) Hector SLAM with Octomap (B) LeGO-LOAM

FIGURE 3.12: 3D maps built by the Hector SLAM and LeGO-LOAM

In Fig. 3.12 are shown the 3D maps as we have seen previously. Table 3.5 depicts the final mapping accuracy between these two algorithms. LeGO-LOAM is still detecting a big amount of feature in the environment rather than Hector SLAM. However, in Table 3.6 we still have a large error in the estimate, as we can see, the occupied voxels that should be empty is 45.36%.

Number of	Hector-SLAM		LeGO-LOAM	
cells that should be occupied	12146			
occupied cells	1707	14.05%	6662	54.84%
occupied cells that should be occupied	389	3.20%	1677	13.80%
occupied cells that should be empty	1170	9.63%	4259	35.06%
empty cells that should be occupied	9376	77.19%	8585	70.68%
unknown cells that should be occupied	2381	19.60%	1884	15.51%

TABLE 3.5: 2D Accuracy parameters in a large map

Fig. 3.12 show the final result in 3D and the Table 3.6 lists the differences in accuracy.

Number of	LeGO-LOAM	
voxels that should be occupied	77731	
occupied voxels	112995	145.36%
occupied voxels that should be occupied	15687	20.18%
occupied voxels that should be empty	35264	45.36%
empty voxels that should be occupied	35698	45.92%
unknown voxels that should be occupied	26346	33.89%

TABLE 3.6: 3D Accuracy parameters in a large map

3.2.4 Conclusion

In conclusion we can say that LeGO-LOAM produces a 2D map with more details rather than Hector SLAM. However, LeGO-LOAM produces a calculation error in the estimation of the trajectory. More details will be explained in the next sections. However, we can conclude that both algorithms estimate loop closures with an accurate probability.

3.3 Path accuracy

The path accuracy test compares the generated path of each algorithm with the actual path that the kinematic robot has travelled. This is presented in a plot which includes the actual path, and the generated path from each algorithm.

The following sections contain the results of the path accuracy test for all three maps. Each figure contains the theoretical path as well as the generated path of Hector SLAM and LeGO-LOAM.

Fig. 3.13 shows all the paths diveded by colors.

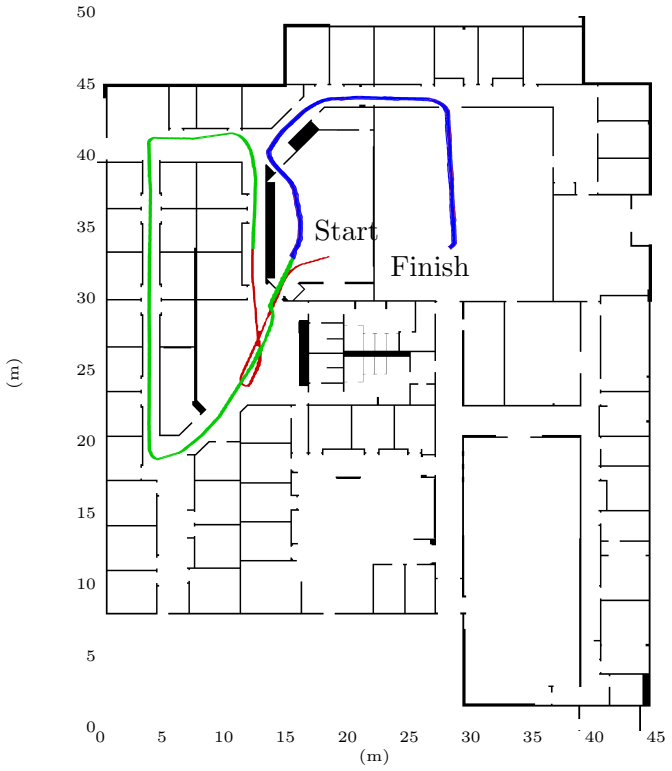


FIGURE 3.13: The red line is the small path, red plus green is the medium one and all together become the large path.

3.3.1 Small map

From now, the colours in the plots are red for the ground truth, green for Hector SLAM and blue for LeGO-LOAM. The following plots show the overall 2D trajectories, see Fig. 3.14. Plots in Fig. 3.15 are taking into account the behaviours (x,y,z) axes and the yaw angles.

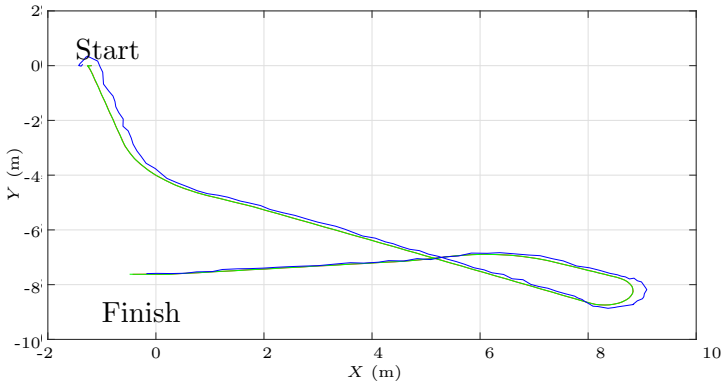


FIGURE 3.14: Trajectories of ground truth, Hector SLAM and LeGO-LOAM in a small map. Red is the ground truth, green is Hector SLAM and blue is LeGO-LOAM

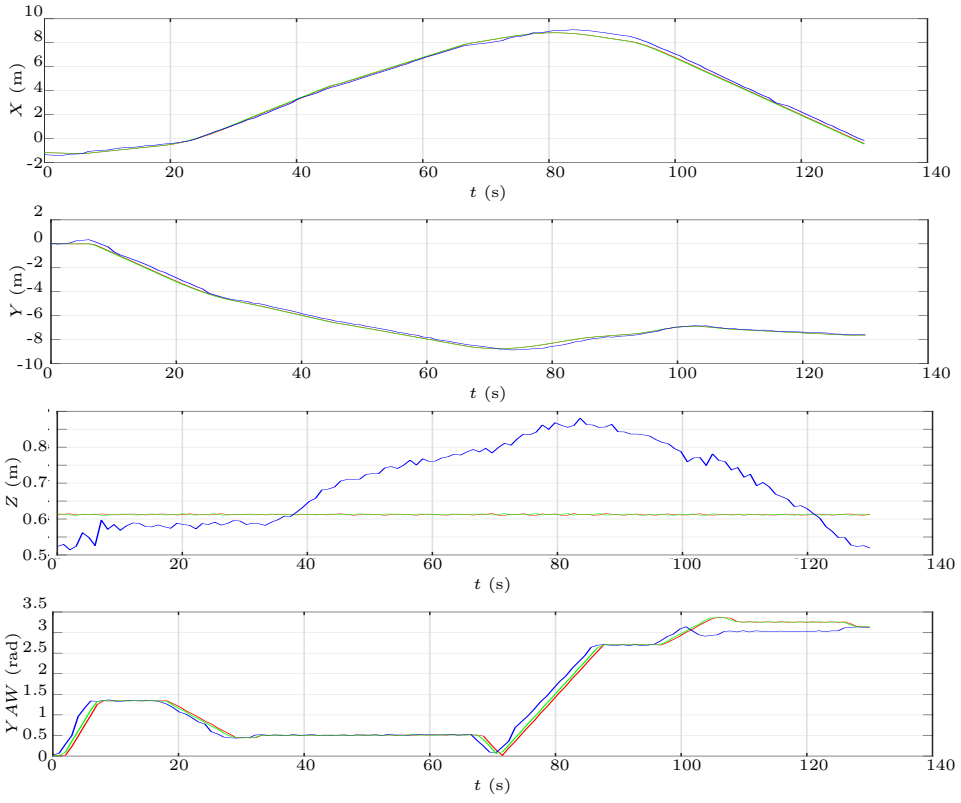


FIGURE 3.15: The plots show the trajectories for each axis and in the last one there are the yaw angles in a small map

3.3.2 Medium map

Fig. 3.16 below shows the generated path of both algorithms compared to the theoretical path when running simulations of the medium map. The generated path of both algorithms does not deviate greatly from the theoretical path. However, LeGO-LOAM shows a significantly erroneous path in the z direction.

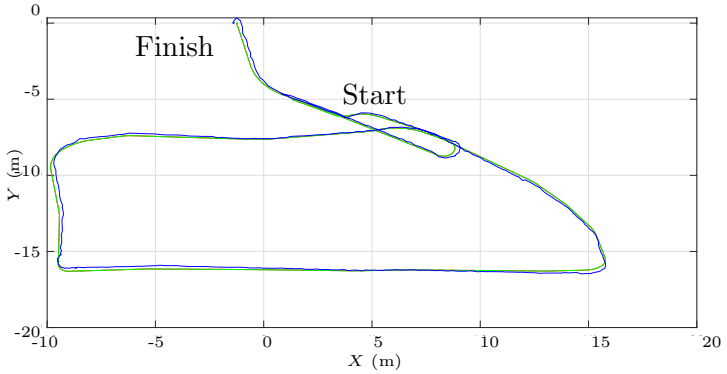


FIGURE 3.16: Trajectories of ground truth, Hector SLAM and LeGO-LOAM in a medium map. Red is the ground truth, green is Hector SLAM and blue is LeGO-LOAM

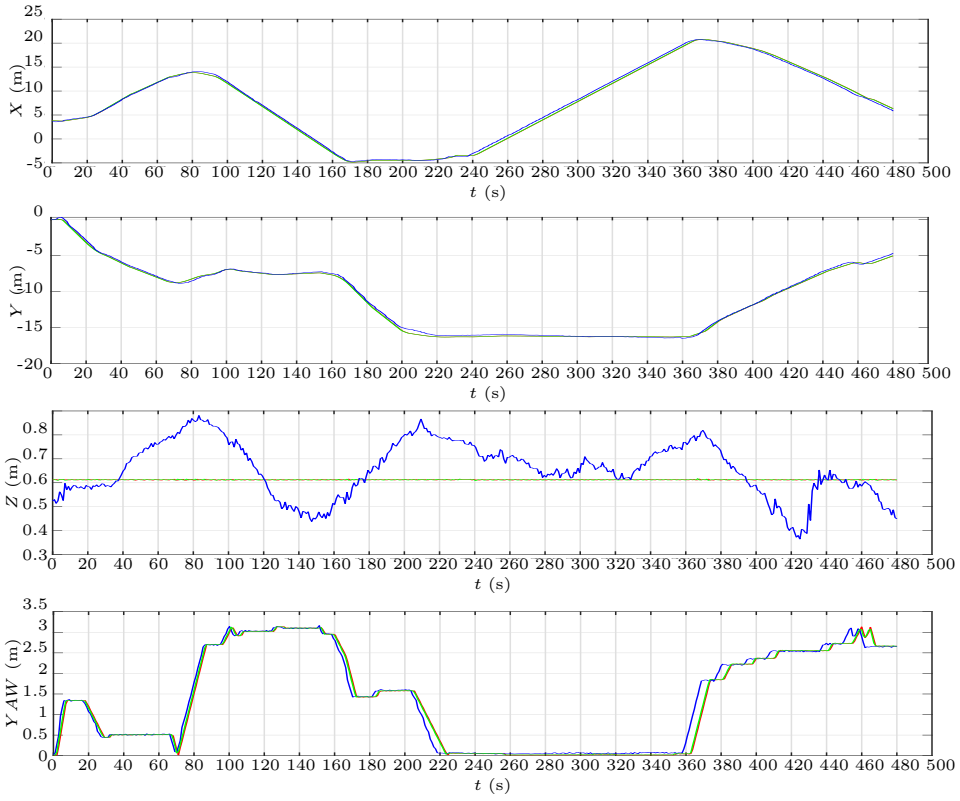


FIGURE 3.17: The plots show the trajectories for each axis and in the last one there are the yaw angles for in a medium map

3.3.3 Large map

In this section we analyse the generated path in a large map. In Fig. 3.19 we can see a good estimation on the x and y axes. The wrong approximation is on the z-axis, at the end of the simulation LeGO-LOAM is considering a large error. However, the yaw angles plot produces a good estimation for both of them.

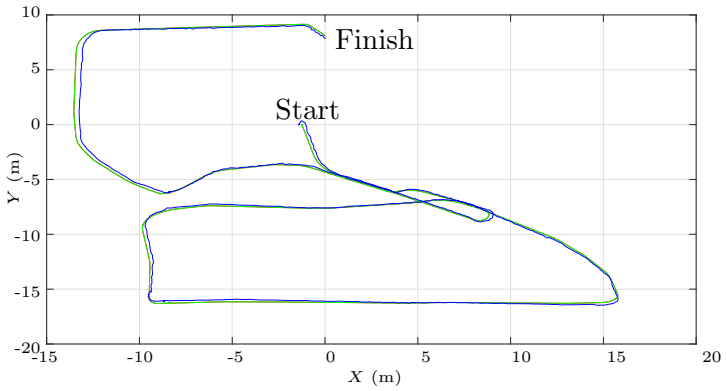


FIGURE 3.18: Trajectories of ground truth, Hector SLAM and LeGO-LOAM in a large map. Red is the ground truth, green is Hector SLAM and blue is LeGO-LOAM

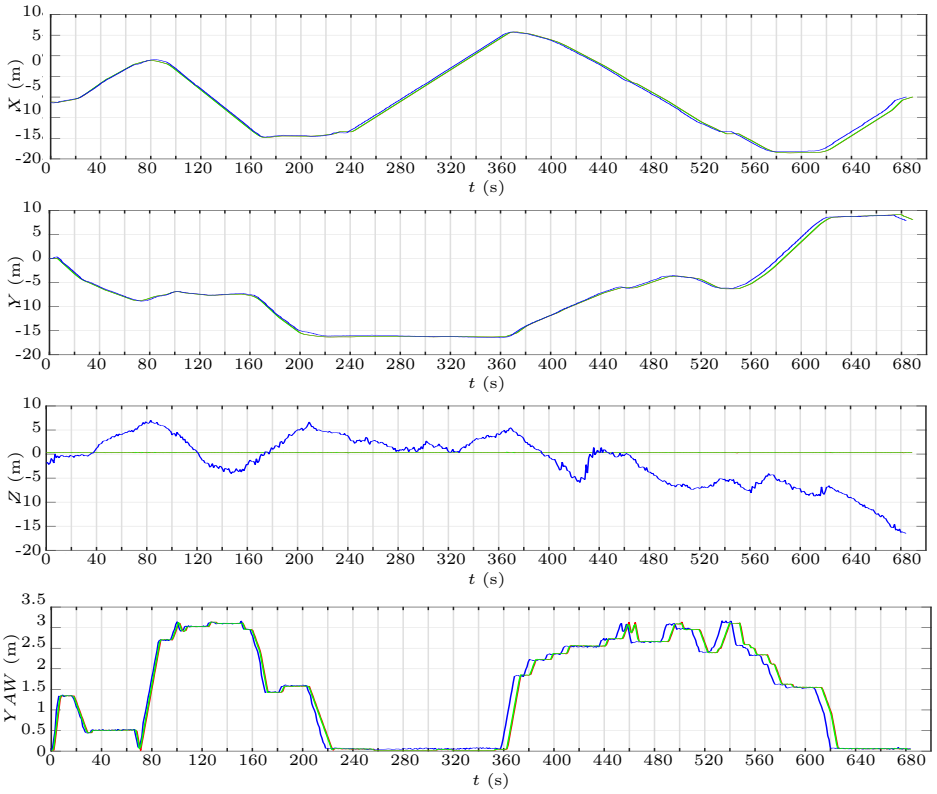


FIGURE 3.19: The plots show the trajectories for each axis and in the last one there are the yaw angles in a large map

3.4 Average position error

The average position error is a performance metric which shows how much the estimated position from the SLAM algorithm differs from the actual position. This is calculated by computing the difference between the estimated position and actual position at each time instance, and computing the mean.

In the following table we can see the RMSE for each algorithm. Hector SLAM has a good value of RMSE in all the simulations. Although, LeGO-LOAM corrects its estimate with the loop closure after a while it has wrong behaviour again. We notice this error by observing the RMSE on the z-axis.

Root Mean Square Error		
	Hector Mapping	LeGo Loam
x[m]	0.1145	0.6588
y[m]	0.0703	0.5581
z[m]	0.0022	0.2419
yaw[rad]	0.3157	0.7346

Table 3.7: RMSE for a small map

Root Mean Square Error		
	Hector Mapping	LeGo Loam
x[m]	0.3337	2.1316
y[m]	0.1410	1.1844
z[m]	0.0019	0.5490
yaw[rad]	4.4908	4.4988

Table 3.8: RMSE for a medium map

Root Mean Square Error		
	Hector Mapping	LeGo Loam
x[m]	0.2365	1.0397
y[m]	0.1561	1.0426
z[m]	0.0022	0.3586
yaw[rad]	0.9524	0.9457

Table 3.9: RMSE for a large map

3.5 Average CPU Load

The Table 3.10 shows tha CPU-load during the simulations. As we said previously, they are all working together. Poitcloud2Laserscan is the node used by Hector SLAM for taking a 2D representation of the Pointcloud. This should increase the effort due to this algorithm, but how we have previously, its accuracy is higher than LeGO-LOAM.

Code/CPU Usage	**Average CPU Usage**
Hector SLAM	9.9%
LeGO-LOAM	12.3%
OctoMap	8%
Pointcloud2Laserscan	15%

TABLE 3.10: CPU usage statistics in simulation environment. (100 corresponds to full usage of one core)

3.6 Conclusion

In conclusion we can say that the mapping process both in 2D and 3D is very accurate for both of them. Hector SLAM has always a good accuracy in map and path accuracy. Instead, LeGO-LOAM produces some errors in the pose estimation and this cause an error in mapping process. The final tables in the large map accuracy test depicts the overall errors. Even though, the z-axis error is less than the others, from the plots we can see that it is the worse estimate. We conclude that, as we said before, Hector SLAM produces the best map because Octomap is using the whole pointcloud and the pose estimation from the algorithm is very accurate.

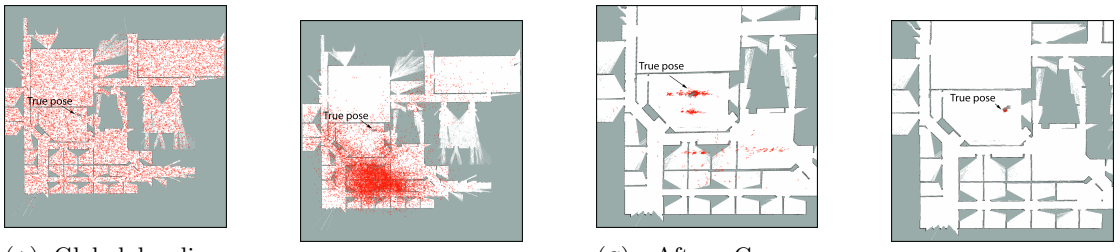
Instead LeGO-LOAM is using the whole pointcloud for calculating the pose estimate and this means that computational effort is higher and for reducing this problem it decreases the density of the elaborated pointcloud.

3.7 Self Localization

3.7.1 Simulation 2D Localization

The following sections are used to test the accuracy of the generated map with the self localization algorithm MCL.

The map is an occupancy map and it is represented as



(A) Global localization (B) Convergence (C) After Convergence (D) Final Result

FIGURE 3.20: Global localization based on AMCL in 3D

- An image showing the floor plans of the environment
- A configuration file (yaml) that gives meta information about the map (origin, size of a pixel in real world)

We have tested both a map produced by Hector SLAM. The Hector SLAM's 2D occupancy grid map is more clear than the LeGOs one. In Fig. 1.5 the red arrows represents the pose particle, the white line, as we have seen previously is the laserscan converted from the LIDAR by Hector SLAM and the 2D map of the environment is the 2D occupancy grid map of the overall simulated world provided by Hector SLAM.

The scenes in Fig. 3.20 display the evolution of 20.000 particles during a global localization experiment. After the initialization, the particles were distributed uniformly in the free space.

The procedure to find the real pose of the robot is structured as follow:

- Convergence: in 3s
- After convergence: in 5s
- Final result: in 8s

3.7.2 Simulation 3D Localization

We applied Monte Carlo localization to globally determine and reliably track a legged robot's 6D pose, consisting of the 3D position and the three rotation angles.

The method presented in [3] is able to accurately estimate the 6D pose of the HyQReal's base link while walking.

A full 3D occupancy grid map is necessary since the map needs to encode both occupied and free volumes. During the simulation, we used the octree map made by LeGO-LOAM.

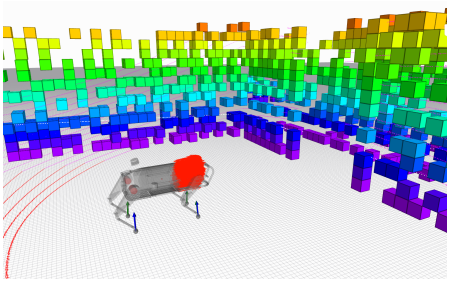
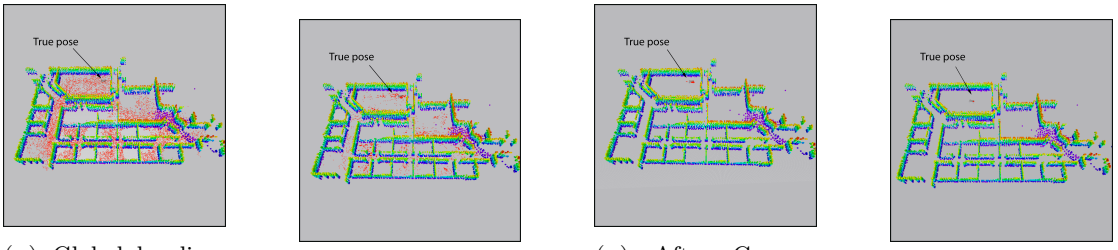


FIGURE 3.21: Particle distribution centred on the baselink



(A) Global localization (B) Convergence (C) After Convergence (D) Final Result

FIGURE 3.22: Global localization based on AMCL in 6D

Fig. 3.22 displays the same evolution of particles that we have seen previously during the global localization. After initialization, the particles were distributed uniformly in the free space on different levels.

We can see, the distribution quickly converged to the baselink of the robot. The procedure to find the real pose of the robot is structured as follow:

- Convergence: in 5s
- After convergence: in 7s
- Final result: in 10s

Obviously, global localization requires more particles than pose tracking. However, once the initial particle cloud has converged, the robot's pose can be tracked using fewer particles, as shown in Fig. 3.22.

3.8 Conclusion

In conclusion, the self localization method allows a quadrupedal robot to find its position in a known map. This procedure allows the robot to solve kidnapping problems, when it loses the orientation, it can repeat the global localization procedure and find the pose. From the simulations it is clear that both algorithms with good probability reach an

accurate final result. However, in order to have fast and versatile motions, a quadruped robot needs a 6D algorithm that allows it to obtain more detailed information.

Chapter 4

Experiments

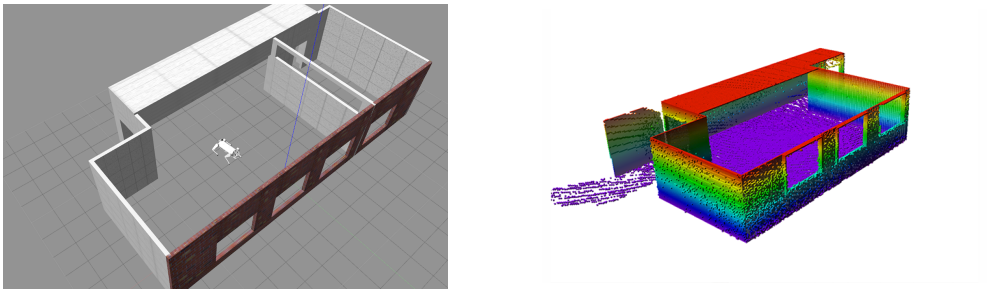
4.1 Indoor Environment

Starting from the blueprints of the environment, actually our LAB, we built a model in Gazebo. Just for visualization the following 3D model is without the ceiling, see Fig. 4.1a.

The obtained ground truth for the benchmarking results are the following in Fig. 4.1b.

The main problem during this experiment was the presence of many objects in the room. We created the model in Gazebo as accurate as possible but in the real world we had desks, people and other objects. Besides we had mounted a safety crane for avoiding accidental damages on the robot that caused a lot of interferences during the SLAM procedure.

In Fig. 4.2 we can see the filtered pointcloud that has been removed during the experiment. It is a 3D filter implemented from the *pcl_ros* library, it is named *"CropBox"* filter. We did the same also in 2D with a filter on the LaserScan, it is the *laserscan_filter*



(A) Gazebo model of the MOOG lab

(B) Best approximation of the 3D map

FIGURE 4.1: Gazebo and Octomap models of the LAB

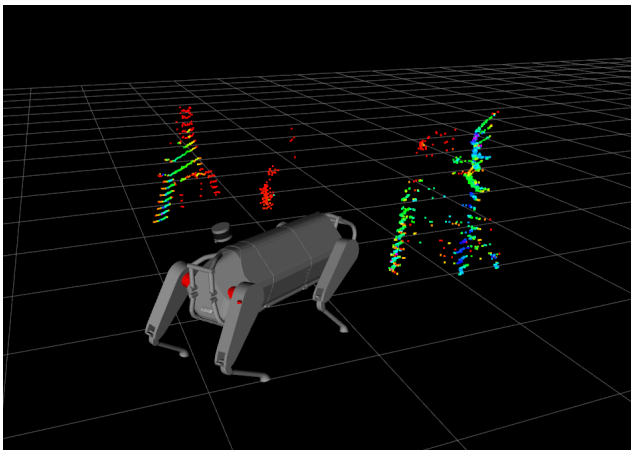
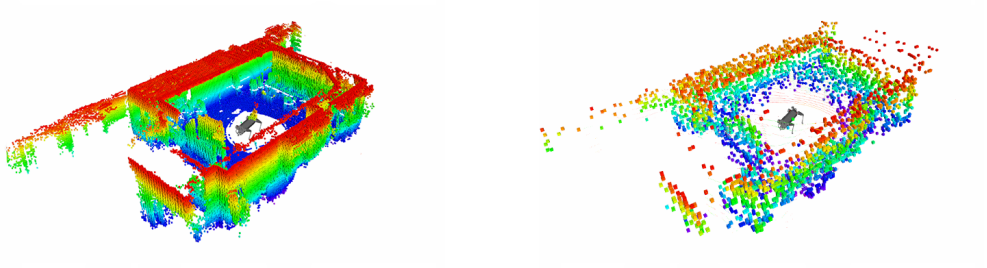


FIGURE 4.2: Filtered pointcloud.



(A) Octomap built with the estimated position made by Hector SLAM

(B) Octomap built with the estimated position made by LeGO LOAM

FIGURE 4.3: Final result at the end of the experiment for Hector SLAM and LeGO-LOAM, respectively

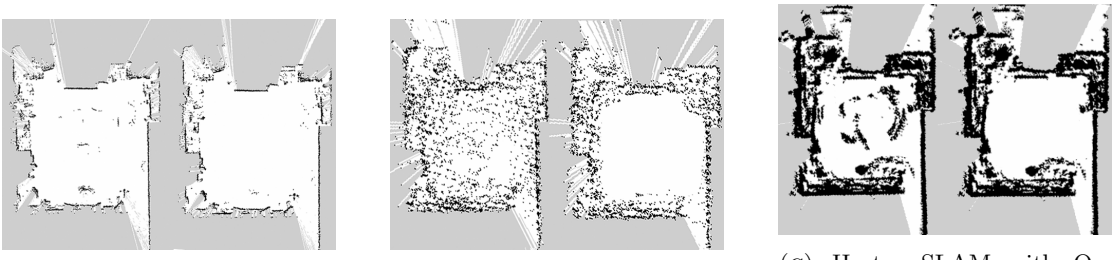
from the pcl's library. They take the pointcloud and remove all the point near to the baselink in a range of 50cm on each axis.

Hector SLAM produces an approximation of the position, and by using the velodyne points as input for the Octomap node we can produce the map in Fig. 4.3a.

In the Fig. 4.3b there is the map produced by LeGO-LOAM. The main difference between these two maps is the density of the pointcloud. Hector SLAM is calculating the pose and Octomap elaborates the pointcloud based on the variable pose. In the following sections we will analyse the metrics used before.

Number of	LeGO-LOAM	
voxels that should be occupied	149467	
occupied voxels	6676	4.4665%
occupied voxels that should be occupied	372	0.25%
occupied voxels that should be empty	3289	2.20%
empty voxels that should be occupied	9845	6.58%
unknown voxels that should be occupied	139250	93.16%

TABLE 4.1: 3D Accuracy parameters of the indoor map



(A) Hector SLAM filter (B) LeGO-LOAM (C) Hector SLAM with Octomap

FIGURE 4.4: Maps built from both Hector SLAM and LeGO-LOAM. On the left the map without filters. On the right the filtered one

In Table 4.1 we can see the result obtained from Matlab and we can observe how Hector SLAM is denser than LeGO-LOAM in an indoor map.

Moreover we can analyse the differences also in 2D. In Fig. 4.4 we have three cases, in Fig. 4.4a there are the Hector SLAM's maps with and without the laser scan filter, as we can see the presence of the crane has been deleted in the filtered map on the right. In Fig. 4.4b we have the 2D down-projected map from LeGO-LOAM, and what we can see is the presence of other objects inside the map, this is because the map is a down-projection of the 3D occupancy grid map, then it is more dense. Instead in Fig. 4.4c there is the down-projected map made by Hector SLAM by using OctoMap. The main difference we can observe is the importance of the filter, it is useful for avoiding useless data. We can analyse the accuracy of the maps in the Table 4.2. We used the 2D map generated by Hector SLAM with Octomap as ground truth. The final result shows as for the simulations, an accurate estimate for Hector SLAM with a low percentage of occupied cells that should empty, and with a 6.5% of occupied cells that should be occupied.

Number of	Hector-SLAM		LeGO-LOAM	
cells that should be occupied	26392			
occupied cells	2964	11.23%	8468	32.08%
occupied cells that should be occupied	1721	6.5%	4368	16.55%
occupied cells that should be empty	945	3.58%	3074	11.64%
empty cells that should be occupied	21942	83.13%	18375	69.62%
unknown cells that should be occupied	2729	10.34%	3649	13.82%

TABLE 4.2: 2D Accuracy parameters of the indoor map

4.1.1 Path Accuracy

In order to obtain a robot's pose as close as possible we positioned markers on the trunk of the robot. We used the Motion Capture System (MCS) from Vicon. We considered

the output from the MCS system as a ground truth for the experiment and then we compared it with the estimated trajectories from Hector SLAM and LeGO-LOAM.

The estimated trajectory from the MCS is used as ground truth and it is printed on the 2D map in Fig. 4.5

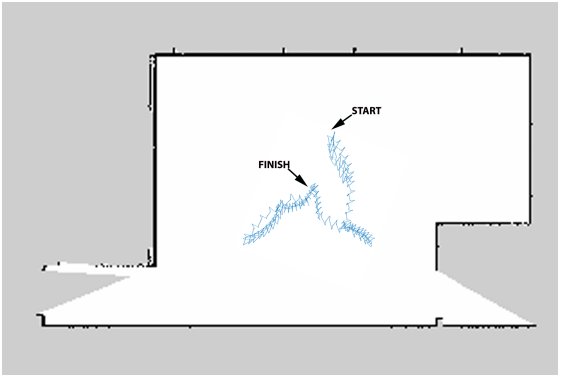


FIGURE 4.5: 2D ground truth of the indoor environment. In blue the trajectory from the MCS

The experiment has been conducted for 500s and we can see in Fig. 4.6 the overall trajectories for each algorithm. In blue we have the ground truth from the MCS, in red Hector SLAM and in green LeGO-LOAM. We can notice the presence of different holes in the MCS's estimate, this is due to the presence of the crane. The interference is caused because it hides the markers and the motion capture is not able to detect them. Furthermore, we can see that the estimates are very different from each other, expect for Hector SLAM. We can's show a RMSE due to the presence of that holes. Even though, we can graphically conclude that they are similar to each other on the x and y axes, although on the z axis LeGO-LOAM is presenting, as we have seen in the simulations, a calculation error. However, this error seems to be always under 0.1 m.

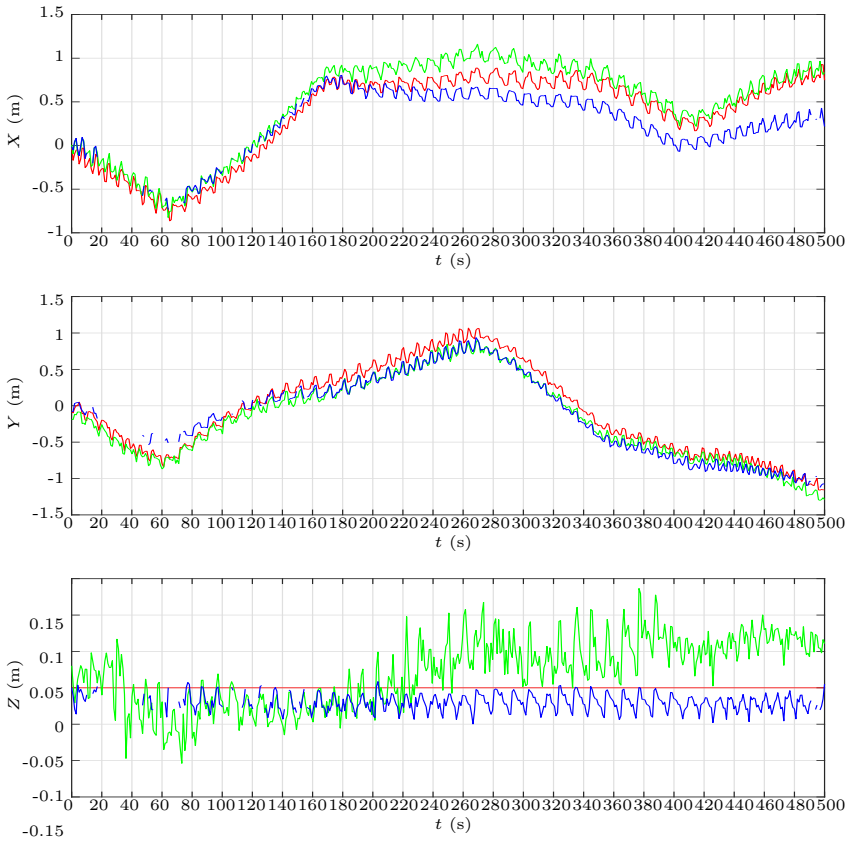
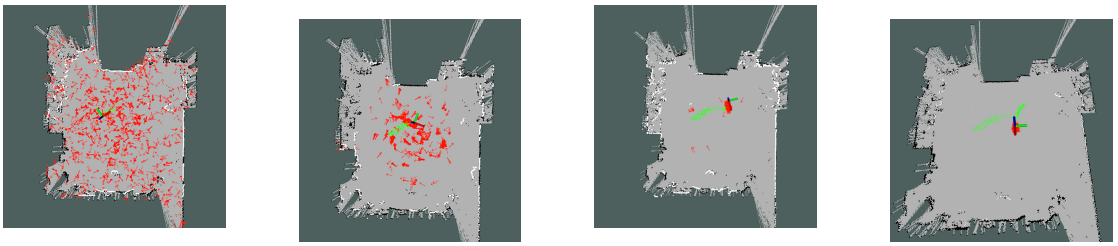


FIGURE 4.6: Trajectories for each axis. Blue is the ground truth, red is Hector SLAM and green is LeGO-LOAM.

4.1.2 3D self localization



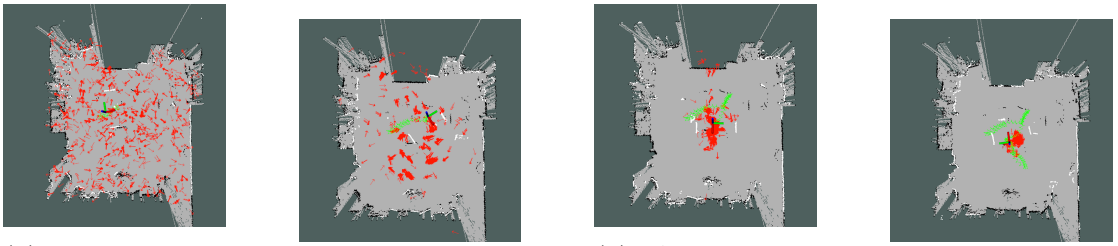
(A) Global localization (B) Convergence (C) After Convergence (D) Final Result

FIGURE 4.7: Global localization in an indoor map with the filter on the pointcloud

We have tested the 3D self localization process to determine its accuracy in a noisy map, with and without the filter. In Fig. 4.7 there is the AMCL procedure by using the filtered map made by Hector SLAM. We choose this map because it is more clear than the LeGO-LOAM's one, the convergence time is faster and the accuracy is higher.

- Convergence: in 2s
- After convergence: in 5s
- Final result: in 10s

Moreover in Fig. 4.8 we analyse the process without the filter and we can see that it is not even able to find the position at the end of the simulation.



(A) Global localization (B) Convergence (C) After Convergence (D) Final Result

FIGURE 4.8: Global localization in an indoor map without the filter on the pointcloud

4.1.3 CPU load

Finally we have tested the CPU usage during the off-line simulation. We can check them in Table 4.3.

The main effort comes from LeGO-LOAM. During the experiment it uses 153% of CPU, although the final result is better than the simulation. It has high accuracy in the mapping and localization processes.

Code/CPU Usage	Average CPU Usage
Hector SLAM	10.9%
LeGO-LOAM	153%
OctoMap	11%
Pointcloud2Laserscan	44%
CropBox Filter	6.9%

TABLE 4.3: CPU usage statistics in an indoor map. (100 corresponds to full usage of one core)

4.2 Outdoor Environment

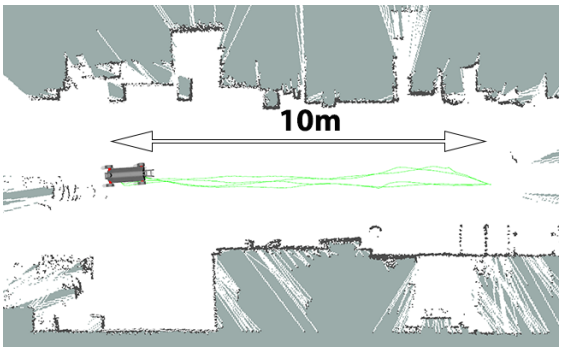


FIGURE 4.9: Trajectory travelled by HyQReal during the experiment.

The goal of this experiment was to travel in an outdoor environment for at least 100 m and to find which algorithm has the best fit between 3D mapping, 2D mapping and trajectory estimation. In this environment there are no slopes, but there are several boxes, pipes and other working tools.

In Fig. 4.9 we can see the path that we made during the experiment. The robot has travelled 10 m forward and backwards 10 times.

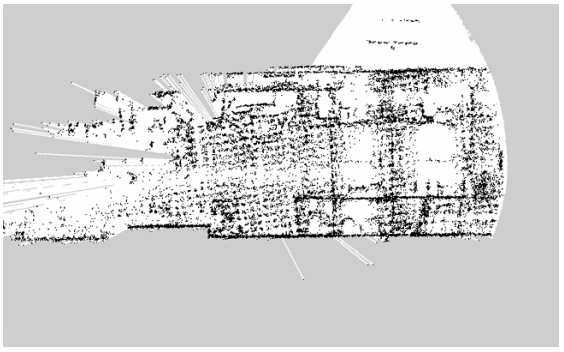
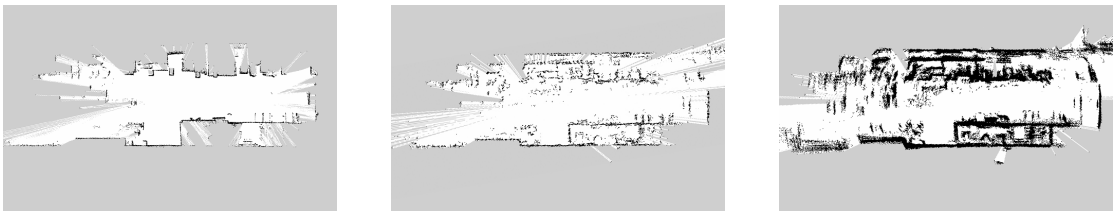


FIGURE 4.10: LeGO-LOAM's down-projected map with the ground and the ceiling

In Fig. 4.10 we can see the down-projected map made by the LeGO-LOAM. As we can see it is very noisy and for this reason in the following images the maps will be shown without the ground and the ceiling. This is just for visualization and for having a better



(A) Hector Slam (B) LeGO-LOAM (C) Hector with Octomap

FIGURE 4.11: 2D maps from Hector SLAM, LeGO-LOAM and Hector SLAM with Octomap

resolution. The 2D down-projected maps are computed from 10 cm up to 1.50 m as for LeGO-LOAM as for Hector SLAM.

4.2.1 Mapping Accuracy

We have seen in the previous section that the presence of the filter is necessary in a small environment. But when we tested Hector SLAM with and without the filter, we noticed that in this situation the filter on the LaserScan is not necessary. In Fig. 4.11 (a) we can see the final 2D map made by Hector SLAM. We made it without the laserscan filter, and we can conclude that if the robot travels several times on the same path Hector SLAM is able to remove moving objects.

For LeGO-LOAM the case is different. LeGO-LOAM is not always able to remove moving objects, it recognizes the safety crane that is beside the robot, the pipes for the power supply and the external pumps.
In order to avoid the calculation of these objects we have introduced the CropBox filter mentioned before.

In Fig. 4.12 we can see the pointcloud that we deleted for the mapping process.

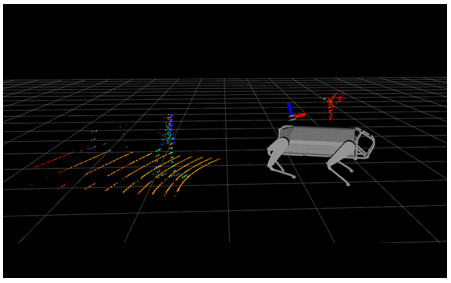
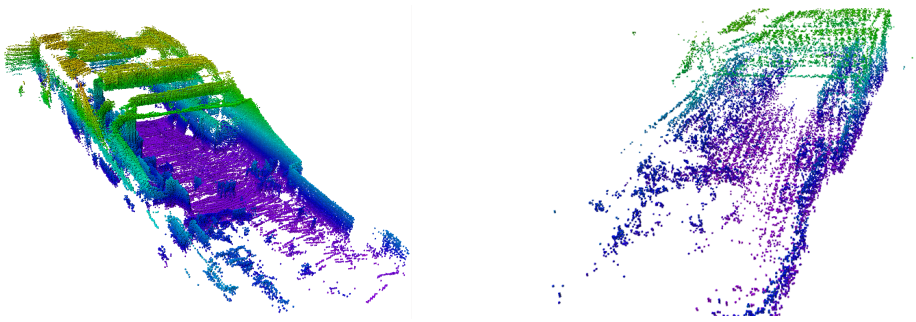


FIGURE 4.12: On the back of the robot, there is the dromedario, it is a safety protection for the arm, close to it there are pipes and behind it there is also a colleague that maintains the crane

After the implementation of the filter we can see the final 2D map made by LeGO-LOAM in Fig. 4.11b. As we have done for the previous experiment, we have mixed the 2D pose



(A) Hector SLAM with Octomap (B) LeGO-LOAM's 3D map

FIGURE 4.13: 3D maps built with Octomap

estimation algorithm (Hector SLAM) with a 3D mapping process (Octomap). The final result in a two dimensional point of view is shown in Fig. 4.11c. It is the best result in terms of 2D/3D map accuracy and CPU usage. The only nodes were Hector SLAM and Octomap, and as we can see in Table 4.4 the CPU usage from Hector SLAM is less than LeGO-LOAM.

Code/CPU Usage	**Average CPU Usage**
Hector SLAM	11.98%
LeGO-LOAM	153.2%
OctoMap	52%
Pointcloud2Laserscan	44.2%
CropBox Filter	7%

TABLE 4.4: CPU usage statistics in an outdoor map. (100 corresponds to full usage of one core)

In Table 4.5 we see the differences between the two results in terms of 2D map accuracy. Their behaviour are similar to the simulations. The percentage of occupied cells that should be empty represent the error in the mapping process. Even though Hector SLAM is less dense than LeGO-LOAM it is always more accurate.

Number of	Hector-SLAM		LeGO-LOAM	
cells that should be occupied	25275			
occupied cells	2640	10.44%	4719	18.67%
occupied cells that should be occupied	1323	5.23%	2434	9.63%
occupied cells that should be empty	975	3.85%	1728	6.83%
empty cells that should be occupied	22270	88.11%	20087	79.47%
unknown cells that should be occupied	1682	6.65%	2754	10.89%

TABLE 4.5: 2D Accuracy parameters for the outdoor map

In Fig. 4.13a we can see the 3D map made by Hector SLAM, its representation as we said is better than LeGO-LOAM. As for the preciouses maps, the ground is represented by the purple voxels, instead the ceiling is green. As we have said before the environment is not very clear, and we can see that the green voxels are pipes and cables.

In Fig. 4.13b we have the 3D Octomap made by LeGO-LOAM. The main difference from the previous one is the density of the pointcloud. Its representation is due to the fact that the algorithm is calculating the odometry by using the whole 3D pointcloud. By doing so, it must to reduce the time by decreasing the acquired data. First it converts the pointcloud in a voxel grid with a lower density, and, obviously the output has a lot of empty points. For increase the quality of this map we should move the robot in different orientations.

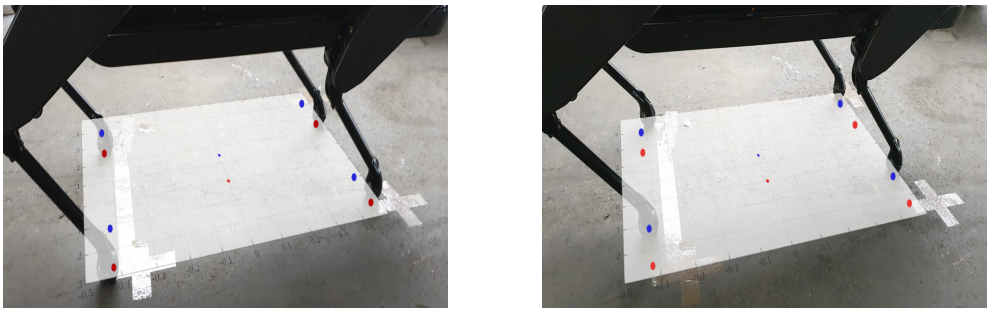
In Table 4.6 we see the differences between the two results in terms of 3D map accuracy.

Number of	LeGO-LOAM	
voxels that should be occupied	41187	
occupied voxels	7121	17.28%
occupied voxels that should be occupied	44	0.1%
occupied voxels that should be empty	702	1.7%
empty voxels that should be occupied	4195	10.18%
unknown voxels that should be occupied	36948	89.70%

TABLE 4.6: 3D Accuracy parameters for the outdoor map

4.2.2 Path Accuracy

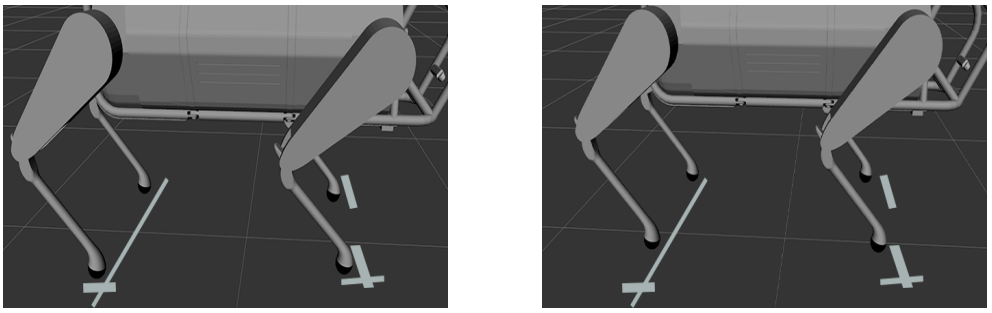
Due to the fact that we don't have a MCS in this environment we can't have a ground truth in order to see an error between the real and the simulated trajectory. For solving this situation we have performed an error on the estimated position of the feet.



(A) starting point (B) ending point

FIGURE 4.14: Real position of the feet during the experiment

In Fig. 4.14 we have two pictures that show the position of the feet respect to 4 markers on the ground. As we can read from the caption, the picture on the left shows the position of the feet before starting, and the right one the final position at the end of the experiment.



(A) starting point (B) ending point

FIGURE 4.15: Position of the feet during the simulation

We have performed the simulation by using the estimated position of the trunk made by Hector SLAM. Hence the state estimator that we have on the robot is working only for the positions of the legs.

In Fig. 4.15 we can see the robot in the same situation that we have seen in Figure 4.14, it shows the feet positions before and after the simulation.

In order to see in detail the error between the starting position and the ending position we took the data of the feet from the state estimator.
In Fig. 4.16 we have plotted the estimated positions made by Hector SLAM. The blue dots are the initial positions of the feet, the red one are the final positions and it is oriented with the front on the positive direction of X. Moreover we have estimated the displacement on real robot from the picture. We made an estimate based on a proportion with respect to the tapes on the ground. Due to the lower precision of the data the circle is as big as the range of accuracy, it is calculated in a range of [-3,3] cm. Finally we can conclude that the real feet are very close to the estimated ones.

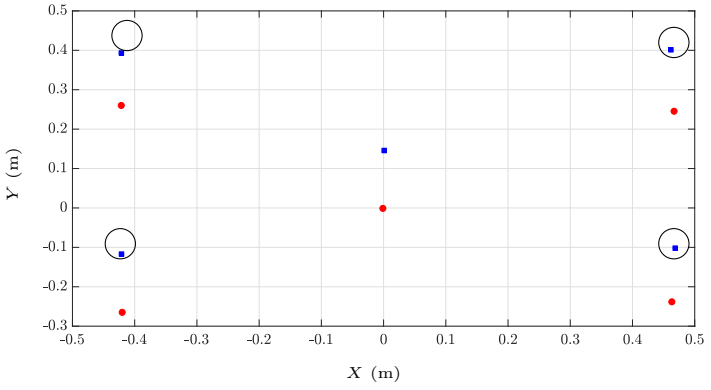


FIGURE 4.16: Red dots are the estimated initial positions, the blue dots are the estimated final positions. The circles are the real feet positions and the dots in the mile represent the position the base link respectively

Moreover in Fig. 4.17 we have calculated the estimated trajectories for each axis. In red Hector SLAM, and green LeGO-LOAM.

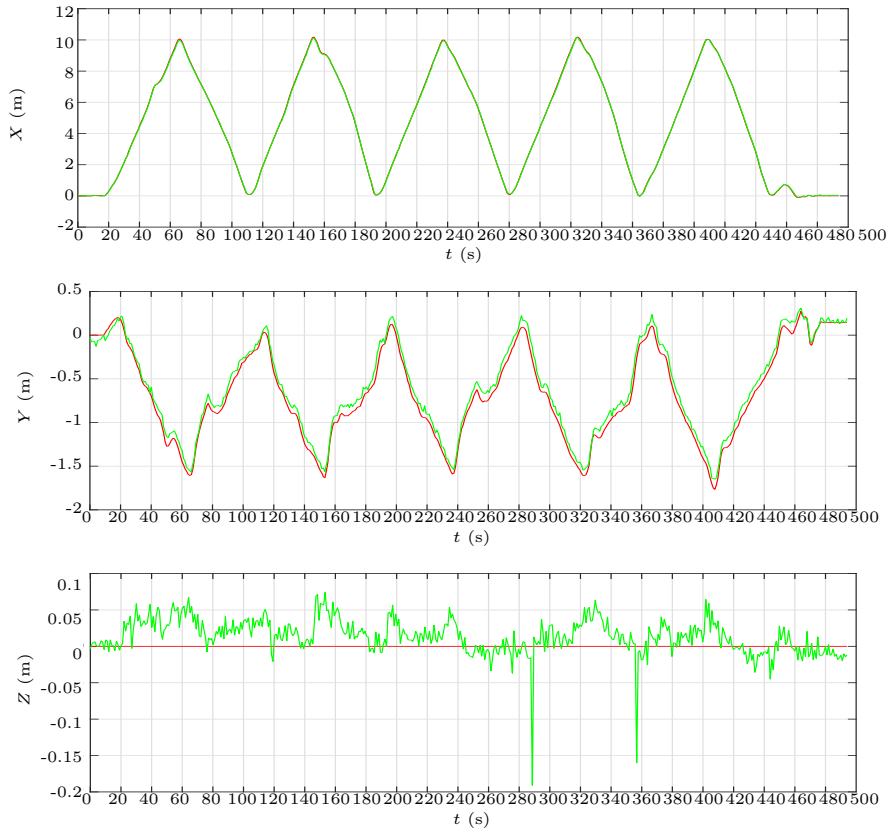


FIGURE 4.17: The overall trajectories for each axis

We can analyse that on the x axis there is no difference between them, instead on the y axis there is a slight difference. We have to say that the x axis is directed forward then for the right hand rule, y is directed to right and the z is looking upward. The estimation on the z axis, as we said several times, for Hector SLAM is zero because it is a two dimensional SLAM. Instead we can observe that LeGO-LOAM is very accurate because when HyQReal moves the trunk in a range of 5 cm.

4.3 Conclusion

We can finally conclude this experiment by saying that Hector SLAM has the low computational effort, the best accuracy in map building and the best accuracy in pose estimation. Instead LeGO-LOAM, has often leakage in the pose estimation, i.e. in the z plot in Fig. 4.17, and this situation could generate a growing error.

However we didn't have the possibility the test the algorithms in non planar environments, hence we can't formulate a global conclusion.

Chapter 5

Conclusion and future work

In this work we have analysed two SLAM algorithms. We decided to analyse the behaviour of Hector SLAM and LeGO-LOAM. We adopted a metric for the accuracy's analysis. For each algorithm we took into account, the mapping accuracy, the estimated position, the mean square error and the computational load. In addition, two self localization methods based on the Monte Carlo localization (MCL) algorithm were verified. These allow to obtain an estimated position as precise as the generated map is. The simulations show that LeGO-LOAM requires an high computational load, higher then Hector SLAM. The higher computation load causes a slowdown in simulation. The extra computation time required by Lego-LOAM versus Hector SLAM is caused by the difference of analysed data. Hector SLAM calculates a position estimate and it generates a 2D map using a single laser beam. Instead, LeGO-LOAM uses a whole point cloud, and for a Velodyne-16 Puck Lite each pointcloud is up to 600,000 points per second. To reduce the calculation time, it introduces a filter to eliminate a part of the informations. In the large map simulation we can see how the algorithms work in the biggest environment. The final result shows a RMSE for Hector SLAM lower than LeGO-LOAM. Values between 0.14 and 0.33 are produced for both the x and y axis. LeGO-LOAM estimates the robot position with a low error in the small and medium map, but at the end of the simulation it accumulates an error higher than 50cm.

Regarding the mapping accuracy, Hector SLAM produces a 2D map using only one laser beam and this does not take into account all the obstacles. The density of the generated 2D map is lower than that of LeGO-LOAM, and this does not allow a quadruped robot to fully identify the surrounding environment. During self operated operations, the quadruped robot needs a dense map and an high pose accuracy. For teleoperated operations a legged robot needs to obtain a map estimate as accurate as possible. In addition, if the robot operates in a known environment, thanks to algorithms such as

MCL, it has to obtain an estimate of the position. From simulations we conclude the robot is able to find its position both in 2D and 3D pre-built maps by Hector SLAM and LeGO-LOAM.

In the experiment section, their behaviour on the HyQReal robot was verified. The experiments were carried out in two different environments, the first was a laboratory, which is an indoor room, instead the second was larger and partly outdoors place.

For the first experiment, we had a motion capture system available which allowed us to obtain the position of the robot in real time. Moreover we created a 3D model of the laboratory starting from the floor plans. This information was used to compare and to elaborate the accuracy of the map and the estimated position. During both experiments we had an interference with the robot's safety structure, which prevented it from accidentally falling. To overcome this problem, we have implemented both 2D and 3D filters that delete unnecessary informations.

For the second experiment the robot travelled more than 100 m with the aim of verifying the accuracy of the algorithms over long distances.

In conclusion LeGO-LOAM still has some calculation errors as seen in simulation. However, the accuracy of the estimated position has improved, and the map turns out to be denser. This allows us to consider LeGO-LOAM more than Hector SLAM. In order to operate in variable terrain environments, a quadruped robot requires a 3D SLAM algorithm capable of identifying closed loops and obtaining an estimate of the position in 6 dimensions.

Furthermore, we didn't have a motion capture system, hence we calculated the errors on feet position for both the real and estimated ones on the basis of some markers placed on the ground.

We can conclude that LeGO-LOAM has an high computational load and sometimes it loses accuracy in the mapping and pose estimation. Instead Hector SLAM achieves the best estimate in each simulation and experiment. However, this SLAM algorithm is conceived in planar environments.

A first future work it to simulate Hector SLAM with Octomap in variable terrains, to verify its robustness. It can already be concluded that even though LeGO-LOAM has problems in processing, many works about SLAM in variable terrains show that it very accurate. These algorithms have been conceived with the task of allowing the robot to orient itself in unknown environments. Methods such as Monte Carlo allow to a SLAM algorithm to find its position in such environments. A future improvement is to develop, starting from LeGO-LOAM, a better algorithm in order to reduce the computational

effort and to increase the accuracy. A future work could be to develop an algorithm able to take the information from Monte Carlo and to automatically move the robot in that position inside the simulator. Furthermore, on the front and on the back of the robot we have several cameras, the next step is the sensor's fusion. LIDAR is able to detect the environment in 360° but the robot needs to have more informations in order to have a safe locomotion. The idea is to design an algorithm which takes into account the pointcloud generated from both the LIDAR and cameras.

The ultimate goal of this integration could be construction of a navigable 3D map of an environment, in which the robot can view all the objects already labelled and classified. This classification would allow the robot to perform obstacle identification, object picking and autonomous navigation tasks.

Appendix A

Hardware/Software Description

A.1 HyQReal and DLS's framework

We perform simulations using the quadruped robot HyQReal [4]. It is 0.9 m tall, weighs 130 kg (onboard hydraulics and battery). This means that when fast motions are required, swing legs play a significant role in the robot dynamics. In addition to the default sensors such as IMU and cameras, our robot is equipped with a Velodyne VLP-16 Lidar. The 3D Lidar is mounted in a modified position with an angle of 15° on the y axis on the back of the robot, providing a field of view of 360°, see Fig. A.1.

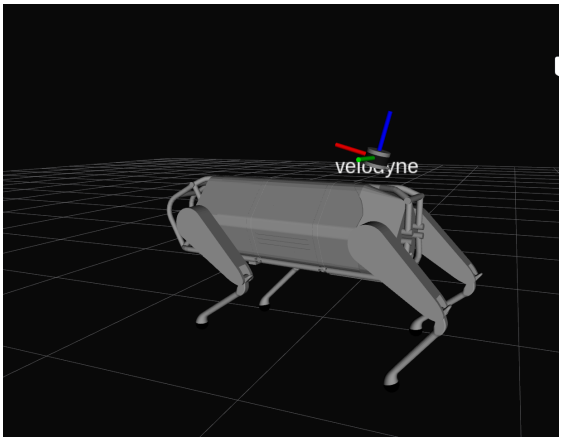


FIGURE A.1: HyQReal in simulation environment with the velodyne on its back

A.2 Velodyne-VLP 16 Puck Lite

Designed for applications that demand a lower weight, the Puck LITE retains the Puck sensor's surround view and best-in-class performance. The Puck LITE is perfect for

use with drones/UAVs, backpacks and other applications requiring reliability and less weight. The Puck LITE has a range of 100 m with dual return mode to capture greater detail in the 3D image with a low power consumption. It supports 16 channels and generates approximately 300,000 points/second from a 360° horizontal field of view and a 30° vertical field of view (±15° from the horizon) The Puck LITE has no visible rotating parts and is encapsulated in a package that allows it to operate over a wide temperature range and environmental conditions.

A.3 Robot Operating System

ROS [5] is an open-source, meta-operating system for robots. It provides the services that would be expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. ROS is not a real-time framework, though it is possible to integrate ROS with real-time code.

Its architecture is based on Peer-to-Peer (P2P) communication between different nodes. These nodes can be defined as programs that perform various tasks and are running on one or more computers being part of a network.

The final transform tree utilised on HyQReal is shown in Fig. A.2.

The employed ROS distribution for this project is ROS Kinetic, released on May 2016.

A.3.1 Gazebo

Gazebo is an open-source 3D robotics simulator. It provides realistic rendering of environments including high-quality lighting, shadows, and textures. It can model sensors that "see" the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors, etc.

A.3.2 Rviz

Rviz is a 3D visualizer for the Robot Operating System (ROS) framework. It offers a view of the robot model, it acquires sensor information from the robot sensors and it

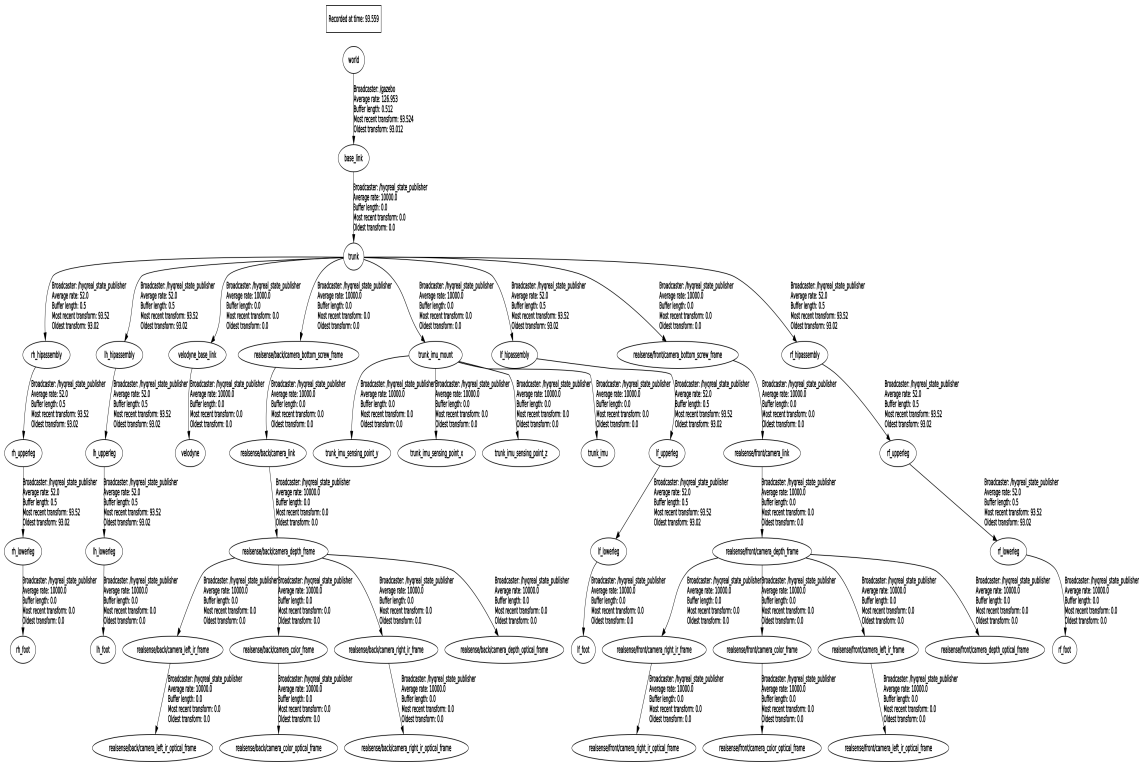


FIGURE A.2: HyQReal's tf tree

reproduces the acquired data. It can display data from video cameras, lasers, 3D and 2D devices, including images and point clouds. To perform the tasks, Rviz must be open and connected to a running simulation process, i.e. Gazebo.

To be able to visualize the information published in Rviz, it is necessary to correctly configure the Global Options: Key parameters:

- **Fixed frame**: Indicates the name of the frame used as reference for all the other frames.
- **Robot Model**: In order to visualize the Robot Model according to its description from the URDF model we need to add "Robot Model" from the list.
- **TF**: A most important plugins allow us to visualize the position and the orientation of all the frames that compose the TF Hierarchy.

In Fig. A.3 we can see the model of the Robot, the generated pointcloud from the Velodyne and the options in Rviz presented before.

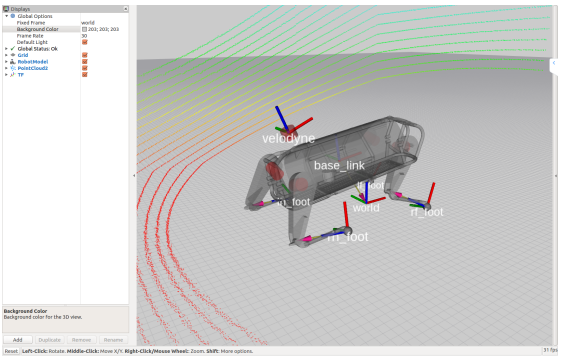


FIGURE A.3: HyQReal in a real and simulated environment

A.4 Vicon's Motion Capture System

Motion capture (mocap) is the process of recording the movement of objects or people. The technology originated in the life science market for gait analysis but is now used widely by VFX studios, sports therapists, neuroscientists, and for validation and control of computer vision and robotics. There are many different approaches to motion capture.

- **Optical-Passive**: This technique uses retroreflective markers that are tracked by infrared cameras. It is the most flexible and common method used in the industry.
- **Optical-Active**: This technique uses LED markers that emit light that are tracked by special cameras. Because of this they need a battery or charger of some kind.
- **Video/Markerless**:This technique does not require markers and instead relies on software to track the subject's movement. Varying tracking methods yield different results, but real-time and final data error ranges tend to be larger than marker-based solutions.
- **Inertial**:This technique does not require cameras except as a localization tool. Inertial sensors (sometimes known as IMUs) are worn by the subject and the data from the sensors is transmitted wirelessly to a computer or smart device.

Passive optical motion capture is the most accurate, flexible and common type of motion capture and is a major technology for Vicon.

Bibliography

[1] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. pages 4758–4765, 10 2018. doi: 10.1109/IROS.2018.8594299.

[2] G. Fink, M. Franke, A. F. Lynch, K. Röbenack, and B. Godbolt. Observer design for visual inertial slam scale on a quadrotor uav. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 830–839, 2017.

[3] A. Hornung, K. M. Wurm, and M. Bennewitz. Humanoid robot localization in complex indoor environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1690–1695, 2010.

[4] C. Semini, V. Barasuol, M. Focchi, C. Boelens, M. Emara, S. Casella, O. Villarreal, R. Orsolino, G. Fink, S. Fahmi, G. Medrano-Cerda, and D.G. Caldwell. Brief introduction to the quadruped robot hyqreal. *Istituto di Robotica e Macchine Intelligenti (I-RIM)*, 2019.

[5] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL https://www.ros.org.

[6] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[7] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[8] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing icp variants on real-world data sets. *Autonomous Robots*, 04 2013. doi: 10.1007/s10514-013-9327-2.

[9] Peter Zhang, Evangelous Millos, and Jason Gu. *General Concept of 3D SLAM*. 05 2009. ISBN 978-953-307-001-8. doi: 10.5772/6993.

[10] J. Alexandersson and O. Nordin. Implementation of slam algorithms in a small-scale vehicle using model-based development. 2017.

[11] Hans Moravec. Robot spatial perception by stereoscopic vision and 3d evidence grids," robotics institute. 04 2011.

[12] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. doi: 10.1007/s10514-012-9321-0. URL http://octomap.github.com. Software available at http://octomap.github.com.

[13] Tae Nam, Jae Shim, and Young Cho. A 2.5d map-based mobile robot localization via cooperation of aerial and ground robots. *Sensors*, 17:2730, 11 2017. doi: 10.3390/s17122730.

[14] S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MA, USA: MIT Press, 2005.

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

[16] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. 07 2014. doi: 10.15607/RSS.2014.X.007.

[17] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41:401–416, 02 2017. doi: 10.1007/s10514-016-9548-2.

[18] Seungpyo Hong, Heedong Ko, and Jinwook Kim. Vicp: Velocity updating iterative closest point algorithm. pages 1893 – 1898, 06 2010. doi: 10.1109/ROBOT.2010.5509312.

[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. In *2011 IEEE International Conference on Robotics and Automation*, pages 3281–3288, 2011.

[20] S. Thrun, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. volume 128, page 99–141, Artificial Intelligence, 2001.

[21] B. F. Zhang, P. Meng, and H. Yue. Overview of mobile robot location method. volume 22, page 250–250, Shandong Industrial Technology, 2014.

[22] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.